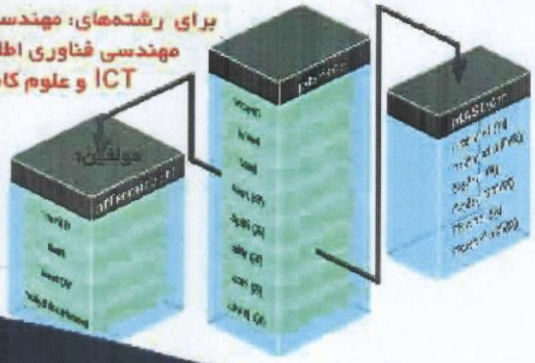
ساختمان داده‌ها

با C++

به همراه ۴۵۰ تست کارشناسی ارشد با حل تشریحی

برای رشته‌های: مهندسی کامپیوتر،
مهندسی فناوری اطلاعات،
ICT و علوم کامپیوتر



مولفین
مهندس رمضان عباس نژاد ورزی
مهندس علی رضا اسمعیلی
مهندس هادی کامفر

برخی از عناوین مهم

ساختار داده‌ها، الگوریتم‌ها و پیچیدگی
آرایه‌ها و کاربرد آن‌ها
پشته، صف و کاربرد آن‌ها
لیست پیوندی و کاربرد آن
درخت و کاربرد آن
گراف و کاربرد آن
مرتب سازی و انواع آن

ساختمان داده‌ها
با
C++

تالیف:

مهندس رمضان عباس‌نژادورزی
مهندس علی‌رضا اسمعیلی – مهندس هادی کامفر

فن‌آوری نوین

سرشناسه	:	عباس‌نژاد ورزی، رمضان، ۱۳٤۸ -
عنوان و نام پدیدآور	:	ساختمان داده‌ها با C++/تالیف رمضان عباس‌نـژادورزی، علـی رضـا اسـمعیلی، هادی کامفر .
مشخصات نشر	:	بابل:فن‌آوری نوین، ۱۳۹۲
مشخصات ظاهری	:	۳۲۸ص.:مصور، جدول.
شابک	:	۱٦٥۰۰۰ ریال:۱-۹-۹۲۲٥٤-٦۰۰-۹۷۸
وضعیت فهرست نویسی	:	فیپا
موضوع	:	سی ++(زبان برنامه نویسی کامپیوتر)
-∞	:	ساختار داده‌ها
شناسه افزوده	:	اسمعیلی، علی رضا، ۱۳٥۱
شناسه افزوده	:	کامفر، هادی، ۱۳٦۳
رده بندی کنگره	:	QA۷٦/۹س۲ع۲۳ ۱۳۹۲
رده بندی دیویی	:	۰۰٥/۷۳
شماره کتابشناسی ملی	:	۳۲٦٤۷۷۳

فن‌آوری نوین بابل، کدپستی ٤۷۱٦۷-۷۳٤٤۸ www.fanavarienovin.net تلفن: ۲۲٥٦٦۸۷-۰۱۱۱

ساختمان داده‌ها با C++

تألیف: مهندس رمضان عباس‌نژادورزی – مهندس علی رضا اسمعیلی- مهندس هادی کامفر

نوبت چاپ: چاپ اول

سال چاپ: پاییز ۱۳۹۲

شمارگان:۱۰۰۰ جلد

قیمت: ۱٦٥۰۰ تومان

نام چاپخانه و صحافی: فرنگاررنگ

شابک: ۱-۹-۹۲۲٥٤-٦۰۰-۹۷۸

نشانی ناشر: بابل، چهارراه نواب، کاظم‌بیگی، جنب حسینیه منصور کاظم‌بیگی، طبقه همکف

ویراستار: مهندس فاطمه عبدی

طراح جلد: کانون آگهی و تبلیغات آبان (مهندس احمد فرجی)

تهران، خ اردیبهشت، نبش وحید نظری، پلاک ۱٤۲ تلفکس: ٦٦٤۰۰۲۲۰-٦٦٤۰۰۱٤٤

فهرست مطالب

فصل اول: ساختار داده‌ها، الگوریتم‌ها و پیچیدگی..... ۹

۱-۱. ساختمان داده‌ها..... ۹

۱-۱-۱. ساختمان داده‌های ایستا..... ۹

۱-۱-۱. ساختمان داده‌های پویا..... ۱۱

۱-۱-۱. ساختمان داده‌های نیمه ایستا..... ۱۱

۱-۲. الگوریتم..... ۱۲

۱-۲-۱. کارایی الگوریتم..... ۱۳

۱-۲-۱. مرتبه اجرائی..... ۱۷

۱-۳. توابع بازگشتی..... ۱۹

۱-۴. مسائل حل شده..... ۲۴

۱-۵. تست‌های ارشد ساختار داده‌ها، الگوریتم‌ها و پیچیدگی..... ۳۵

۱-۶. پاسخ تشریحی تست‌های ارشد ساختار داده-ها، الگوریتم‌ها و پیچیدگی..... ۴۴

فصل دوم: آرایه..... ۵۴

۲ -۱. آرایه یک بعدی(بردار)..... ۵۴

۲ -۱-۱. مفاهیم کلی و پیاده‌سازی آرایه یک بعدی..... ۵۴

۲ -۱-۲. عملیات مهم بر روی آرایه یک بعدی(لیست خطی)..... ۵۵

۲ -۱-۳. جست‌وجو در آرایه..... ۵۸

۲ -۲. آرایه دو بعدی و چند بعدی..... ۶۵

۲ -۲-۱. مفاهیم کلی و پیاده‌سازی آرایه دو بعدی و چند بعدی..... ۶۵

۲ -۲-۲. عملیات مهم بر روی آرایه دوبعدی بعدی(ماتریس)..... ۶۸

۲ -۲-۳. چند جمله‌ای..... ۷۳

۲ -۲-۴. رشته..... ۷۴

۲ – ۳. مسائل حل شده..... ۷۵

۲ -۴. تست‌های ارشد آرایه..... ۷۶

۲ – ۵. پاسخ تشریحی تست‌های ارشد آرایه..... ۸۳

فصل سوم: پشته و صف ۸۹

۳ – ۱. پشته..... ۸۹

۳ -۱-۱. پیاده‌سازی پشته با آرایه..... ۸۹

۳ -۱-۲. عملیاتی که بر روی پشته انجام می‌شود ۹۰

۳ -۱-۳.ساختار تعریف کلاس Stack..... ۹۰

۳ -۱-۴. کاربردهای پشته..... ۹۳

۳ -۱-۵. پشته چندگانه..... ۱۰۱

۳ -۲. صف..... ۱۰۲

۳ -۲-۱. پیاده‌سازی صف با آرایه..... ۱۰۲

۳ -۲-۲. عملیاتی که بر روی صف انجام می‌شود..... ۱۰۳

۳ -۲-۳. مشکل صف معمولی..... ۱۰۴

۳ -۲-۴. راه حل مشکل صف خطی حذف و جا به جایی عناصر..... ۱۰۵

۳ -۲-۵. حل مشکل صف خطی با استفاده از صف حلقوی..... ۱۰۵

۳ -۲-۶. طراحی صف به صورت شی‌گرا..... ۱۰۸

۳ -۲-۷.چند نکته در مورد صف و پشته ۱۰۹

۳ – ۳. مسائل حل شده..... ۱۱۴

۳ – ۴. تست‌های ارشد صف و پشته..... ۱۱۶

۳ -۵. پاسخ تشریحی تست‌های ارشد پشته وصف..... ۱۲۱

فصل چهارم: لیست پیوندی ۱۲۸

۴ – ۱. لیست تک پیوندی..... ۱۳۱

۴ – ۱ – ۱. تعریف ساختار هر گره در لیست پیوندی..... ۱۳۱

۲ – ۱ – ۴. تعریف اشاره‌گری برای نگهداری آدرس اولین گره لیست پیوندی..... ۱۳۲
۳ – ۱ – ۴. پیاده‌سازی عملیات اساسی روی لیست پیوندی..... ۱۳۲
۲ – ۴. لیست پیوندی حلقوی..... ۱۳۶
۳ – ۴. لیست دو پیوندی..... ۱۴۰
۱ – ۳ – ۴. تعریف کلاس‌های لیست دو پیوندی ۱۴۱
۲ – ۳ – ۴. درج گره‌ای به ابتدای لیست دو پیوندی ۱۴۱
۳ – ۳- ۴. درج گره‌ای در انتهای لیست دو پیوندی ۱۴۲
۴-۳-۴. درج گره‌ای بعد از گره خاص لیست دو پیوندی..... ۱۴۳
۵- ۳ – ۴. حذف گره‌ای از ابتدای لیست دو پیوندی..... ۱۴۴
۶-۳-۴. حذف گره‌ای از انتهای لیست دو پیوندی ۱۴۵
۷ – ۳ – ۴. حذف گره خاص از لیست دو پیوندی ۱۴۵
۸-۳-۴. پیمایش و نمایش گره‌های لیست دو پیوندی از ابتدا به انتها..... ۱۴۶
۹ – ۳ – ۴. پیمایش و نمایش گره‌های لیست دو پیوندی از انتها به ابتدا..... ۱۴۷
۴ – ۴. لیست دو پیوندی چرخشی(حلقوی) ۱۴۷
۵ – ۴. پیاده‌سازی پشته با لیست پیوندی..... ۱۵۲
۱ – ۵ – ۴. تعریف کلاس پشته..... ۱۵۳
۲ – ۵ – ۴. تست خالی بودن پشته..... ۱۵۳
۳-۵-۴. پیاده‌سازی تابع push..... ۱۵۳
۴-۵-۴. پیاده‌سازی تابع pop..... ۱۵۳
۵-۵-۴. نمایش اطلاعات پشته..... ۱۵۳
۶-۴. پیاده‌سازی صف با لیست پیوندی..... ۱۵۶
۱-۶-۴. تعریف کلاس لیست پیوندی با دو اشاره‌گر و تعریف عملیات روی آن..... ۱۵۶
۲ – ۶ – ۴. تعریف کلاس صف و عملیات روی آن ۱۵۶
۷ – ۴. پیچیدگی اعمال مختلف لیست پیوندی در یک نگاه..... ۱۶۲
۸ – ۴. تست‌های ارشد لیست پیوندی..... ۱۶۲
۹ – ۴. پاسخ تشریحی تست‌های ارشد لیست پیوندی..... ۱۶۶
فصل پنجم: درخت ۱۶۹
۱-۵. تعاریف..... ۱۷۲
۲-۵. درخت دودویی..... ۱۷۳
۱-۲-۵. شمارش درخت‌های دودویی ۱۷۳
۲-۲-۵. انواع درخت دودویی ۱۷۳
۳-۲-۵. پیاده سازی درخت‌های دودویی ۱۷۵
۴-۲-۵. پیمایش درخت دودویی ۱۷۸
۵-۲-۵. ساخت درخت دودویی با استفاده از پیمایش‌های آن..... ۱۸۳
۶-۲-۵. نمایش عبارت‌های محاسباتی با درخت دودویی..... ۱۸۵
۷-۲-۵. درخت نخی دودویی..... ۱۸۶
۳-۵. درخت‌های عمومی..... ۱۸۷

۵-۳-۱. نمایش درخت عمومی ۱۸۸
۵-۳-۲. پیمایش درخت عمومی ۱۹۱
۵-۳-۳. تبدیل درخت‌های عمومی به دودویی. ۱۹۱
۵-۴. جنگل ۱۹۲
۵-۴-۱. تبدیل جنگل به درخت دودویی ۱۹۳
۵-۵. درخت‌هایی با ساختار ویژه ۱۹۳
۵-۵-۱. درخت Heap ۱۹۳
۵-۵-۲. درخت جست‌وجوی دودویی ۱۹۹
۵-۵-۳. درخت AVL ۲۰۷
۵-۵-۴. درخت انتخابی ۲۱۵
۵ -۶. تست‌های ارشد درخت ۲۳۰
۵-۷. پاسخ تشریحی تست‌های ارشد درخت .. ۲۴۲

فصل ششم: گراف ۲۵۳

۶-۱. تعاریف ۲۵۳
۶-۲. نمایش گراف ۲۵۸
۶-۲-۱. ماتریس مجاورتی ۲۵۹
۶-۲-۲. ماتریس برخورد ۲۵۹
۶-۲-۳. لیست مجاورتی ۲۵۹
۶-۲-۴.لیست مجاورتی معکوس ۲۶۰
۶-۳. تعداد مسیرها در گراف ۲۶۱
۶-۴. ماتریس مسیر ۲۶۱
۶-۵. پیمایش گراف ۲۶۲
۶-۵-۱. پیمایش عمقی ۲۶۲
۶-۵-۲. پیمایش ردیفی(عرضی) ۲۶۴
۶-۶. یافتن مؤلفه‌های متصل گراف ۲۶۶
۶-۷. مرتب سازی توپولوژیکی ۲۶۷
۶-۸. درخت پوشا ۲۶۸
۶ – ۹. تست‌های کارشناسی ارشد گراف ۲۷۲
۶ – ۱۰. پاسخ تشریحی تست‌های ارشد گراف ۲۶۲

فصل هفتم: مرتب‌سازی ۲۸۸

۷ – ۱. مفاهیم مهم در الگوریتم‌های مرتب‌سازی ۲۸۸
۷ – ۲. الگوریتم‌های مرتب‌سازی ۲۸۹
۷-۲-۱. مرتب‌سازی حبابی ۲۸۹
۷-۲-۲. مرتب‌سازی انتخابی ۲۹۱
۷-۲-۳. مرتب سازی درجی ۲۹۳
۷-۲-۴. مرتب‌سازی درجی Shell ۲۹۴
۷ – ۲-۵. مرتب‌سازی جا به جایی ۲۹۶
۷ – ۲-۶. مرتب‌سازی سریع ۲۹۷
۷ – ۲-۷. مرتب‌سازی ادغامی ۲۹۹
۷-۲-۸. مرتب‌سازی مبنا ۳۰۲
۷-۲-۹. مرتب‌سازی هرمی ۳۰۳
۷ – ۲ – ۱۰. مرتب‌سازی‌های دیگر ۳۰۵
۷-۳.تست‌های ارشد مرتب‌سازی ۳۰۹
۷ – ۴. پاسخ تشریحی تست‌های ارشد مرتب‌سازی ۳۱۹

منابع: ۳۲۶

مقدمه

ساختمان داده‌ها یکی از درس‌های پایه‌ای و مهم رشته مهندسی کامپیوتر، فناوری اطلاعات و علوم کامپیوتر است. کتاب‌های زیادی در زمینه ساختمان داده‌ها تالیف و ترجمه شده است که جای تشکر دارد. کتاب‌های موجود، روی یک بخش خاص متمرکز شده اند. اما، کتاب حاضر علاوه بر تدریس مفاهیم ساختمان داده‌ها با مثال‌های متعدد، الگوریتم‌های بیان شده را با زبان C++ پیاده‌سازی نموده است و در محیط ویژوال استودیو اجرا کرده است. از نکات بارز این کتاب علاوه بر تدریس علمی مفاهیم نکات تستی را نیز بیان نموده است. در همین راستا حدود ٤٥٠ تست کنکور کارشناسی ارشد رشته‌های مهندسی کامپیوتر، فناوری اطلاعات و علوم کامپیوتر دانشگاه‌های دولتی و آزاد به همراه حل تشریحی آن‌ها در کتاب آمده است.

این کتاب شامل ۷ فصل است. در فصل اول، الگوریتم‌ها، پیچیدگی آن‌ها و الگوریتم‌های بازگشتی بیان گردیده است. در فصل دوم کتاب آرایه و کاربردهای آمده است. در فصل سوم، مفاهیم صف و پشته و کاربردآن‌ها شرح داده شده است. در فصل چهارم، لیست پیوندی بیان گردیده است. در فصل پنجم، درخت و کاربردهای آن را می‌بینید. در فصل ششم، گراف و کاربردهای آن بحث شده است و در پایان در فصل هفتم، روش‌های مختلف مرتب‌سازی بیان گردیده است.

در پایان از تمام اساتیدی که در چاپ این کتاب ما را همراهی کردند، به ویژه مهندسین احمد محمدی نژاد، جواد رضا نژاد قادیکلائی، مرتضی بابازاده و اسماعیل میزائی کمال تشکر را داریم.

امیدواریم این اثر نیز مورد توجه اساتید و دانشجویان عزیز واقع شود.

پیاده‌سازی‌های برنامه‌ها را می‌توانید از سایت انتشارات فن‌آوری نوین به آدرس www.fanavarienovin.net دریافت نمایید.

در پایان از تمامی خوانندگان عزیز (اساتید و دانشجویان) تقاضا داریم، هرگونه اشکال، ابهام در متن کتاب، پیشنهادات و انتقادات را به آدرس پست الکترونیک fanavarienovin@gmail.com ارسال نمایند.

بابل، پاییز ۱۳۹۲

مولفین

کتاب‌های منتشر شده انتشارات فن‌آوری نوین	
ردیف	نام کتاب
۱	حل مسائل C (مرجع کامل)
۲	حل مسائل C++ (مرجع کامل)
۳	آموزش گام به گام برنامه نویسی بانک اطلاعات با C# (مرجع کامل)
٤	حل مسائل C# (مرجع کامل)
٥	حل مسائل پاسکال (مرجع کامل)
٦	آموزش گام به گام برنامه نویسی بانک اطلاعات با ویژوال بیسیک‌نت (مرجع کامل)
۷	آموزش گام به گام LINQ با C#
۸	تجارت الکترونیکی
۹	امنیت شبکه
۱۰	اصول طراحی پایگاه‌داده
۱۱	طراحی سیستم‌های شی‌گرا با زبان C#
۱۲	مدیریت استراتیژیک (فن آوری اطلاعات)
۱۳	کاربرد رایانه در مدیریت و حسابداری
۱٤	آموزش گام به گام برنامه نویسی به زبان C++
۱٥	سنجش از دور کاربردی جلد اول
۱٦	گرافیک رایانه‌ای با زبانبرنامه نویسی C#
۱۷	آزمایشگاه پایگاه داده با SQL Server 2012
۱۸	فیزیک الکتریسته
۱۹	ساختمان داده‌ها

فصل ۱ ساختار داده‌ها، الگوریتم‌ها و پیچیدگی

عوامل زیادی در سرعت اجرای برنامه موثرند. دو عامل بسیار مهم کارائی برنامه عبارت‌انـد از: ۱. **سـاختمان داده در نظر گرفته شده برای برنامه** ۲. **الگوریتم.** برنامه‌ای که حافظه کمتری مصرف کند و از الگوریتم کاراتری (سریع‌تر) استفاده نماید، بهتر است.

۱–۱. ساختمان داده‌ها

هر برنامه از دو بخش بسیار مهم تشکیل می‌شود که عبارت‌اند از: ۱. ساختمان داده‌ها ۲. الگوریتم‌ها.

ساختمان داده‌ها، برای دریافت داده‌ها توسط رایانه جهت پیاده‌سازی و اجرای الگـوریتم‌هـا مـورد اسـتفاده قرار می‌گیرند. اما، **الگوریتم‌ها،** دستورالعمل‌هایی هستند که بر روی داده‌ها اعمال می‌شوند.

برنامه‌ای خوب است که ساختمان داده‌ای مناسب داشته و از الگوریتم‌های بهینه‌تری استفاده کند. به عنـوان مثال، فرض کنید، بخواهیـد ۲۰ عـدد صـحیح را خوانـده، مرتـب کنیـد و نمـایش دهیـد. در ایـن مثـال، بهتـرین ساختمان داده آرایه است. چون، تعداد اعداد ثابت(مشخص شده) است. اما، الگوریتم‌های مختلف مرتب‌سازی از قبیل Bubble، Select، Insert، Quick، Shell و غیره وجود دارند که در فصل ۷ به آن‌ها مـی‌پـردازیم. کـه در ادامه می‌آموزیم کدام یک از الگوریتم‌های مرتب‌سازی را انتخاب کنیم.

در این بخش به انواع ساختمان داده‌ها می‌پردازیم. انواع مختلف ساختمان داده‌ها وجود دارند که عبارت‌اند از (شکل ۱–۱).

۱–۱–۱. ساختمان داده‌های ایستا

این نوع ساختمان داده‌ها، تعداد عناصرشان ثابت (ایستا) است. یعنی، در هنگام کامپایل (ترجمه) باید تعـداد عناصر آن مشخص باشد. ساختمان داده‌های ایستا نیز دو نوع‌اند که عبارت‌اند از:

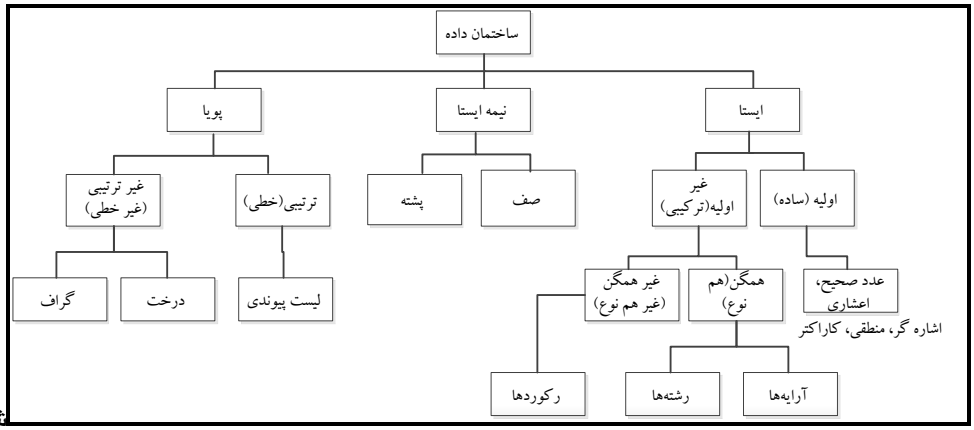
☒**ساختمان داده‌های اولیه**، ساختمان داده‌هایی از قبیل داده‌های عـددی صـحیح (int، long، unsigned int و ...)، عددی اعشاری (float و double)، کاراکتری (char)، منطقی (bool) و اشاره‌گر، ساختمان داده اولیـه هستند. برای تعریف متغیری از این نوع ساختمان داده به صورت زیر عمل می‌شود:

نوع داده لیست متغیرها ;

به عنوان مثال، دستورات زیر را ببینید:

```
int   x, y;
bool  q;
float *p;
```

دستور اول، متغیرهای x و y را از نوع صحیح تعریف می‌کند، دستور دوم، متغیر q را از نوع منطقی تعریف می‌نماید و دستور سوم، متغیر p را از نوع اشاره‌گر تعریف می‌کند که می‌توان آدرس متغیرهای اعشاری را در آن قرار داد.



شکل ۱-۱ انواع ساختمان داده‌ها.

☒**ساختمان داده‌های ترکیبی (غیر اولیه)**، از ترکیبی از داده‌های ساده تشکیل می‌شوند. برخی از این ساختمان داده‌ها در زیر آمده‌اند:

۱. **آرایه:** تعدادی از خانه‌های پشت سر هم حافظه که دارای یک نام و یک نوع باشند. تعریف آرایه در زبان C++ به صورت زیر است:

نوع آرایه نام آرایه [تعداد عناصر];

به عنوان مثال، دستور زیر آرایه‌ای به نام a با ۱۰ عنصر از نوع double (اعشاری با دقت مضاعف) معرفی می‌کند:

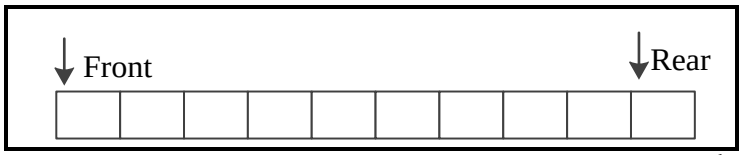
double a[10];

رکورد (ساختمان)، برای نگهداری داده‌های با انواع مختلف که دارای یک نام باشند به کار می‌رود. در زبان C++، ساختمان به صورت زیر تعریف می‌شود:

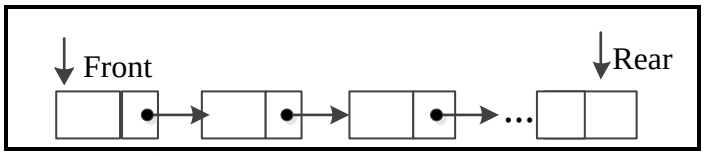
☒ **صف**[1]، ساختمان داده‌ای که در آن اولین ورودی اولین خروجی[2] باشد. یعنی، عناصر در سر[3] صف اضافه می‌شوند و از انتهای[4] آن حذف می‌شوند. اگر در هنگام ترجمه تعداد عناصر صف مشخص باشد، از طریق آرایه می‌توان آن را پیاده‌سازی نمود. این پیاده‌سازی را در شکل زیر می‌بینید:

---

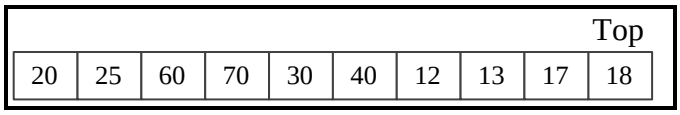
[1].Queue [2]. First Input First Output (FIFO) [3]. Front [4].Rear
[5].Stack [6]. First Input Last Output(FILO) [7]. First Output Last Input (FOLI)



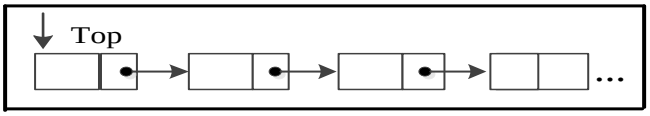
اما، اگر در هنگام ترجمه تعداد عناصر صف مشخص نباشد، از طریق لیست پیوندی می‌توان آن را پیاده-سازی نمود (شکل زیر):



☒ **پشته**[۵]، ساختمان داده‌ای که اولین ورودی آن، آخرین خروجی باشد[۶] یا آخرین ورودی آن، اولین خروجی است[۷]. اعضای پشته از یک طرف آن اضافه یا حذف می‌شوند. پشته دارای اشاره‌گری به نام top است که به بالای پشته اشاره می‌کند (مکانی که عناصر پشته می‌توانند از آن مکان اضافه یا حذف شوند). اگر تعداد عناصر پشته در هنگام ترجمه مشخص باشد، پشته از طریق آرایه پیاده‌سازی می‌شود (شکل زیر):



اما، اگر تعداد عناصر پشته در هنگام ترجمه مشخص نباشد، باید از طریق لیست پیوندی پیاده‌سازی شود (شکل زیر):



ساختمان داده‌های ایستا و نیمه ایستا از نوع ساختمان داده‌های ترتیبی هستند.

مثال ۳۵. برنامه‌ای که اعمال زیر را انجام می‌دهد:

☒ تابع بازگشتی که پارامتر n را دریافت کرده، فاکتوریل n(!n) را برمی‌گرداند (تابع fact).

☒ تابع بازگشتی که پارامتر n را دریافت کرده، مجموع اعداد فرد ۱ تا n را برمی‌گرداند (تابع SumOdd).

☒ تابع بازگشتی که پارامتر n را دریافت کرده، مجموع اعداد زوج 1 تا n را برمی‌گرداند (تابع SumEven).

☒ تابع بازگشتی که پارامتر n را دریافت کرده، مجموع ارقام n را برمی‌گرداند (تابع Sum Digits).

☒ تابع بازگشتی که پارامتر n را دریافت کرده، حاصل ضرب ارقام فرد آن را برمی‌گرداند (تابع MulOddDigits).

☒ تابع بازگشتی که پارامتر n را دریافت کرده، مجموع ارقام بالای ۶ آن را برمی‌گرداند (تابع SumDigitsG6).

☒ تابع بازگشتی که پارامتر n را دریافت کرده، n امین عدد فیبوناچی را برمی‌گرداند (تابع Fibo).

☒ تابع بازگشتی که پارامتر n را دریافت کرده، حاصل ضرب اعداد زوج ۱ تا n را برمی‌گرداند (تابع MulEven).

☒ تابع بازگشتی که پارامتر n را دریافت کرده، مجموع اعداد مضرب ۳ از ۱ تا n را برمی‌گرداند(تابع M3).

☒ تابع بازگشتی پارامتر که n را دریافت کرده، تعداد ارقام صفر آن را می‌شمرد (تابع CountZero).

☒ تابع بازگشتی که پارامتر n را دریافت کرده، مقلوب آن را برمی‌گرداند (تابع Reverse).

☒ تابع بازگشتی که a و b را به عنوان پارامتر دریافت کرده، a^b را برمی‌گرداند (تابع Pow).

☒ تابع بازگشتی که پارامترهای a و b را دریافت کرده، حاصل a تقسیم بر b را برمی‌گرداند (تابع Div).

☒ تابع بازگشتی که پارامترهای a و b را دریافت کرده، حاصل باقی مانده تقسیم صحیح a بر b را برمی‌گرداند (تابع Mod).

☒ تابع بازگشتی که پارامترهای a و b را دریافت کرده، حاصل a ضرب در b را برمی‌گرداند (تابع Mul).

☒ تابع بازگشتی که a و b را به عنوان پارامتر دریافت کرده، حاصل عبارت $\sqrt{a+\sqrt{a+\sqrt{a+\cdots}}}$ (n بار) را برمی‌گرداند (تابع S).

☒ تابع بازگشتی که پارامترهای n و m را دریافت کرده، حاصل $C\binom{n}{m}$ را برمی‌گرداند (تابع C).

این توابع بازگشتی توسط تابع main آزمایش شده‌اند.

```
#include "stdafx.h"
#include <iostream>
#include "math.h"
using namespace std;
unsigned long int Fact(int n) {
if(n == 0) return 1;
else return (n* Fact(n-1));
}
int sumOdd(int n) {
 if(n <= 0) return 0;
 else if (n % 2== 1) return (n+ sumOdd(n-2));
 else return sumOdd(n-1);
}
int sumEven(int n) {
 if(n == 0) return 0;
 else if (n % 2 == 0) return (n+ sumEven(n-2));
 else return sumEven(n-1);
}
int sumDigits(int n) {
  if(n == 0) return 0;
  else  return (n % 10 + sumDigits(n/10));
}
int mulOddDigit(int n) {
 if (n == 0) return 1;
 else if(n % 10 % 2==1) return(n % 10 * mulOddDigit(n/10));
 else return(mulOddDigit(n / 10));
}
int sumDigitG6(int n) {
 if (n == 0) return 0;
 else if (n % 10 > 6) return(n % 10 + sumDigitG6(n / 10));
```

```
  else return(sumDigitG6(n / 10));
}
int Fibo(int n)  {
  if (n == 1 || n == 2) return 1;
  else return(Fibo(n -1) + Fibo(n - 2));
}
long mulEven(int n)  {
   if (n < 2) return 1;
   else if (n % 2 == 0) return(n * mulEven(n - 2));
   else return(mulEven(n - 1));
}
int M3(int n)  {
  if (n < 3) return 0;
  else if (n % 3 == 0) return(n + M3(n - 3));
  else return(M3(n - 1));
}
int countZero(int n)  {
  if (n == 0) return 0;
  else if ( n % 10 == 0) return(1 + countZero(n / 10));
   else return(countZero(n / 10));
 }
int Reverse( int n)  {
  static int r = 0;
  if (n == 0) return r;
  else   {
    r = r * 10 + n % 10;
    return(Reverse(n/10));
  }
}
unsigned long int Pow(int a, int b)  {
  if(b == 0) return 1;
  else return (a * Pow(a, b-1));
}
int Div(int a, int b)  {
  if(a < b) return 0;
  else return (1 + Div(a-b, b));
}
  int Mod(int a, int b)  {
  if(a < b) return a;
  else return (Mod(a-b, b));
}
int Mul(int a, int b)   {
  if(b == 0) return 0;
  else return (a + Mul(a, b -1));
}
double S(double a, int n)  {
  if (n == 1) return(sqrt(a));
  else return(sqrt(a + S(a, n -1)));
}
int C(int n, int m)  {
  if (n == m || m == 0) return (1);
  else return (C(n -1, m -1) + C(n -1, m));
 }
int main()  {
   int n, a, b;
   cout << "Enter n:";
   cin >> n;
   cout << "Fact(" << n << ") = " << Fact(n);
   cout << endl << "Sum Odd number 1 .. " << n << " = " << sumOdd(n);
```

```
 cout << endl << "Sum Even number 1 .. " << n << " = " << sumEven(n);
 cout << endl << "Multiply even number Even 2 .. " << n << " = " << mulEven(n);
 cout << endl << "Sum 3, 6, ..., " << n << " = " << M3(n);
cout << endl << "Sum digits " << n << " = " << sumDigits(n);
cout << endl << "Count zero digits " << n << " = " << countZero(n);
cout << endl << "Multiply odd digits " << n << " = " << mulOddDigit(n);
cout << endl << "Sum digits > 6 " << n << " = " << sumDigitG6(n);
cout << endl << "Fibo(" << n << ") = " << Fibo(n);
cout << endl << "Reverse(" << n << ") = " << Reverse(n);
 cout << "\nEnter a, b:";
 cin >> a >> b;
 cout << a << " ^ " << b << " = " << Pow (a, b);
 cout << endl << a << " * " << b << " = " << Mul (a, b);
 cout << endl  << a << " / " << b << " = " << Div (a, b);
 cout << endl  << a << " % " << b << " = " << Mod (a, b);
 cout << endl  << "Sqrt(Sqrt(... (Sqrt(" << a << " ))...) = "  << S(a, b);
 cout << endl  << "C(" << a << "," << b <<") = "  << C(a, b);
 cin >> n;
 return 0;
}
```

۵–۱. تست‌های ارشد ساختار داده‌ها، الگوریتم‌ها و پیچیدگی

۱. تابع ACK به صورت زیر تعریف می‌شود. مقدار ACK(1,1) برابر است با: (مهندسی کامپیوتر - دولتی ۷۷).

الف: ۵ ب: ۴ ج: ۳ د: ۶

```
int ACK(int m, int n) {
  if (m < 0 || n < 0) return 0;
     else if (m == 0) return (n + 1);
      else if (n == 0)return ACK(m -
1, 1);
else return ACK (m -1, ACK (m, n-
1));
}
```

۲. تابع زیر چه کاری را انجام می‌دهد و L(25) را محاسبه نمایید (مهندسی کامپیوتر- آزاد ۷۶).

$$L(n) = \begin{cases} 0 & if\ n == 1 \\ L([n/2]) + 1 & if\ n > 1 \end{cases}$$

الف: نصف عدد داده شده + ۱ و ۱۳ ب: بزرگ‌ترین عدد صحیح به طوری که $2^L <= n$ و ۴

ج: $L = [log_2^n]$ و ۴ د: گزینه ب و ج صحیح است.

۳. خروجی الگوریتم زیر با فراخوانی test(4) چیست؟ (علوم کامپیوتر- دولتی ۸۴).

```
void test(n) {
  int i;
  if(i > 0) i = n;
  test(--n);
  cout << i << ",";
}
```

الف: خروجی تولید نمی‌کند.

ب: ۴، ۳، ۲، ۱

ج: ۴، ۳، ۲، ۱، ۰

د: ۳، ۲، ۱، ۰

۴. درباره تابع زیر کدام گزینه صحیح است؟(علوم کامپیوتر- دولتی ۸۵).

```
int F(int n, int i) {
  if (i<= n) return (F(n, i + 1) + i);
  else return (0);
}
```

الف: F(10, 1) = 10 ب: F(10, 1) = 11

ج: F(10, 1) = 54 د: F(10, 1) = 55

۵. مقدار F(6) برای تابع زیر برابر است با: (ریاضی- دولتی ۸۱).

```
int F(int m) {
  if (m <= 1) return(m *m -10);
  return (2 * F(m – 2) + 7);
}
```

الف: ۳۱- ب: ۱۳-

ج: ۱۸- د: ۲۶

۶. کدام یک از موارد زیر صحیح است؟ (مهندسی کامپیوتر- آزاد ۸۳).

۱: $n! = O(n^n)$ ۲: $\frac{n^2}{logn} = \theta(n^2)$ ۳: $n^2 logn = \theta(n^2)$ ۴. $n^{2^n} + 6(2^n) = \theta(n^{2^n})$

الف: ۱ و ۳ ب: ۲ و ۳ ج: ۱ و ۴ د: همه موارد

۷. تابع بازگشتی زیر چه عملی را انجام می‌دهد؟ (مهندسی فناوری اطلاعات - آزاد۹۰).

$$T(a,b) = \begin{cases} a & if \quad b = 1 \\ T(a,b-1) + a & \text{در غیر این صورت} \end{cases}$$

الف: محاسبه a * b

ب: محاسبه (a – 1) * b ج: محاسبه a * (b – 1) د: محاسبه (a – 1) * (b -1)

۸. رابطه زیر را در نظر بگیرید. مرتبه اجرایی تابع برابر کدام گزینه می‌باشد؟ (مهندسی فناوری اطلاعات- آزاد۹۰).

$$T(n) = T(n-1) \times T(n-1)$$

الف: $O(2^n)$ ب: $O(n)$ ج: $O(n^{\frac{n}{2}})$ د: $O(log_2^n)$

۹. زمان‌های اجرای چهار الگوریتم مختلف به صورت زیر مشخص شده است، کدام یک دارای پیچیدگی زمانی بیشتری است؟ (مهندسی فناوری اطلاعات – آزاد ۸۶).

الف: $O(n^3)$ ب: $O(n^2)$ ج: $O(2^n)$ د: O(nlogn)

۶۶. در رابطه‌ی بازگشتی $T(n) = 9T\left(\frac{n}{3}\right) + F(n)$ به ازای چند تا از عبارت‌های g(n) زیر، دست کم یک تابع برای f(n)، وجود دارد به طوری که $T(n) = \Theta(g(n))$؟(مهندسی کامپیوتر- دولتی ۹۱).

$n \log n \;\|\; n^2 \;\|\; n^2 \log^2 n \;\|\; n^3$

الف: ۱ ب: ۲ ج: ۳ د: ۴

۶۷. تابع رو به رو را در نظر بگیرید. فرض کنید که T(n) نشان دهنده تعداد عملیات ++(هر سه) در تابع فوق باشد. اگر تابع فوق با پارامتر n فرا خوانده شود،کدام رابطه بازگشتی زیر درست است؟(مهندسی کامپیوتر- دولتی ۹۱).

```
int  test(int  n) {
  int i, j, count=0;
  for(i=0; i < n; i++)
   for(j=0; j < i; j++)
        count++;
  return  count++;
}
```

الف: $T(n) = T(n-1) + 2n + 1$

ب: $T(n) = nT(n-1) + n - 1$

ج: $T(n) = T(n-1) + 2n - 1$

د: $T(n) = nT(n-1) + n + 1$

۶۸. مرتبه زمانی تابع بازگشتی زیر چیست؟(مهندسی فناوری اطلاعات- دولتی ۹۱).

$T(n) = 3T(\lfloor \frac{n}{2} \rfloor)+n^2)$

الف: $\Omega(\log n)$ ب: $\Omega(n^2)$ ج: $\Omega(n \log n)$ د: $\Omega(n^2 \log n)$

۶۹. هزینه زمانی تکه برنامه‌ی زیر کدام است؟(مهندسی فناوری اطلاعات- دولتی ۹۱).

```
i=n; int
while (i>1){
  i /= 2;
  j=i;
  while (j>1)
    j /= 3;
}
```

الف: $O(\log n)$

ب: $O(\log^2 n)$

ج: $O(n)$

د: $O(n^2)$

۷۰. مرتبه زمانی الگوریتم زیر چیست؟(مهندسی کامپیوتر-آزاد ۹۱).

```
i=1; i<=n; i=i+1) for (int
for(int  j=1; j<=n; j=j+1)
```

الف: $\theta(n)$ ب: $\theta(n^2)$

ج: $\theta(n^3)$ د: $\theta(n \log n)$

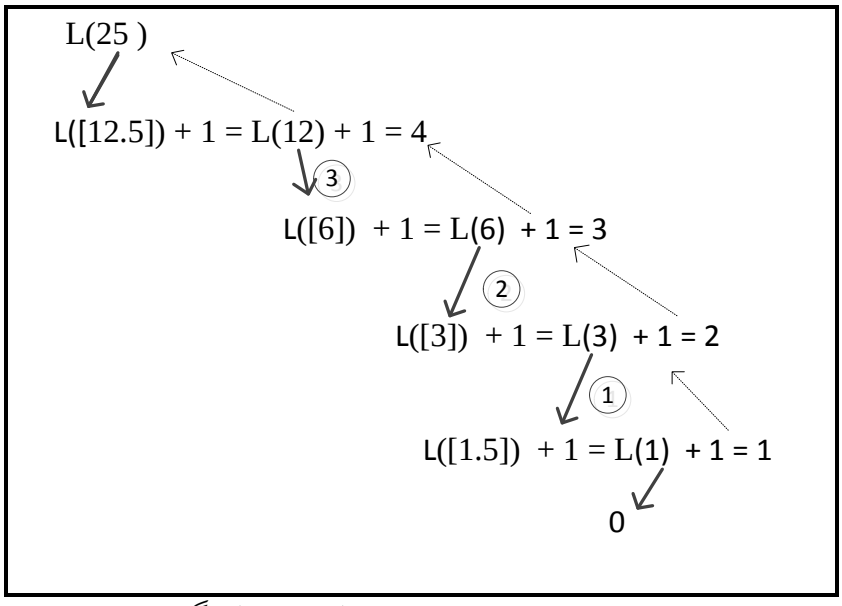
۶-۱. پاسخ تشریحی تست‌های ارشد ساختار داده‌ها، الگوریتم‌ها و پیچیدگی

۱. **گزینه (ج) صحیح است.** چون این عمل در پشته به صورت زیر انجام می‌شود:

ACK (0, 1) = 1 + 1 = 2
ACK (1, 0) = ACK (0, 1) = 2
ACK (1, 1) = ACK (0, ACK (1, 0)) = ACK (0, 2)

پس ACK (1, 1) برابر با ACK (0, 2) است و ACK (0, 2)، برابر با n + 1 یعنی ۳ است.

۲. **گزینه (د) صحیح است.** چون L(25) به صورت زیر محاسبه می‌شود:

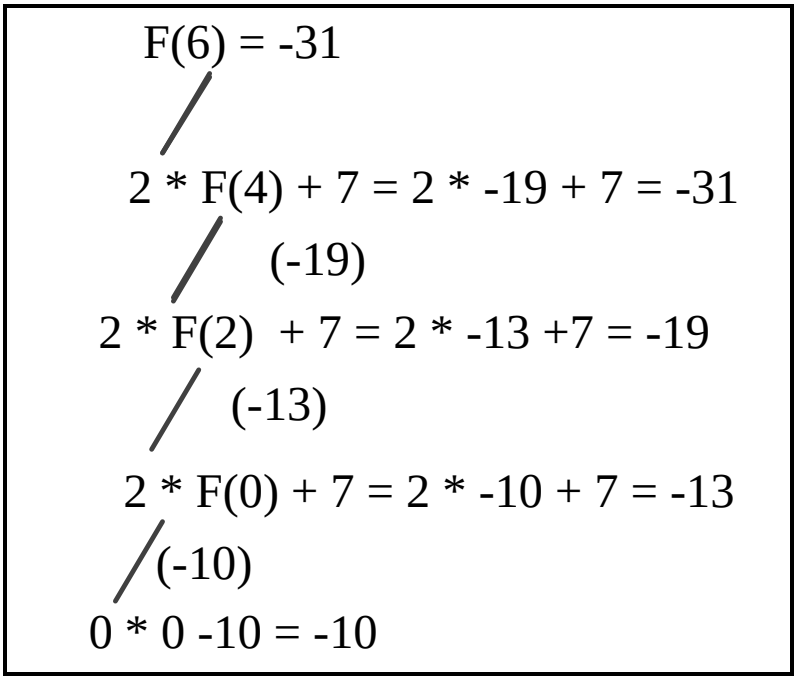


۳. **گزینه (الف) صحیح است.** چون، i در تابع test تعریف شده، ولی مقدار اولیه نگرفته است.

۴. **گزینه (د) صحیح است.** این تابع به صورت بازگشتی جمع اعداد i تا n را برمی‌گرداند. چون F به صورت F(10, 1) فراخوانی گردید. پس جمع اعداد ۱ تا ۱۰ را برمی‌گرداند که برابر است با:

$1 + 2 + \ldots + 10 = \frac{10 \times 11}{2} = 55$

۵. گزینه (الف) صحیح است. چون، درخت بازگشتی F(6) به صورت زیر است:



۶۷. **گزینه (ب) صحیح است.** حلقه بیرونی n بار تکرار می‌شود و دستور ++i آن n-1 بار تکرار خواهد شد. تعداد تکرار حلقه درونی به مقدار شمارنده حلقه بیرونی بستگی دارد و از رابطه بازگشتی $T(n) = n\,T(n-1)$ به دست می‌آید. در نتیجه تعداد کل ++ها برابر با $T(n) = n\,T(n-1) + n + 1$ است.

۶۸. **گزینه (ب) صحیح است.**

$$T(n) = 3\,T\left(\frac{n}{2}\right) + n^2$$
$$= 3\left(3\,T\left(\frac{n}{4}\right) + \left(\frac{n}{2}\right)^2\right) + n^2 = 9T\left(\frac{n}{4}\right) + n^2 + 3\left(\frac{n}{2}\right)^2$$
$$\ldots$$
$$\rightarrow T(n) = 3^i + T\left(\frac{n}{2^i}\right) + n^2 + 3^2\left(\frac{n}{2^2}\right)^2 + \cdots + 3^{i-1}\left(\frac{n}{2^{i-1}}\right)^2$$
$$\rightarrow \quad \left(\frac{n}{2^i}\right) = 1 \rightarrow n = 2^i \rightarrow i = \log_2 n$$

اگر i را در معادله T(n) به دست آمده قرار دهیم، $3^i + T\left(\frac{n}{2^i}\right)$ از درجه n و بقیه جمله از درجه n^2 است. پس، گزینه (ب) صحیح است.

۶۹. **گزینه (ب) صحیح است**(مانند مسائل حل شده کتاب).

۷۰. **گزینه (ب) صحیح است**(مانند مسائل حل شده کتاب).

۲ آرایه

آرایه، ساختمان داده‌ای از نوع خطی می‌باشد که شامل مجموعه عناصر هم نوع و هم‌جوار در حافظه بوده و با یک نام ولی اندیس‌های مختلف معرفی می‌شود. هر عنصر آن با اندیس[۲] و مقدار[۲] مشخص می‌گردد. از آرایه در مواردی **نظیر لیست خطی یا ترتیبی، ماتریس، چند جمله‌ای، صف و پشته، درخت و گراف، پردازش تکاملی و الگوریتم‌های ژنتیک، شبیه‌سازی، پردازش تصویر و ایجاد فونت** می‌توان استفاده نمود. آرایه‌ها می‌توانند به صورت یک بعدی، دوبعدی یا چند بعدی استفاده شوند که هر کدام کاربرد خاصی دارند.

۱– ۲. آرایه یک بعدی(بردار)

از این نوع آرایه برای معرفی لیست‌های خطی، صف، پشته و... استفاده می‌شود که در این فصل و فصل‌های بعدی بررسی می‌شوند.

۱–۱– ۲. مفاهیم کلی و پیاده‌سازی آرایه یک بعدی

قبل از به کار بردن الگوریتم‌های مربوط به آرایه یک بعدی، بهتر است اشاره مختصری به نحوه تعریف و مقداردهی آرایه داشته باشیم.

۱. **تعریف آرایه،** برای استفاده از آرایه همانند متغیر معمولی ابتدا باید آن را معرفی نمود(به صورت زیر):

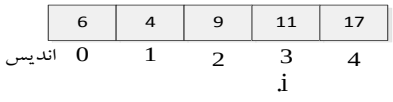
نوع آرایه نام آرایه [تعداد عناصر];

برای مثال، معرفی یک آرایه با ۵ عنصر از نوع صحیح به صورت int x[5]; است:

همان طور که مشاهده می‌شود، نوع آرایه int، نام آرایه x، تعداد عنصر ۵ و اندیس در بازه ۰ تا ۴ است (باید توجه داشت C++ این بازه را کنترل نمی‌کند).

۲. **مقداردهی اولیه به عناصر آرایه،** می‌توان هنگام معرفی آرایه به آن مقدار اولیه داد:

int x[5]={6, 4, 9, 11, 17};



[2]. Index [2] .value

۳. دسترسی به عناصر آرایه، برای دسترسی به عناصر آرایه x دو روش وجود دارد. روش اول دسترسی مستقیم x[*i*] و روش دوم استفاده از اشاره‌گر (x+*i*)*. به عنوان مثال، برای نمایش عنصر شماره ۲ از روش‌های زیر می‌توان استفاده نمود:

	x[0]	x[1]	x[2]	x[3]	x[4]
	یا	یا	یا	یا	یا
	*x	*(x + 1)	*(x + 2)	*(x + 3)	*(x + 4)
x	6	4	9	11	17
اندیس	0	1	2	3	4

روش اول: cout << x[2];

روش دوم: cout << *(x+2);

در هر دو روش خروجی عدد ۹ است.

ذخیره و بازیابی یک عنصر از آرایه در زمان ثابتی صورت می گیرد.

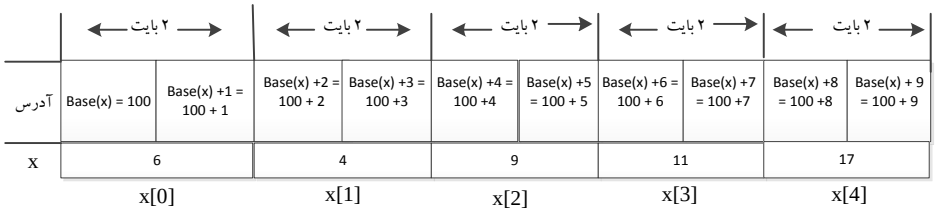
۴. چگونگی ذخیره عناصر آرایه در حافظه، اگر آدرس شروع آرایه x را در حافظه base(x) در نظر بگیریم، آدرس محل ذخیره عنصر*i*ام به صورت زیر محاسبه می‌شود:

آدرس محل ذخیره x[*i*] = base(x) + *i* * اندازه نوع آرایه

مثال ۱-۲. اگر آدرس شروع آرایه x که از نوع int است خانه ۱۰۰ حافظه باشد، آدرس محل ذخیره x[3] چیست؟

i=3، base(x) = 100 و اندازه نوع int برابر با ۲ بایت است. پس طبق رابطه برابر با ۱۰۶ می‌شود:

آدرس محل ذخیره x[3] = 100 + *3 * 2=106*



۲-۱-۲. عملیات مهم بر روی آرایه یک بعدی(لیست خطی)

اعمال زیادی را می‌توان بر روی آرایه یک بعدی انجام داد. متداول‌ترین آن‌ها عبارت‌اند از پیمایش، درج، حذف عنصر، جست‌وجو و ادغام. در این بخش به بررسی بعضی از آن‌ها می‌پردازیم:

پیمایش عناصر، رویت کردن و پردازش تمامی عناصر لیست را پیمایش می گویند. الگوریتم پیمایش آرایه با n عنصر در زیر آمده است:

پیچیدگی زمانی : O(n)

```
for(i=0; i < n ; i++)
  دستورات پردازش x[i]
```

مثال ۲ - ۲. نمونه‌ای از پیمایش عناصر آرایه که محتویات عناصر آرایه را نمایش می‌دهد.

```
for(i=0; i < n; i++)
   cout << x[i];
```

درج عنصری در آرایه: قرار دادن یک عنصر جدید در مکانی از لیست را درج کردن می‌گویند.

می‌خواهیم یک عنصر جدید با مقدار value را در مکان pos از آرایه x درج نماییم. برای انجام این کار باید ابتدا تمامی عناصری که در مکان pos تا n در آرایه قرار دارند، یک خانه به سمت راست انتقال یابند. عنصر در محل مورد نظر درج شده، سپس به n یک واحد اضافه شود (n شماره آخرین عنصر آرایه است).

```
void InsertArray (int x[], int &n, int pos, int value)
{
  for(int i=n; i >= pos ; i--)
        x[i+1] = x[i];
   x[pos] = value;
   ++n;
}
```

در تابع InsertArray، پارامتر n شماره آخرین عنصر آرایه، value مقداری که باید درج شود و pos مکانی که عنصر باید درج شود را مشخص می‌کند.

جدول ۱ – ۲ پیچیدگی زمانی الگوریتم درج در آرایه است.O(n)

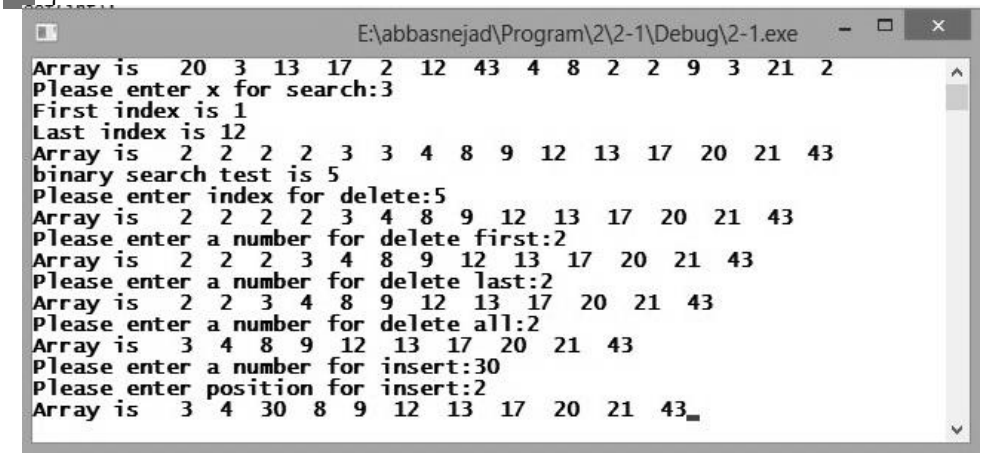
حالت	تعداد جابه‌جایی	علت
بهترین حالت	صفر	زمانی است که عنصر در انتهای لیست درج شود.
بدترین حالت	برابر با تعداد عناصر آرایه	زمانی است که عنصر در ابتدای لیست درج شود.
حالت متوسط	$\frac{n}{2}$	

پیاده‌سازی ۱ – ۲. برنامه‌ای که آرایه‌ای را تعریف کرده، مقدار اولیه به آن تخصیص می‌دهد. سپس اعمالی از قبیل مرتب‌سازی، چاپ عناصر آرایه، جست‌وجو ترتیبی اولین وقوع یک مقدار، جست‌وجو ترتیبی آخرین وقوع یک مقدار، جست‌وجو دودویی یک مقدار، حذف مقداری با اندیس خاص، حذف مقداری از آرایه، حذف تمام تکرار یک مقدار خاص و درج مقداری در مکان خاصی از آرایه را انجام می‌دهد.

```
#include "stdafx.h"
#include<iostream>
using namespace std;
void readArray(int arr[], int n)  {
 cout << "Please enter " << n << " numbers:";
 for(int i = 0; i < n; i ++)
   cin >> arr[i];
}
void printArray(int arr[], int n) {
 cout << "Array is ";
 for(int i = 0; i < n; i ++)
   cout << "  " << arr[i];
}
void bubbleSort(int arr[], int n)  {
 for(int i = 0; i < n - 1 ;  i ++)
   for(int j = i+1 ; j < n; j++)
     if( arr[i] > arr [j])  {
       int temp = arr[i];
       arr[i]= arr[j];
       arr[j] = temp;
     }
}
int binarySearch(int arr[], int n, int key)  {
```

```
   int mid, low = 0, high = n - 1;
   while(low<=high)    {
      mid =(low + high) / 2;
      if(key < arr[mid])
         high = mid - 1;
      else if(key > arr[mid])
         low = mid + 1;
      else
         return mid;
   }
   return -1;
}
int firstIndex(int arr[],int n,int value) {
 for(int i = 0; i < n; i ++)
  if(arr[i] == value) return i;
   return -1;
}
int lastIndex(int arr[],int n, int value) {
 for(int i = n - 1; i >= 0; i --)
  if(arr[i] == value)return i;
   return -1;
}
void removeAt(int arr[],int& n,int index) {
 if (index < n && index >= 0)  {
  for(int i = index; i < n; i++) arr[i] = arr[i+1];
  n --;
 }
}
void removeFirst(int arr[],int& n, int value) {
 for(int i = 0; i < n; i++)
  if(value == arr[i])  {
   for(int j = i; j < n; j ++)  arr[j] = arr[j + 1];
   --n;
   break;
  }
}
void removeAll(int arr[],int& n, int value) {
  for( int i = 0; i < n; i++)
   if(value == arr[i])   {
     for(int j = i; j < n; j++)   arr[j] = arr[j + 1];
     n --;
     --i;
   }
}
void insert (int arr[] , int &n , int pos, int value) {
   for(int i=n; i >= pos ; i--)
     arr[i+1] = arr[i];
   arr[pos] = value;
   ++n;
}
void main() {
 int  a[] = {20,  3, 13, 17, 2, 12, 43, 4, 8, 2, 2, 9, 3,  21,  2};
 int n = sizeof(a) / sizeof(int);
 int x;
 printArray(a, n);
 cout << endl << "Please enter x for search:";
 cin >> x;
 int index= firstIndex(a, n, x);
 if (index != -1 ) cout << "First index is " << index;
```

```
index= lastIndex(a, n, x);
if (index != -1 ) cout << "\nLast index is " << index << endl;
bubbleSort(a, n);
printArray(a, n);
index = binarySearch(a, n, x);
if (index != -1 ) cout << "\nbinary search test is " << index;
cout << "\nPlease enter index for delete:";
cin >> x;
removeAt(a, n, x);
printArray(a, n);
cout << "\nPlease enter a number for delete first:";
cin >> x;
removeFirst(a, n , x);
printArray(a, n);
cout << "\nPlease enter a number for delete last:";
cin >> x;
index = lastIndex(a, n, x);
if (index != -1 ) removeAt(a, n, index);
printArray(a, n);
cout << "\nPlease enter a number for delete all:";
cin >> x;
removeAll(a, n ,x);
printArray(a, n);
cout << endl << "Please enter a number for insert:";
cin >> x;
cout << "Please enter position for insert:";
int pos;
cin >> pos;
insert(a, n, pos,x);
printArray(a, n);
cin >> x; // stop
}
```



این برنامه شامل توابع زیر است:

☒ **تابع readArray()**، آرایه (arr) و تعداد عناصر آن (n) را به عنوان پارامتر دریافت کرده، n عدد خوانده در آرایه قرار می‌دهد.

☒ **تابع printArray()**، آرایه (arr) و تعداد عناصر آن (n) را به عنوان پارامتر دریافت کرده، عناصر آرایه را نمایش می‌دهد.

☒ **تابع bubbleSort()**، آرایه (arr) و تعداد عناصر آن (n) را به عنوان پارامتر دریافت کرده، عناصر آرایه را مرتب می‌کند.

☒ **تابع binarySearch()**،آرایه (arr)، تعداد عناصر آن (n) و مقدار مورد جست‌وجو (key) را به عنوان پارامتر دریافت کرده، مقدار key را در آرایه با روش دودویی جست‌وجو می‌کند. اگر key در آرایه موجود باشد، اندیس مکان آن، وگرنه 1- را برمی‌گرداند.

☒ **تابع firstIndex()**،آرایه (arr)، تعداد عناصر آن (n) و مقدار مورد جست‌وجو(value) را به عنوان پارامتر دریافت کرده، اولین مکان وقوع value را در آرایه برمی‌گرداند. اگر value در آرایه وجود نداشته باشد، 1- را برگشت خواهد داد.

☒ **تابع lastIndex()**،آرایه (arr)، تعداد عناصر آن (n) و مقدار مورد جست‌وجو(value) را به عنوان پارامتر دریافت کرده، آخرین مکان وقوع value را در آرایه برمی‌گرداند. اگر value در آرایه وجود نداشته باشد، 1- را برگشت می‌دهد.

☒ **تابع removeAt()**، آرایه(arr)، مقدار عناصر آن (اشاره‌گر n) و اندیس مکانی که باید عنصر آن مکان حذف شود (index) را به عنوان پارامتر دریافت کرده، عنصر مکان index را از آرایه حذف می‌کند.

☒ **تابع removeFirst()**، آرایه (arr)، تعداد عناصر آن (اشاره‌گر n) و مقداری که باید اولین وقوع آن در آرایه حذف شود (value) را به عنوان پارامتر دریافت کرده، اولین وقوع مقدار value را در آرایه حذف می‌کند.

☒ **تابع removAll()**، آرایه (arr)، تعداد عناصر آن (اشاره‌گر n) و مقداری که تمام تکرار آن باید در آرایه حذف شود (value) را به عنوان پارامتر دریافت کرده، تمام مقدار value را در آرایه حذف می‌کند.

☒ **تابع insert()**، آرایه (arr)، تعداد عناصر آن (اشاره‌گر n)، مکانی که باید مقدار جدید درج شود (pos) و مقدار جدیدی که باید در آرایه درج شود (value) را به عنوان پارامتر دریافت کرده، مقدار value را در مکان pos درج می‌کند.

☒ **تابع main()**، آرایه را تعریف کرده، مقدار اولیه می‌دهد. سپس هر یک از توابع بیان شده را آزمایش می‌نماید.

۳ – ۲. مسائل حل شده

۱. آرایه پویا چیست ؟ در c++ چگونه معرفی می‌شود.

آرایه پویا همانند آرایه ایستا است، با این تفاوت که می‌توان در زمان تعریف، با یک متغیر تعداد عناصر آرایه را تعیین کرد(همانند دستور redim در محیط ویژوال‌بیسیک). آرایه پویا با عبارت زیر معرفی می‌شود:

```
type  *name;
name = new type[ number of elements ];
```

به عنوان نمونه، در مثال زیر، یک آرایه پویا به نام x از نوع اعشاری با n عنصر معرفی شده است (مقدار n نیز از ورودی دریافت می‌شود):

```
int n;
float  *x;
cin >> n;
```

```
x = new float[ n ];
```

آرایه های پویا پس از تعریف، هیچ تفاوتی با آرایه ایستای معمولی ندارند و دسترسی به عناصر آن‌ها همانند آرایه ایستا صورت می‌گیرد. اما باید به این نکته توجه داشت که حافظه استفاده شده برای آرایه پویا را باید بعد از استفاده، با دستور delete آزاد کنید:

```
delete[ ] x;
```

حافظه‌ای که به آرایه‌های پویا x تخصیص داده بود، آزاد می‌کند.

۲. الگوریتمی برای بررسی تساوی دو آرایه بنویسید و پیچیدگی زمانی آن را مشخص نمایید.

شرط تساوی دو آرایه عبارت‌اند از:

- ☒ تعداد عناصر دو آرایه با هم برابر باشد.
- ☒ محتویات عناصر دو آرایه با هم برابر باشد.

با توجه به یک حلقه تکرار for، مرتبه اجرایی آن O(n) می‌باشد.

```
int IsEqual(int a[], const int m, int b[], const int n)    {
   if (n!=m) return  0;
    for(int i=0; i<n ; i++)
       if (a[i]!=b[i]) return  0;
   return 1;
}
```

۴ – ۲. تست‌های ارشد آرایه

۱. اگر آرایه $A[L_1..U_1, L_2..U_2]$ را به صورت ستونی در یک آرایه یک بعدی ذخیره نماییم، فرمول ذخیره سازی چیست؟(با فرض α آدرس شروع آرایه باشد)(مهندسی فناوری اطلاعات – آزاد ۸۵).

الف: $Loc(A[i, j])=\alpha+[(i - L_1)(u_1 - L_1 + 1) + j - L_2] \times e$

ب: $Loc(A[i, j])=\alpha + [(j - L_2)(u_1 - L_1 + 1) + i - L_1] \times e$

ج: $Loc(A[i, j])=\alpha+[(i - L_1)(u_2 - L_2 + 1) + j - L_2] \times e$

د: $Loc(A[i, j])=\alpha+[(i - L_2)(u_2 - L_2 + 1) + i - L_1]\times e$

۲. ماتریس سه قطری، ماتریس مربعی است که در آن به جز عناصر قطر اصلی و قطر بالا و پایین آن، سایر عناصر برابر صفر می‌باشند. عناصر غیر صفر را به صورت سطری در یک آرایه یک بعدی می‌ریزیم. فرمول ذخیره‌سازی چیست؟(به فرض ابعاد ماتریس$n\times n$ باشد و به فرض آدرس شروع آرایه α باشد)(مهندسی کامپیوتر– آزاد ۸۵).

الف: $\alpha+2i+j-3$ ب: $\alpha+2j+i-3$ ج: $\alpha+i+2j+3$ د: $\alpha+2i+j+3$

۳. آرایه دو بعدی $A[1..m, 1..n]$ به صورت سطری در حافظه با آدرس شروع ۱۰۰ ذخیره شده است. در این صورت، آدرس عضو $A[i, j]$ آرایه در حافظه کدام است؟(فرض کنید اندازه هر عنصر آرایه یک بایت باشد)(مهندسی فناوری اطلاعات– آزاد ۸۶).

الف: $i+j\times m-(m-99)$ ب: $n\times i-n+j+99$ ج: $i\times m+ j\times n+100$ د: $j+i\times m-m+101$

۴. با فرض آن که تعداد عناصر موجود درآرایه n باشد و بخواهیم عنصر x را در آرایه جست‌وجو کنیم و عنصر x در آرایه وجود داشته باشد، در این صورت، پیچیدگی زمانی حالت میانی جست‌وجو کدام است؟(مهندسی کامپیوتر– آزاد ۸۶).

الف: $\frac{n}{2}$ ب: $\frac{n-1}{2}$ ج: $\log_2^n$ د: $\frac{n+1}{2}$

۵. ماتریس پنج قطری، ماتریس مربعی است $n\times n$ که به جز عناصر قطر اصلی و دو قطر بالا و دو قطر پایین آن، سایر عناصر برابر صفر است. اگر عناصر غیر صفر را به صورت سطری در یک آرایه یک بعدی ذخیره نماییم، فرمول ذخیره سازی $loc(A[i, j])$ چیست؟(به فرض α آدرس شروع آرایه و e اندازه هر عنصر آرایه باشد)(مهندسی کامپیوتر– آزاد ۸۶).

الف: $i\leq 2$، $\alpha+[3i+j-4]\times e$ و $i>2$، $\alpha+[4i+j-6]\times e$ ب: $i\leq 2$، $\alpha+[4i+j-6]\times e$ و $i>2$، $\alpha+[3i+j-4]\times e$

ج: $i\leq 2$، $\alpha+[4i+j-3]\times e$ و $i>2$، $\alpha+[3i+j-6]\times e$ د: $i\leq 2$، $\alpha+[3i+j-6]\times e$ و $i>2$، $\alpha+[4i+j-3]\times e$

۶. بهترین زمان برای ترانهاده گرفتن از یک ماتریس خلوت (Sparse) چقدر است؟ اگر n، تعداد سطرها، m، تعداد ستون‌ها و t، تعداد عناصر ماتریس باشد(مهندسی کامپیوتر– آزاد ۸۲).

الف: $O(nt)$ ب: $O(mt)$ ج: $O(mnt)$ د: $O(nm)$

۷. اگر $A_{13\times 5}$، $B_{5\times 89}$، $C_{89\times 3}$ و $D_{3\times 34}$ باشد، حداقل تعداد ضرب لازم برای محاسبه ABCD چقدر است؟ (مهندسی کامپیوتر– آزاد ۸۱).

الف: 2586 ب: 4055 ج: 2856 د: 5420

۵ – ۲. پاسخ تشریحی تست‌های ارشد آرایه

۱. گزینه (ب)صحیح است.

۲. گزینه (الف) صحیح است. اگر اندیس اولین خانه آرایه یک (۱) باشد فرمول $2i+j-2$ صحیح می‌باشد. امّا اگر اندیس اولین خانه آرایه صفر(۰) باشد، فرمول$2i+j-3$ صحیح خواهد بود.

۳. گزینه (ب)صحیح است. زیرا آدرس A[i, j] برابر است با:

$[(i-1)\times n+(j-1)\times 1+\alpha=n\times i-n+j-1+100=n\times i-n+j+99$

٤. گزینه (د) صحیح است. چون عناصر آرایه ممکن است، مرتب نباشند، در بهترین حالت، x در اولین خانه آرایه قرار دارد. در بدترین حالت x در آخرین خانه (nامین خانه) قرار دارد. پس به طور متوسط $\frac{n+1}{2}$ مقایسه نیاز است.

۵. گزینه (الف) صحیح است.

٦. گزینه (ب) صحیح است.

۷. گزینه (ج) صحیح است. چون $A_{13\times 5}$، $B_{5\times 9}$ و $C_{89\times 3}$ و $D_{3\times 34}$ به صورت زیر محاسبه می‌شود:

$A_{13\times 5}\ B_{5\times 89}\ C_{89\times 3}\ D_{3\times 34}$

$A_{13\times 5}\ (B_{5\times 89}\ C_{89\times 3})\ D_{3\times 34}= A_{13\times 5}\ (B\ C)_{5\times 3}\ D_{3\times 34}=(A\ B\ C)_{13\times 3}\ D_{3\times 34}=(A\ B\ C\ D)_{13\times 34}$

فصل ۳ پشته وصف

پشته و صف ساختمان داده‌ای از نوع خطی نیمه ایستا می‌باشند که می‌توان آن‌ها را با استفاده از آرایه و لیست پیوندی پیاده‌سازی نمود. پشته و صف کاربردهای زیادی در اکثر برنامه‌های کامپیوتری دارند.

۱– ۳. پشته

پشته ساختمان داده‌ای ترتیبی شامل لیستی از داده‌ها است که در آن عملیات درج و حذف عناصر فقط از یک طرف آن که بالای پشته نام دارد، صورت می‌گیرد. در واقع، عناصر یک پشته، به ترتیب عکس ورودی حذف می‌شوند که به این حالت FILO یا LIFO می‌گویند(یعنی، آخرین ورودی به پشته، اولین خروجی از پشته است).

به عنوان مثال، اگر در یک پشته به ترتیب عناصر A، B، C، D، E و F اضافه شوند، آنگاه اولین عنصری که از پشته برداشته می‌شود، F خواهد بود. یعنی، F که آخرین عنصر ورودی به پشته می‌باشد، اولین عنصری است که از پشته خارج می‌شود.

۱–۱– ۳. پیاده‌سازی پشته با آرایه

پشته را می‌توان با استفاده از آرایه و لیست پیوندی پیاده‌سازی نمود. در این فصل نحوه پیاده‌سازی پشته با آرایه و الگوریتم‌های مربوط به آن بررسی می‌شود و در فصل بعد پشته را با لیست پیوندی پیاده‌سازی نموده و مزایا و معایب هر یک از نظر پیچیدگی زمانی و حافظه بیان می‌شود.

برای پیاده‌سازی پشته با آرایه ساختار داده‌ای به صورت زیر در نظر می‌گیریم:

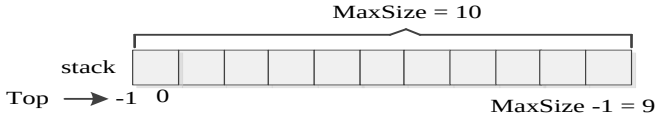
- ☒ **ثابت maxSize**، حداکثر طول پشته را مشخص می‌کند.
- ☒ **متغیر top**، اندیس عنصر بالای پشته را مشخص می‌کند. مقدار اولیه آن ۱- است که نشان دهنده خالی بودن پشته است.
- ☒ **معرفی آرایه‌ای به نام Stack**، آرایه‌ای به نام Stack به طول maxSize ایجاد کرده که نوع آن می‌تواند int ، char و... باشد.

مثال ۱ – ۳. پشته‌ای با ده عنصر از نوع int پیاده‌سازی نمایید.

برای پیاده‌سازی یک پشته با ده عنصر دستورات زیر به کار می‌روند:

```
const int maxSize=10;
int top = -1;
int Stack [maxSize];
```

شکل زیر را ببینید:



۲–۱– ۳. عملیاتی که بر روی پشته انجام می‌شود

عملیات زیر را می‌توان بر روی پشته انجام داد:

۱. **بررسی پر بودن پشته(IsStackFull):** با توجه به این که شرط پر بودن پشته top == maxSize-1 است،کافی است تابعی نوشته شود تا اگر پشته پر باشد(top == maxSize-1)، مقدار ۱، در غیر این صورت، مقدار صفر را برگرداند:

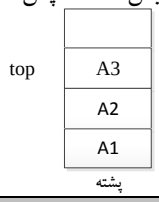
```
int IsStackFull() {
   if (top == maxSize-1)   return 1;
   else    return 0;
}
```

۲. **بررسی خالی بودن پشته**، با توجه به این که شرط خالی بودن پشته top == -1 است،کافی است تابعی نوشته شود تا اگر پشته خالی باشد(top == -1)، مقدار 1، در غیر این صورت، مقدار صفر را برگرداند:

```
int IsStackEmpty() {
   if (top == -1)  return 1;
   else    return 0;
}
```

۳. **عمل درج مقدار جدید در بالای پشته(Push)**، برای انجام این عمل، اگر پشته پر نباشد، ابتدا متغیر top را یک واحد افزایش داده، سپس مقدار جدید را در موقعیتی که top اشاره می‌کند، قرار می‌دهیم:

```
void Push(int value) {
    if (IsStackFull( ) )   cout<< " Stack is Full "
    else Stack[++top]=value;
}
```



و یک عمل جایگزینی است. ifچون شامل یک دستور است. O(1)پیچیدگی زمانی الگوریتم درج در پشته

۴. **عمل حذف از بالای پشته(Pop)**، برای انجام این عمل، اگر پشته خالی نباشد، ابتدا محتویات بالای پشته را در یک متغیر قرار داده، سپس از top یک واحد کم می‌شود:

```
int Pop() {
   if (IsStackEmpty( ) ) {
       cout<<" Stack is Empty
       return -1;
    }
    else {
      value =Stack[top --];
      return value;
    }
}
```

است،O(1)برابر با Popپیچیدگی زمانی تابع و یک عمل جایگزینی است. ifچون شامل یک دستور

۳–۱–۳.ساختار تعریف کلاس Stack

ساختار پشته را با کلاس نیز می‌توان تعریف کرد. تعریف کلاس Stack به صورت زیر می‌باشد:

```
template < class KeyType > class Stack{
  public:
   Stack(int  maxSize=DefaultSize);
   ~Stack();
   boo l  isFullStack ();
   bool   isEmptyStack ();
```

```
    void  Push(constKeyType& item);
    KeyType*  Pop(KeyType&);
};
```

کلاس Stack از بخش‌های زیر تشکیل می‌شود:

☒ **داده‌های عضو آن**، که شامل Stack، maxSize و top است:

```
KeyType *stack;//آرایه از نوع KeyType
int maxSize;// حداکثر عناصر پشته
int top;//اشاره‌گر به بالای پشته که اندیس فعلی آن را بر می‌گرداند
```

☒ **تابع سازنده پشته**، تابعی هم نام با نام کلاس است که فضای مورد نیاز کلاس را اخذ کرده، به اعضای آن مقدار اولیه می‌دهد:

```
template <class KeyType>
Stack<KeyType>::Stack(int maxStackSize):maxSize(maxStackSize) {
    stack = new KeyType[maxSize];
    top = -1;
}
```

☒ **تابع مخرب پشته**، حافظه‌ای که برای پشته تخصیص یافته را پس گرفته و top را برابر با ۱- در نظر می‌گیرد تا نشان دهد پشته خالی است:

```
template <class KeyType>
Stack <KeyType>::~ Stack() {
    delete [] stack;
    top = -1;
}
```

☒ **تست پر بودن پشته**، تابعی به نام IsFull است که به صورت زیر تعریف می‌شود:

```
template <class KeyType>
bool  Stack<KeyType>::IsFull() {
    return (top == (maxSize-1));
}
```

☒ **تست خالی بودن پشته**، تابعی به نام IsEmpty است که به صورت زیر تعریف می‌شود:

```
Template <class KeyType>
bool Stack<KeyType>::IsEmpty(){
    return (top == -1);
}
```

☒ **عمل Push**، چنانچه پشته پر نباشد، عنصری در بالای پشته قرار می‌دهد:

```
template <class KeyType>
void Stack<KeyType>::Push(constKeyType& x) {
    if (IsFull()) return;
    else {
    top = top + 1;
    stack[top] = x;
    }
}
```

☒ **عمل Pop**، چنانچه پشته خالی نباشد، عنصری بالای پشته را بازیابی می‌کند:

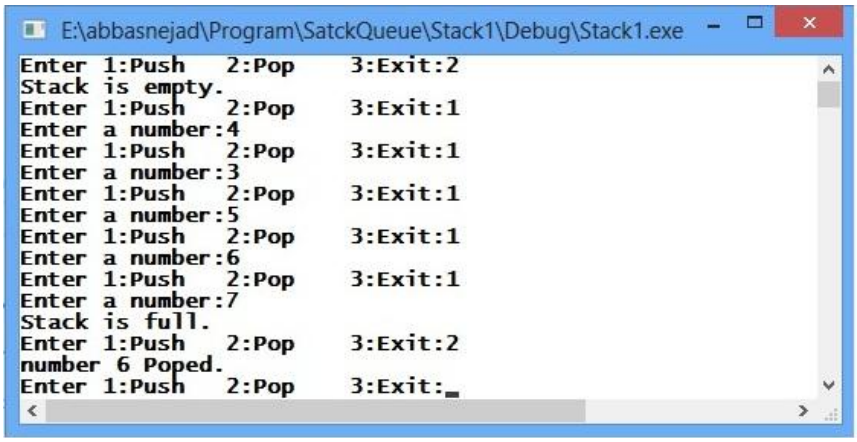
```
template <class KeyType>
KeyType* Stack<KeyType>::Pop(KeyType& x) {
    if (IsEmpty())  return null;
```

```
      else {
      x = stack[top];
      top = top – 1;
      return  &x;
      }
}
```

پیاده‌سازی ۱ – ۳. برنامه‌ای که کلاسی به نام Stack تعریف کرده، اعمال مختلف بر روی آن از قبیل Push و Pop را انجام می‌دهد.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
class  Stack   {
   private:
     int maxSize;
     double *stack;
     int top;
   public:
     Stack(int s) : maxSize(s), top(-1) {
     stack =new double(maxSize);  }
     ~Stack()    {   delete stack;  }
    void  Push(double j) {
       if( isFull() == true)  cout<< "Stack is full.\n";
 else  stack[++top] = j;
    }
   double Pop() {
      if( isEmpty() == true) {
cout<< "Stack is empty.\n";
            return -1;
      }
     else  return stack[top--];
  }
  bool  IsEmpty() {  return (top == -1);    }
  bool  IsFull()  {  return (top == maxSize-1);   }
};
int main()  {
   Stack s1(4);
   int item;
  double num;
  while(true)
  {
    cout<< "Enter 1:Push   2:Pop    3:Exit:";
    cin>> item;
    if( item== 3) break;
    if (item == 1)    {
        cout<< "Enter a number:";
        cin>>num;
        s1.Push(num);
    }
   else if (item == 2 ) {
       double pop=s1.Pop();
      if (pop != -1) cout<< "number " << pop << " Poped.\n";
   }
   }
```

```
  s1.~Stack();
  return 0;
}
```

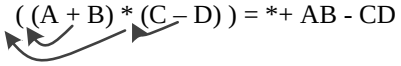


این برنامه شامل بخش‌های زیر می‌باشد:

کلاس Stack دارای اعضای داده‌ای maxSize (حداکثر تعداد عناصر پشته)، اشاره‌گر Stack (مکانی که عناصر پشته قرار می‌گیرند) و top (اشاره‌گر بالای پشته) می‌باشد. علاوه بر این، کلاس دارای تابع سازنده Stack (تابع سازنده Stack که s را به عنوان پارامتر دریافت کرده، maxSize را برابر s، top را برابر ۱- و به اندازه maxSize فضا از حافظه اخذ می‌کند)، تابع مخرب Stack~ (فضای اخذ شده توسط Stack را به سیستم عامل برمی‌گرداند)، تابع Push() (مقدار j را به عنوان پارامتر دریافت کرده، اگر پشته پر نباشد، مقدار j را در بالای پشته قرار می‌دهد)، تابع Pop() (اگر پشته خالی باشد، یک پیغام نمایش داده، ۱- را برمی‌گرداند. وگرنه عنصر بالای پشته را بازیابی می‌کند)، تابع IsEmpty() (اگر پشته خالی باشد، true را برمی‌گرداند. وگرنه false را برگشت خواهد داد) و تابع IsFull() (اگر پشته پر باشد، true وگرنه، false را برمی‌گرداند) است.

۱. معادل پیشوندی و پسوندی عبارت میانوندی مقابل چیست؟ ((A + B) * (C – D))

برای تبدیل عبارت میانوندی به معادل پیشوندی، هر عملگر را به قبل از پرانتز خودش منتقل می‌کنیم. پس، داریم:



ولی برای تبدیل عبارت به معادل پسوندی، هر عملگر را به بعد از پرانتز خودش منتقل می‌نماییم. پس، داریم:

۲. عبارت پسوندی و پیشوندی عبارت میانوندی مقابل چیست؟ A / B – C + D * E – A * C / D

برای بدست آوردن عبارت پسوندی، ابتدا عبارت را پرانتزگذاری می‌کنیم. پس، داریم:

((((A / B) – C) + (D * E)) – ((A * C) / D))

سپس، عملگرها را به بعد از پرانتز خودش انتقال می‌دهیم:

((((A / B) – C) + (D* E)) – ((A * C) / D)) = AB / C – DE * + AC * D / -

اکنون برای بدست آوردن عبارت پیشوندی، عملگر را به قبل از پرانتز خودش انتقال می‌دهیم:

((((A / B) – C) + (D* E)) – ((A * C) / D)) = - + - / A B C * D E / * A C D

۳. عبارت پسوندی و پیشوندی عبارت میانوندی مقابل چیست؟ ((A – B) * (C – (D + E)))

برای تبدیل عبارت میانوندی به عبارت پسوندی عملگرها را بعد از پرانتز خودش انتقال می‌دهیم.

((A - B) * (C – (D + E))) = AB – DE + C - *

اما، برای تبدیل این عبارت به عبارت پیشوندی عملگرها را به قبل از پرانتز خودش منتقل می‌نماییم:

((A - B) * (C – (D - E))) = * - ABC – C - DE

==========

۳۰. دو پشته را در یک آرایه ارائه کرده‌ایم که در خلاف جهت یک دیگر رشد می‌کنند. مرتبه زمانی روال حذف، اضافه و شرط پر بودن هر دو پشته چیست؟(مهندسی فناوری اطلاعات – آزاد۸۵)

الف: O(1) و $TOP_1+1=TOP_2$ ب: O(1) و $TOP_1=TOP_2$

ج: O(m) و $TOP_1+1=TOP_2$ د: O(m) و $TOP_1=TOP_2$

۳۱. اگر رشته اعداد 4، 3، 5، 9، 7، 2 را به ترتیب از سمت چپ وارد یک Stack نماییم، کدام یک از خروجی-های زیر از این Stack امکان‌پذیر خواهد بود؟(خروجی‌ها را از سمت چپ به راست بخوانید) (مهندسی فناوری اطلاعات – آزاد۸۶)

الف: 4، 3، 5، 7، 2، 9 ب: 2، 3، 4، 9، 7، 5 ج: 3، 4، 2، 9، 5، 7 د: 4، 3، 9، 7، 5، 2

۳۲. عبارت Infix زیر معادل با کدام یک از عبارات Postfix است؟(مهندسی کامپیوتر- آزاد ۷۶)

((A + B) * D)↑(E- F)

الف: AB+D*EF -↑ ب: ABD+*EF -↑ ج: AB+D*EF-↑+ د: هیچ کدام

۳۳. به چند حالت می‌توان عبارت زیر را به طور کامل پرانتز گذاری کرد تا عبارت حاصل (با توجه به اولویت عملگرها) به ازای همه مقادیر برابر شوند؟(علوم کامپیوتر – دولتی ۸۲)

a – b * c – d * e / f

توضیح : در پرانتزگذاری کامل هر عبارت به صورت (E1 Op E2) است که E1 و E2 هم به همین صورت پرانتزگذاری شده و Op یک عملگر است.

الف: ۱ ب: ۲ ج: 5! د: $\frac{1}{5}\binom{10}{5}$

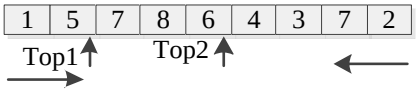
۳۴. برای ساخت یک صف Q از دو پشته S_1, S_2 استفاده می‌کنیم. برای درج x در انتهای Q، عمل Push(S_1,x) را انجام می‌دهیم. برای حذف یک عنصر از ابتدای Q، اگر S_2 خالی نباشد، عمل Pop(S_2) را انجام می‌دهیم. در غیر این صورت، همه‌ی عناصر S_1 را به ترتیب Pop کرده و در S_2، Push می‌کنیم. اکنون عمل Pop بر روی S_2 عنصر ابتدایی Qرا بر می‌گرداند. اگر بر روی Q که در ابتدا خالی است، ۱۰۰عمل صورت می‌گیرد (درج در انتها، حذف از

ابتدا یا هر ترتیب دلخواهی از آن‌ها). حداکثر هزینه چه مقدار خواهد بود؟ فرض کنید هر Push و هر Pop بر روی هر یک از این دو پشته ۱ واحد هزینه دارد(مهندسی کامپیوتر- دولتی ۹۲).

الف:۱۵۰ ب:۱۵۱ ج:۱۹۹ د:۲۰۰

۵- ۳ . پاسخ تشریحی تست‌های ارشد پشته وصف

۳۰. گزینه (الف) صحیح است. از آن جایی که عملیات درج و حذف از پشته(Push و Pop) از مرتبه O(1) هستند. چون نیاز به حلقه تکرار ندارند. پس گزینه‌های(ج) و(د) نادرست هستند. چون پشته‌ها در خلاف جهت هم حرکت می-کنند، پس به صورت شکل زیر پر می‌شوند:



همان طور که مشاهده می‌شود، Top2 برابر با Top1+1 است.

۳۱. گزینه (ج) صحیح است. زیرا ابتدا اعداد ۲ و ۷ را در پشته قرار می‌دهیم. پشته به شکل

رو به رو تغییر می‌یابد.

اکنون ۷ را از پشته Pop می‌نماییم(خروجی ۷). در ادامه اعداد ۹ و ۵ را در پشته می‌نویسیم. پشته به شکل مقابل تغییر می‌یابد.

اکنون همه مقادیر را از پشته Pop می‌نماییم تا خروجی به اعداد ۲، ۹، ۵،۷ تغییر یابد. در ادامه ۳ و۴ را در پشته می نویسیم تا پشته به شکل مقابل تغییر یابد.

در پایان، ۳ و ۴ را از پشته Pop می‌نماییم تا اعداد۳ و۴ به انتهای خروجی ۲، ۹، ۵، ۷ اضافه شوند. پس خروجی به صورت ۳، ۴،۲، ۹، ۵ و ۷ خواهد بود.

۳۲. گزینه (الف) صحیح است. ابتدا کل عبارت را پرانتز گذاری کامل کرده و سپس عملگر مربوط به هر عبارت را به انتهای همان عبارت انتقال می‌دهیم تا عبارت پسوندی آن حاصل شود:

(((A + B) * D)↑(E- F)) =>(((A B +) D*) (E F-)↑) => A B + D* E F-↑

۳۳. گزینه (الف) صحیح است. چون فقط عبارت پرانتز گذاری ((a – (b * c) –(d * e) / f)) صحیح است. بقیه پرانتز گذاری‌ها عبارت دیگری را ایجاد خواهند کرد.

۳٤. گزینه (ج) صحیح است. چون برای درج (Push)، هزینه برابر ۹۹ خواهد بود و برای حذف (Pop) (هزینه ۱۰۰ (۹۹+۱) است. پس، در مجموع هزینه برابر ۱۹۹ می‌باشد.

فصل ۴ لیست پیوندی

در ساختمان داده‌ها اشیایی نظیر آرایه، صف و پشته، عناصر به صورت پشت سر هم (متوالی) ذخیره می‌شوند. بنابراین ۱: اگر عنصر a_i آرایه در محل L_i قرار داشته باشد، عنصر $a_i + 1$، در محل $L_i + C$ قرار دارد (C مقدار ثابتی است که طول هر عنصر آرایه می‌باشد). ۲: اگر i اولین گره در صف حلقوی، در مکان L_i باشد، $(i+1)$امین گره در مکان $(L_i + C)\%n$ قرار می‌گیرد. ۳: و اگر بالاترین عنصر پشته در مکان L_i باشد، پایین‌ترین عنصر پشته در مکان $L_i - C$ خواهد بود. این نوع ساختمان داده‌ها دارای دو مشکل زیر می‌باشند:

۱. تعداد عناصر باید در زمان کامپایل مشخص باشد. این محدودیت نه تنها ممکن است موجب به هدر رفتن حافظه زیادی گردد. بلکه، امکان دارد گاهی اوقات نتوان بعضی از عناصر را در آرایه ذخیره کرد. به عنوان مثال، فرض کنید آرایه‌ای با ۱۰۰ عنصر در نظر گرفته‌اید. اکنون بخواهید فقط ۱۰ عنصر را در آرایه ذخیره نمایید، ۹۰ عنصر آرایه بی‌استفاده خواهند شد (حافظه در نظر گرفته شده برای این ۹۰ عنصر هدر می‌رود) یا بخواهید ۱۱۰ عنصر را در این آرایه ذخیره کنید، ۱۰ عنصر آخر را نمی‌توانید در آرایه ذخیره کنید. چون، آرایه ۱۰۰ عنصری در نظر گرفته شده است. پس نمی‌توان ۱۱۰ عنصر را در آن ذخیره نمود.

۲. اگر عناصر آرایه مرتب شده باشند (مانند شکل ۱ - ۴) و بخواهید عنصری مثل ۴۵ را به آرایه اضافه کنید، عناصر ۸۵، ۷۵، ۵۵ باید به سمت راست شیفت یابند یا اگر بخواهید عنصری مانند ۳۵ را حذف کنید، عناصر ۵۸، ۷۵ و ۸۵ باید به سمت چپ انتقال یابند. عمل شیفت موجب کندی الگوریتم‌های درج و حذف عناصر در آرایه مرتب خواهد شد.

10	20	35	55	75	85

شکل ۱ - ۴ ذخیره اعداد مرتب در آرایه.

راه حل مناسب برای رفع مشکل شیفت داده‌ها در نمایش ترتیبی استفاده از نمایش پیوندی است. بر خلاف ساختمان داده‌های ایستا که داده‌ها به صورت پشت سر هم در حافظه قرار می‌گیرند، در لیست پیوندی[3] عناصر می‌توانند در هر کجای حافظه قرار گیرند. اکنون باید بتوان عناصر لیست پیوندی را به هم پیوند زد. برای این منظور، هر عنصر لیست پیوندی که گره[2] نام دارد، از دو بخش تشکیل شده است: این دو بخش عبارت‌اند از:

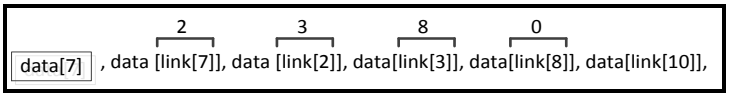
۱. بخش داده، اطلاعاتی که لیست پیوندی باید نگهداری کند را معرفی می‌کند. این بخش می‌تواند از یک یا چند قلم داده (فیلد) تشکیل شود. تعداد اقلام این بخش به نوع سیستم‌تان بستگی دارد.

---

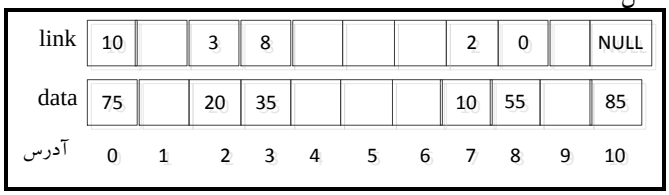
[3].Link List [2].Node

۲. بخش پیوندها، اشاره‌گرهایی هستند که برای ایجاد ارتباط منطقی بین گره‌ها به کار می‌روند. این بخش نیز می‌تواند شامل یک یا چند فیلد باشد. لیست پیوندی که دارای یک فیلد پیوند باشد، لیست **تک پیوندی** نام دارد. در این صورت پیوند گره، به گره بعدی یا NULL[۴] اشاره می‌کند. اما، در لیست دو پیوندی که دارای دو پیوند است. پیوند دوم، به گره قبلی لیست اشاره می‌نماید.

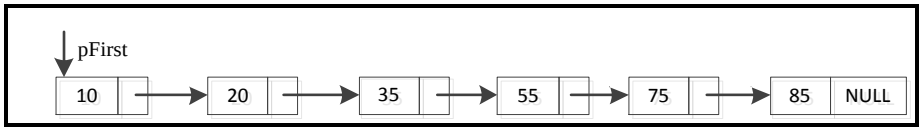
برای این که با مفهوم لیست پیوندی بیشتر آشنا شویم، شکل ۲ – ۴ را در نظر بگیرید. در این شکل، عناصر لیست، در آرایه یک بعدی به نام data ذخیره می‌گردند. همان طوری که در این آرایه می‌بینید، عناصر آرایه ترتیب خاصی ندارد (مرتب شده نیستند)، یعنی هر عنصر در هر جای آرایه قرار گرفته است. برای حفظ ترتیب واقعی عناصر، آرایه دیگری به نام link در نظر گرفته شده است. مقادیر این آرایه به عناصر آرایه data اشاره می‌کنند. چون آرایه با عنصر ۱۰ شروع می‌شود، پس متغیر شروع لیست (pFirst) برابر ۷ خواهد شد. بنابراین data[7]، یعنی مقدار ۱۰ اولین عنصر لیست است. آدرس عنصر بعدی لیست در link[7] قرار دارد. یعنی، عنصر بعدی در آدرس ۲ قرار دارد. داده بعدی در data[2] قرار دارد که ۲۰ است. آدرس عنصر بعدی در link[2] (همان آدرس ۳) قرار دارد. پس داده بعدی برابر با data[3] (مقدار ۳۵) است. آدرس عنصر بعدی در link[3] (آدرس ۸) قرار دارد و مقدار داده بعدی link[8] (آدرس صفر) است و داده بعدی در data[0] قرار دارد که همان مقدار ۷۵ می‌باشد. اما آدرس داده بعدی در link[0] (آدرس ۱۰) قرار دارد. داده بعدی مقدار data[10] (۸۵) است. آدرس داده بعدی در link[10] قرار دارد که NULL می‌باشد. مقدار NULL، بیان می‌کند که پیمایش داده‌های لیست خاتمه یافته است. عملیات پیمایش داده‌های لیست به صورت زیر است:



این عملیات در شکل ۲ – ۴ آمده است.



شکل ۲ – ٤ پیاده‌سازی لیست پیوندی با آرایه‌ها.



شکل ۳ – ٤ نمایش عناصر لیست پیوندی.

همان‌طوری که دیده‌اید، ۱. گره‌های لیست پیوندی به صورت پشت سرهم در حافظه قرار نمی‌گیرند. ۲. مکان گره‌ها در اجرای‌های مختلف می‌تواند تغییر یابد. اکنون مراحل درج عنصری ۴۵ در لیست در زیر آمده است (شکل ۴ – ۴):

---

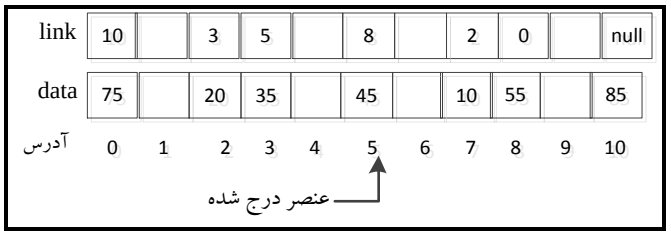
[۴]. برای آخرین گره لیست، پیوند برابر Null خواهد بود.

۱. حافظه‌ای بری مقدار ۴۵ اخذ می‌کنیم. فرض کنید این مقدار در آدرس ۵ قرار گرفته است.

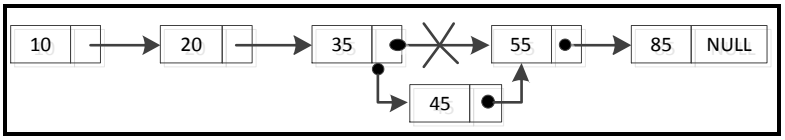
۲. مقدار داده آدرس ۵ برابر۴۵ قرار می‌گیرد. یعنی، data[5] = 45.

۳. فیلد ۴۵ باید به گره بعد از ۳۵ که ۵۵ است، اشاره کند (دستور مقابل): link [5] = link[3]

۴. فیلد پیوند گره‌ای که شامل ۵۵ است، به آدرس مقدار ۴۵ اشاره کند (دستور مقابل): link[3] = 5

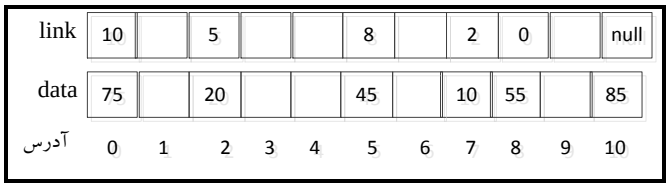


شکل ٤ – ٤ (الف) درج مقدار ٤٥ در محل data[5] و تغییر مقادیر آرایه link[5] و link[3].

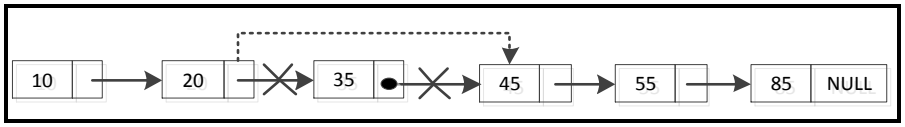


شکل ٤ – ٤ (ب) درج مقدار ٤٥ بعد از ۳۵.

اکنون اگر بخواهید گره‌ای با مقدار ۳۵ را حذف کنید، در این صورت، گره با داده ۲۰ باید به گره با داده ۴۵ اشاره کند (شکل ۵ – ۴): link [2] = link [5]



شکل ٥ – ٤ (الف) حذف مقدار ۳۵ (مقدار data[35]) از لیست از طریق آرایه.



شکل ٥ – ٤ (ب) حذف مقدار ۳۵ از لیست پیوندی.

همان‌طور که دیده‌اید، عیب لیست پیوندی نسبت به آرایه این است که هر گره نیاز به حافظه اضافی به نام پیوند دارد.

از لحاظ نوع پیوند چهار نوع مختلف لیست پیوندی وجود دارد که عبارت‌اند از:

۱. **لیست تک پیوندی**، در این نوع لیست پیوندی، پیوند هر گره آدرس گره بعدی را در خودش نگهداری می‌کند. لیست تک پیوندی (**یک طرفه**)، دارای یک پیوند است (این پیوند جهت اشاره به گره بعدی به کار می‌رود و آخرین گره در لیست پیوندی یک طرفه به جایی اشاره نمی‌کند (شکل ۶ – ۴). (یعنی، مقدار پیوند آن Null (تهی) است.

۲. **لیست پیوندی حلقوی**، یک نوع لیست پیوندی یک طرفه است که آخرین گره به اولین گره لیست اشاره می‌کند (شکل ۶- ۴) مقدار پیوند آخرین گره Null نیست).

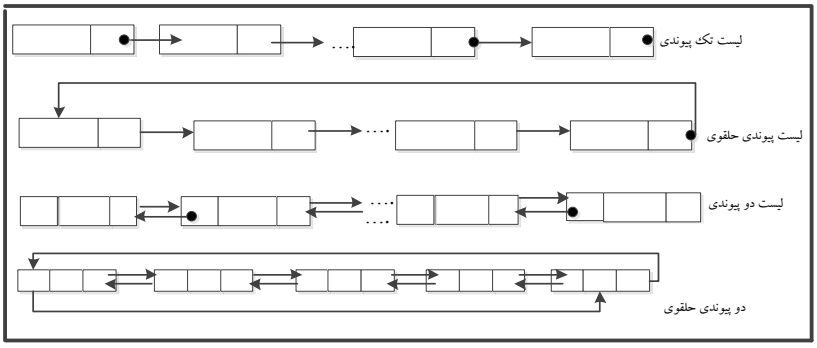
۳. **لیست دو پیوندی (دو طرفه)**،در این نوع لیست هر گره دارای دو پیوند است. یک پیوند، آدرس گره بعدی (مانند لیست پیوند یک طرفه) و پیوند دیگر آدرس گره قبلی را نگهداری می‌کند. بنابراین، در این نوع لیست پیوندی امکان حرکت به گره بعدی و گره قبلی وجود دارد. به همین دلیل، این نوع لیست‌های پیوندی، دو طرفه نیز نام دارند (شکل ۶ - ۴).

۴. **لیست دو پیوندی حلقوی**، یک نوع لیست دو پیوندی است که اشاره‌گر به گره قبلی در اولین گره به آخرین گره اشاره می‌کند و اشاره‌گر به گره بعدی در اولین گره به آخرین گره اشاره می‌کند. به عبارت دیگر، گره قبلی اولین، گره آخرین گره خواهد بود و گره بعدی آخرین گره، اولین گره خواهد بود.

۱ – ۴. لیست تک پیوندی

همان‌طور که بیان گردید، در لیست پیوندی یک طرفه، هر گره دارای بخش **داده** و **پیوند** است. برای ایجاد و استفاده از لیست پیوندی باید عملیات زیر انجام شود:

۱. تعریف کلاس هر گره در لیست پیوندی.
۲. تعریف کلاس پیوند.
۳. تعریف نمونه‌ای برای آدرس اولین گره لیست پیوندی.
۴. پیاده‌سازی عملیات اساسی روی لیست پیوندی.
۵. استفاده از عملیات اساسی پیاده‌سازی شده جهت پردازش و مدیریت لیست پیوندی.



شکل ۶- ۴. انواع لیست پیوندی.

۱ – ۱ – ۴. تعریف ساختار هر گره در لیست پیوندی

```
class Link
{
 public:
  int iData;
  double dData;
  Link* pNext;
 Link(int id, double dd) :
```

ساختار هر گره در لیست پیوندی از دو بخش داده و پیوند تشکیل شده است. برای تعریف ساختار گره می‌توان از یک کلاس به صورت زیر استفاده نمود:

```
iData(id), dData(dd), pNext(NULL)
{}
...
};
```

این دستورات، کلاسی به نام Link تعریف می‌کنند که دارای دو فیلد داده به نام‌های iData (عددی صحیح) و dData (عددی اعشاری) است. فیلد pNext از نوع اشاره‌گر می‌باشد که به گره بعدی اشاره خواهد کرد. این کلاس متد سازنده‌ای به نام Link دارد که id و dd را از نوع int و double به عنوان پارامتر دریافت می‌کند، در فیلدهای iData و dData قرار می‌دهد و مقدار NULL را در pNext قرار می‌دهد.

۲ – ۱ – ۴. تعریف اشاره‌گری برای نگهداری آدرس اولین گره لیست پیوندی

در هر لیست پیوندی باید آدرس اولین گره از لیست پیوندی را ذخیره نمود. اگر آدرس اولین گره از لیست پیوندی را ذخیره نکنید، لیست پیوندی در حافظه گم خواهد شد (لیست پیوندی به عنوان زباله در حافظه نگهداری می‌شود). معمولاً آدرس اولین گره از لیست پیوندی را به نام first یا head نام گذاری می‌کنند. دستور زیر را در نظر بگیرید:

```
Link*    pFirst;
```

این دستور اشاره‌گر pFirst را از نوع کلاس Link تعریف می‌کند.

پیاده‌سازی ۱ – ۴. برنامه‌ی که لیست پیوندی با امکانات افزودن گره‌ای به ابتدای لیست، حذف گره‌ای از ابتدای لیست، نمایش گره‌های لیست پیوندی یک طرفه را پیاده‌سازی می‌کند.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
class Link  {
    public:
    int iData;
    double dData;
    Link* pNext;
    Link(int id, double dd) :  iData(id), dData(dd), pNext(NULL) { }
    void displayLink()   {   cout << "{" << iData << ", " << dData << "} ";  }
    };
class LinkList   {
    private:
      Link* pFirst;
    public:
       LinkList() : pFirst(NULL)  { }
return pFirst==NULL;  }        bool isEmpty()   {
 void insertFirst(int id, double dd)  {
       Link* pNewLink = new Link(id, dd);
      pNewLink->pNext = pFirst;
      pFirst = pNewLink;
  }
return pFirst;   }   Link* getFirst()   {
  void removeFirst()   {
   Link* pTemp = pFirst;
   pFirst = pFirst->pNext;
   delete pTemp;
 }
  void displayList()  {
    cout << "List (first-->last): ";
    Link* pCurrent = pFirst;
    while(pCurrent != NULL)    {
      pCurrent->displayLink();
      pCurrent = pCurrent->pNext;
   }
   cout << endl;
  }
};
int main() {
   int  id;
   char choose;
   double  dd;
   LinkList theList;
   Link* pTemp;
{    while(true)
      cout << "Enter number of 1:Insert First\t 2:Remove First 3:Display All
4:Exit: ";
           cin >> choose;
    if (choose == '4' ) break;
    switch(choose)     {
            case '1':
       cout <<  "Enter id, dd:";
       cin >> id >> dd;
```

```
      theList.insertFirst(id, dd);
           break;
          case '2':
       pTemp = theList.getFirst();
  cout  <<  "Removing link with key " << pTemp->iData << endl;
            theList.removeFirst();
  break;
case '3':
  theList.displayList();
  break;
     default:
  cout << "Error !!!!!!!" << endl;
  break;
   }
}
    while( !theList.isEmpty() )   {
        pTemp = theList.getFirst();
         cout << "Removing link with key " << pTemp->iData << endl;
        theList.removeFirst();
     }
  getchar();
  return 0;
}
```

این برنامه از کلاس‌های Link و LinkList و تابع main() تشکیل شده است که شرح کلاس‌های Link و LinkList را قبلاً دیده‌اید. امّا، توضیح تابع main() در زیر آمده است:

تابع main() اعمال زیر را انجام می‌دهد:

☒ **متغیرهای id** (بخش صحیح داده لیست پیوندی)، dd (بخش اعشاری داده لیست پیوندی)، choose (کاراکتری که کاربر برای انجام اعمال افزودن، حذف، نمایش و غیره وارد می‌کند)، theList (نمونه‌ای از کلاس LinkList) و اشاره‌گر ptemp (اشاره‌گری از نوع Link) را تعریف می‌کند.

☒ با استفاده از while(true) یک حلقه بی‌نهایت ایجاد می‌کند و داخل این حلقه اعمال زیر را انجام می‌دهد:

۱. ابتدا یک پیغام نشان داده تا کاربر یکی از کاراکترهای '1' تا '4' را وارد کند. سپس کاراکتر وارد شده توسط کاربر را می‌خواند و در متغیر choose قرار می‌دهد.

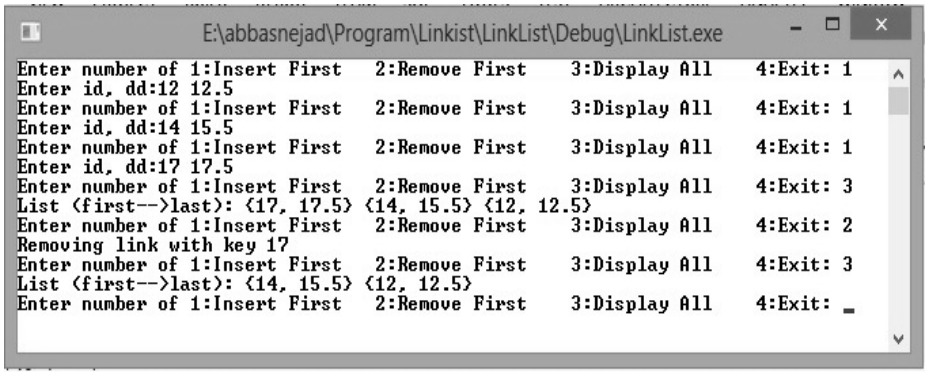
۲. اگر کاراکتر وارد شده '4' باشد، حلقه while بی‌نهایت را خاتمه می‌دهد.

۳. اگر کاراکتر وارد شده '1' باشد، با نمایش یک پیغام دو عدد را از کاربر می‌خواند و با فراخوانی تابع insertFirst() آن‌ها را به ابتدای لیست پیوندی اضافه می‌کند.

٤. اگر کاراکتر وارد شده '2' باشد، با فراخوانی تابع getFirst()، عنصر ابتدای لیست را نمایش می‌دهد. سپس با فراخوانی متد remove() عنصر ابتدای لیست پیوندی را حذف می‌کند.

۵. اگر کاربر کاراکتر '3' را وارد کرده باشد، با فراخوانی متد displayList()، تمام عناصر موجود در لیست پیوندی را نمایش می‌دهد.

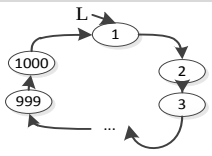
☒ در پایان تا زمانی که لیست پیوندی خالی نباشد (حلقه while دوم)، عناصر لیست پیوندی را نمـایش داده و آن‌ها را از لیست پیوندی حذف می‌کند.



۸ – ۴. تست‌های ارشد لیست پیوندی

۱. با فرض اجرای تابع بر روی لیست پیوندی حلقوی شکل زیر، مقدار خروجی چقـدر اسـت؟ (مهندسـی فنـاوری اطلاعات - سال ۸۸).

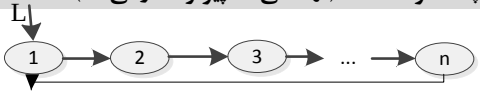
```
int S(List *L) {
   if (L → next == L) return L → data;
L → next = L → next → next;
return S(L → next);
}
```



الف: ۱ ب: ۳۲۷ ج: ۹۷۷ د: ۱۰۰۰

۲. با توجه به تابع روبه رو لیست حلقوی مذکور به ازای مقادیر n برابر ۷۲۹ و ۲۲۰۰ مقدار خروجی به ترتیـب برابـر چند خواهد شد؟ (مهندسی کامپیوتر – دولتی ۸۹).

```
int So(List *L) {
while (L → next != L) {
     L → next = L → next → next;
     L = L → next;
  }
return (L → data);
}
```



الف: ۱ و ۴۰ ب: ۱ و ۱ ج: ۷۲۹ و ۲۲۰۰ د: هیچ‌کدام

۳. الگوریتم زیر کدام عملیات را انجام می‌دهد؟ (مهندسی کامپیوتر – آزاد ۸۷).

```
void fun (Dlink *p, Dlink * t)    {
  p → llink = t;
  p → rlink = t → rlink;
  t → rlink → l link = p;
  t → rlink = p;
 }
```

الف: گره t را به سمت راست گره p اضافه می‌کند.

ب: گره t را از لیست حذف نموده و p را جایگزین آن می‌نماید.

ج: گره p را به سمت راست گره t اضافه می‌کند.

د: گره p را از لیست حذف نموده و t را جایگزین آن می‌نماید.

۴. خروجی حاصل از اجرای ()proce به ترتیب از راست به چپ چیست؟ (مهندسی کامپیوتر – آزاد ۸۸).

```
void proce()   {
char *s[3] = {"red","green",
"blue"};
cout<< s[1];
cout << "\n" << (s[1] + 1);
}
```

الف: blue ،green

ب: green ،red

ج: green و خروجی یک آدرس را نمایش می‌دهد.

د: reen ،green

۵. اگر بخواهید رشته‌ای بـه طـول t را در مکـان iام رشته دیگر درج کنیم، به ترتیب در حالتی که لیست غیر دوری (غیر حلقوی) و دوری (حلقوی) باشد، زمان عملیات متناسب است با: (مهندسی کامپیوتر – آزاد ۸۱).

الف: O(i + t) ،O(i + t)

ب: O(t) ،O(i + t)

ج: O(i + t) ،O(i)

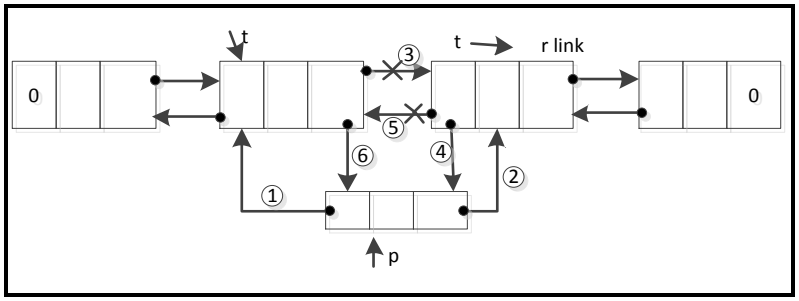
د: O(t) و O(t)

۹ – ۴. پاسخ تشریحی تست‌های ارشد لیست پیوندی

۱. گزینه (ج) صحیح اسـت. چـون دسـتور ;next → next →next = L → L، در بـار اول، گـره‌هـای زوج را از لیست خارج می‌نمایـد. بنـابراین اعـداد 1 → 999 → 997 → ... → 3 → 1 پیمـایش مـی‌شـوند، پـس گزینـه (د) (مقدار ۱۰۰۰) نادرست است. اما، در مرتبه دوم اجرا، این تابع اعداد ۳، ۷، ۱۳، ۱۷، ۲۱ و ... تا 4n – 1 را از لیست خارج می‌نماید. پس گزینه (ب) مقدار ۳۲۷ (۱– ۸۲ * ۴) نادرست می‌باشد. ولی در مرتبه سوم، اعداد ۵، ۱۳، ۲۱، ۲۹، ۳۷، ... و 8n – 3 از لیست حذف خواهند شـد. در نهایـت در مرحلـه آخـر (چهـارمین مرتبـه) اعـداد ۹، ۲۵، ۴۱، ... 16n – 7 از لیست حذف می‌گردند. در این مرحله عدد ۹۸۵ حذف می‌گردد. سپس ۹۷۷ به ۹۹۳ اشاره می‌نماید و بعد از آن ۹۹۳ نیز عدد ۱ را رد می‌کند. بنابراین گزینه (الف) نیز نادرست است.

۲. گزینه (د) صحیح است. چون دستور ;next → next → L = next → L باشد، با شروع از مقدار ۱، اعـداد فـرد پیمایش می‌گردند و اعداد زوج حذف می‌شوند. پس، گزینه (الف) نادرست است. در مرتبه دوم، مقادیر ۱، ۵، ۹، ۱۳، ... و 4n – 1 از لیست حذف می‌شوند. چون ۷۲۹ برابر با 1 – 182 * 4 است. پس، این گزینه نیز از لیست حذف خواهـد شد. بنابراین گزینه‌های (ب) و (ج) نیز نادرست‌اند.

۳. گزینه (ج) صحیح است. زیرا، اگر گره‌ها را به شکل زیر در نظر بگیرید، متوجه خواهید شد:



۴. گزینه (د) صحیح است. چون، اطلاعات به صورت زیر در اشاره‌گر ذخیره می‌شوند:

S[0]	S[1]	S[2]
red	green	bule

پس دستور cout << s[1]; مقدار green را نمایش خواهد داد. اما، دستور cout << "\n" << (S[1] + 1);، ابتدا به خط بعد می‌رود و آدرس شروع S[1] را یک واحد اضافه می‌کند، یعنی اشاره‌گر را به r می‌برد تا انتهای آن یعنی reen را نمایش می‌دهد.

۵. گزینه (ج) صحیح است. در لیست ساده (غیر حلقوی)به اندازه O(i) زمان نیاز است تا به مکان درج انتقال یابد و به اندازه O(t) نیز زمان نیاز است تا به انتهای رشته برسیم. پس در لیست ساده، زمان O(t + i) مورد نیاز است. اما، در لیست حلقوی، چون آدرس آخرین عنصر (عنصر tام) را داریم، پس زمان O(t) حذف می‌شود و فقط زمان O(i) می‌ماند.

فصل ۵ درخت

درخت همانند لیست پیوندی یکی از ساختمان داده‌های پویا و غیر ترتیبی است. درخت ساختمان داده‌ای پویا است. یعنی، هنگام درج عنصر جدید در درخت فضای مورد نیاز عنصر به درخت اضافه می‌شود و هنگام حذف عنصر نیز فضای مورد نظر، آزاد خواهد شد. از این رو، مصرف حافظه در درخت نسبت به ساختمان داده‌های ایستا مانند آرایه بهینه‌تر است. به عنوان مثال، اگر آرایه‌ای ۱۰۰ عنصری تعریف نماییم و از ۱۰ خانه‌ی آن استفاده نماییم، ۹۰ خانه‌ی دیگر آرایه استفاده نشده، ولی فضای حافظه را اشغال کرده است. از طرف دیگر نمی‌توان در این آرایه ۱۱۰ عنصر ذخیره نمود. چرا که در ابتدا آرایه ۱۰۰ خانه‌ای تعریف شده است. ولی درخت به عنوان ساختمان داده‌ای پویا این دو مشکل را ندارد. چون هنگام درج، فضای جدید اختصاص داده می‌شود و هنگام حذف، فضای اختصاص داده شده آزاد می‌شود.

درخت ساختمان داده‌ای غیر ترتیبی یا غیر خطی است. یعنی، عناصر درخت در حافظه می‌توانند پشت سرهم قرار نگیرند. چون عناصر ساختمان داده‌های ترتیبی در حافظه پشت سر هم قرار دارند، دسترسی به آن‌ها سریع‌تر انجام خواهد شد. ولی ساختمان داده‌های پویا نمی‌توانند ترتیبی باشند. چون درج و حذف در ساختمان داده‌های پویا منجر به اختصاص و آزادسازی فضا در حافظه خواهد شد و ترتیب عناصر به هم خواهد خورد.

عمل درج و حذف در آرایه مرتب کند بوده و از درجه $O(n)$ است. عمل جست‌وجو نیز در لیست پیوندی کند بوده و از درجه $O(n)$ است. یکی از دلایل اصلی استفاده از درخت، ترکیب نقاط قوت دو ساختمان داده‌ی آرایه‌ی مرتب و لیست پیوندی است. عمل جست‌وجو در درخت مانند آرایه مرتب سریع است. از طرف دیگر، عمل درج و حذف در درخت مانند لیست پیوندی سریع است.

ساختمان داده‌ی درخت کاربردهای فراوانی دارد. از آن جمله می‌توان به انواع ذخیره‌سازی داده‌ها و اطلاعات، تبدیل و محاسبه عبارت‌های ریاضی، مرتب‌سازی، جست‌وجو، نمایش داده‌ها با ساختار سلسله مراتبی مانند پوشه‌ها در ویندوز اکسپلورر، فهرست مطالب و غیره اشاره نمود. درخت‌ها زیر مجموعه ساختمان داده‌ای به نام گراف هستند. درخت‌ها به دو دسته‌ی درخت‌های دودویی و درخت‌های عمومی تقسیم بندی می‌شوند که در ادامه هر کدام از آن‌ها را بررسی می‌نماییم.

۱-۵. تعاریف

برای این که کاربردهای مختلف درخت را بیان نماییم، ابتدا به چند تعریف در بخش درخت می‌پردازیم:

☒ **درخت:** مجموعه‌ی محدودی از یک یا چند گره[۵] که:

۱. دارای گره خاصی به نام ریشه[۲] باشد.

---

[5]Node [2].root

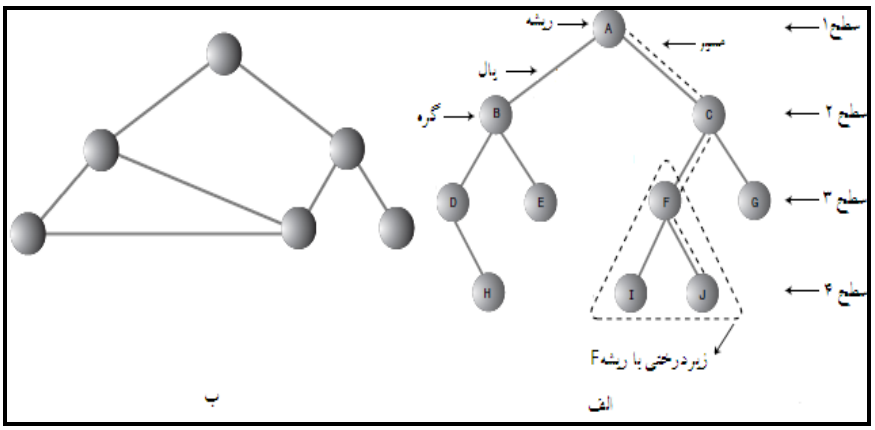
۲. بقیه گره‌ها به مجموعه‌های مجزای $T_1, T_2, \ldots, T_n$ تقسیم می‌شوند که هر یک از این مجموعه‌ها خود یک درخت هستند. $T_1, T_2, \ldots, T_n$ زیر درخت[۶] های ریشه نامیده می‌شوند.

نکته: درخت دور ندارد. یعنی، از یک گره به گره دیگر تنها یک مسیر وجود دارد.

نکته: ریشه در درخت کامپیوتری در بالا قرار دارد.

نکته: یک درخت با n گره دارای n-1 یال است.

در شکل ۱-۵ الف یک نمونه درخت را مشاهده می کنید. شکل ۱-۵ ب دور دارد پس درخت نیست. در فصل بعد خواهید دید که شکل ۱-۵ ب یک گراف است.



شکل ۱- ۵ الف) یک نمونه درخت ب) دور دارد و درخت نیست.

☒ **گره**، به عناصر موجود در درخت گره گویند. برای راحتی هر گره را با یکی از حروف الفبا نشان می‌دهند.

☒ **یال**، به خطی که از یک گره به گره بعدی رسم می‌شود، یال[۲] گویند.در شکل ۱-۵ الفFI و BD دو نمونه یال هستند.

☒ **گره فرزند و والد**، اگر از گره X توسط یک یال به سمت پایین به گره Y متصل باشد. گره Y فرزند[۳] X است و گره X پدر یا والد[۴] Y است. به عنوان مثال، در شکل ۱-۵ الف C پدر F و G است. D و E فرزندان B هستند ریشه تنها گره‌ای است که والد ندارد.

☒ **درجه گره**، به تعداد زیر درخت‌های یک گره، درجه[۵] گره گویند. به عبارت دیگر، به تعداد فرزندان یک گره گویند. در شکل ۱-۵ الف، درجه‌ی ریشه ۲ و درجه‌ی گره‌ی D برابر ۱ است.

☒ **برگ**، گره با درجه‌ی صفر را گویند. به عبارت دیگر، گره‌ای که فرزند نداشته باشد. در شکل ۱- ۵ الف، گره‌های E, F, G, H, J, I برگ هستند. به گره‌های برگ گره‌های خارجی[۶] نیز می گویند.

☒ **گره داخلی**، گره‌های غیر برگ را گره داخلی[۷] گویند. به عبارت دیگر، گره‌هایی که حداقل یک فرزند داشته باشند. به عنوان مثال، در شکل ۱-۵ الف، گره‌های D, C, B, A, F گره‌های داخلی هستند.

☒ **درجه‌ی درخت**، حداکثر درجه گره‌های درخت را گویند. درجه‌ی درخت شکل ۱-۵ الف برابر ۲ است.

---

[6].subtree [2].edge [3].child [4].parent [5].degree [6].external [7].Internal

☒ **گره‌های همزاد یا هم نیا**، فرزندان یک گره را گره‌های همزاد[۷] گویند. به عبارت دیگر، گره‌هایی که دارای والد مشترک هستند. به عنوان مثال، گره‌های I, J با هم و گره‌های B, C با هم همزادند.

☒ **سطح گره**، هر گره دارای سطحی است. سطح[۲] گره ریشه یک است و سطح هر گره دیگر یک واحد بیشتر از گره والد خودش است. به عنوان مثال، گره‌ی D در سطح ۳ قرار دارد. بعضی از کتاب‌ها سطح ریشه را صفر در نظر می‌گیرند.

☒ **عمق یا ارتفاع درخت**، به بیشترین سطح گره‌های درخت عمق[۳] یا ارتفاع گویند. سطح درخت ۱-۵ الف، برابر ۴ است.

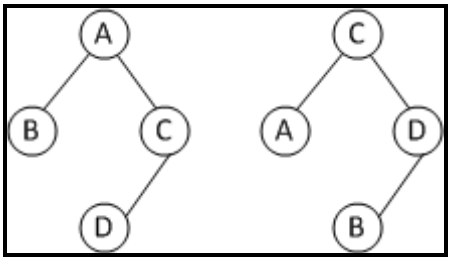
☒ **جدد گره**، گره‌هایی هستند که در مسیر طی شده از ریشه تا آن گره وجود دارند. به عنوان مثال، در شکل ۱-۵ الف، گره‌های A, C, F اجداد گره‌ی J هستند.

☒ **نسل گره**، اگر گره X جد گره Y باشد، گره Y نسل گره‌ی X است. به عنوان مثال، در شکل ۱- ۵ الف، گره I نسل گره A, C, F است.

☒ **مسیر**، یک یال و یا دنباله‌ای از یال‌های متوالی را مسیر[۴] گویند. به عنوان مثال، در شکل ۱- ۵ الف، چند نمونه مسیر نظیر ACFJ, BDH, AB را می‌بینید.

☒ **شاخه**، مسیری که به یک برگ ختم می‌شود را شاخه[۵] گویند. به عنوان مثال، در شکل ۱- ۵ الف، چند شاخه نظیر ACFJ, CFI آمده‌اند.

☒ **درخت مشابه**، درخت‌هایی که دارای ساختار و شکل یکسان هستند را درخت مشابه گویند. دو درخت مشابه در شکل ۲- ۵ نشان داده شده است. درخت‌های مشابه‌ای که محتوای گره‌های متناظر آن‌ها نیز یکسان باشد را درخت کپی گویند.



شکل۲- ۵ درخت‌های مشابه.

☒ **درخت kتایی**، درختی که تعداد فرزندان هر گره آن حداکثر K است. در شکل ۱-۵ الف، یک درخت ۲تایی را می‌بینید.

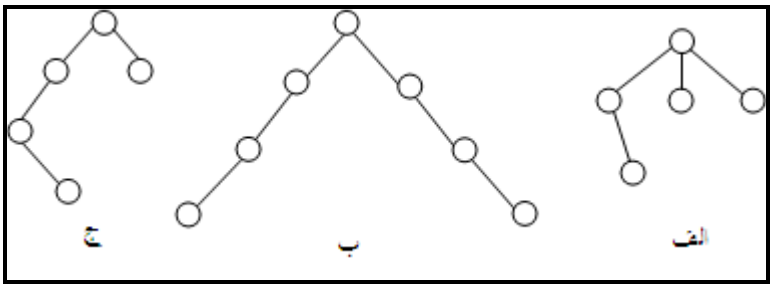
نکته: در درخت Kتایی تعداد برگ‌ها برابر است با $n_0=(k-1)n_k+(k-2)n_{k-1}+\ldots+n_2+1$ که تعداد گره‌ها از درجه‌ی K است.یعنی، برای درخت ۴تایی فرمول تعداد برگ بدین صورت است: $n_0=3n_4+2n_3+n_2+1$

☒ **درخت متوازن**، درختی که اختلاف سطح برگ‌های آن حداکثر یک باشد. درخت‌های الف و ب شکل ۳ - ۵ متوازن بوده و درخت ج متوازن نیست. اگر اختلاف سطح برگ‌ها صفر باشد (یعنی، همه‌ی برگ‌ها در

---

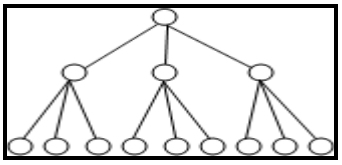
[7]. Sibling [2]. Level [3].depth [4].path [5]. branch

یک سطح باشند) آن درخت کاملاً متوازن است. درخت شکل ۳- ۵ ب کاملاً متوازن است. در قسمت درخت‌های AVL تعریف متفاوتی از درخت‌های متوازن ارائه خواهد شد.

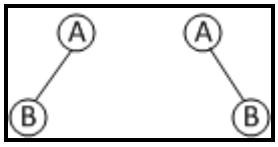


شکل۳-۵ الف) درخت متوازن ب) درخت کاملاً متوازن ج) درخت غیر متوازن.

☒ **درخت پر**، درختی است که درجه‌ی تمام گره‌های داخلی آن یکسان باشد و تمام برگ‌های آن در یک سطح قرار داشته باشند.



شکل ۴-۵ درخت پر.



شکل ۵- ۵ دو درخت دودویی متفاوت.

نکته: تعداد گره‌های درخت پر با درجه‌ی d و ارتفاع h برابر با $\frac{d^h-1}{d-1}$ است. حداکثر تعداد گره‌های درخت از درجه‌ی d و به ارتفاع h نیز برابر همین مقدار است.

نکته: تعداد برگ در درخت kتایی پر با n گره برابر است با $n - \frac{n-1}{k}$ است.

۲-۵. درخت دودویی

درخت دودویی[8] درختی است که یا خالی (تهی) است و یا هر گره آن حداکثر دو فرزند دارد. هر گره در درخت دودویی از زیر درخت چپ و راست تشکیل شده است.

نکته: درخت‌های دودویی می‌توانند تهی (پوچ) باشند. ولی درخت‌های عمومی نمی‌توانند تهی باشند.

نکته: ترتیب فرزندان در درخت دودویی اهمیت دارد. در صورتی که در درخت‌های عمومی این ترتیب مهم نیست. به عنوان مثال، درخت‌های الف و ب در شکل ۵-۵، اگر دودویی در نظر گرفته شوند با یک دیگر تفاوت دارند. ولی اگر عمومی در نظر گرفته شوند، یکسان هستند.

☒ **قضیه**، در درختان دودویی داریم: $n_0=n_2+1$. در این فرمول n_0 تعداد گره‌ها از درجه‌ی صفر (تعداد برگ‌ها) و تعداد گره‌ها از درجه‌ی دو است.

☒ **اثبات**، تعداد کل گره‌ها برابر است با مجموع گره‌ها از درجه‌ی صفر، یک و دو $n = n_0 + n_1 + n_2$ (1)

همچنین تعداد یال‌ها (y) در درخت یکی کمتر از تعداد گره‌ها است. $n = y + 1$ (2)

[8].Binary tree

از طرف دیگر، یک یال به گره‌ها با درجه‌ی یک و دو یال به گره‌ها با درجه‌ی دو متصل هستند و مجموع آن‌ها برابر تعداد کل یال‌ها است.

$$y = n_1 + 2n_2 \quad (3)$$

از جای گذاری رابطه (3) در رابطه‌ی (2) داریم:

$$n = n_1 + 2n_2 + 1 \quad (4)$$

و از تساوی رابطه‌ی (1) و (4) داریم:

$$n_0 + n_1 + n_2 = n_1 + 2n_2 + 1 \rightarrow n_0 = n_2 + 1$$

۵-۵. درخت‌هایی با ساختار ویژه

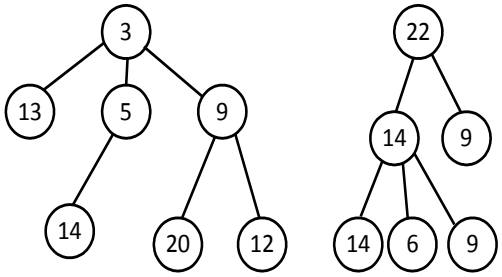
ساختار درختانی که تا به اینجا بررسی کردیم، بر اساس مقدار گره‌ها نبود. ولی، درختان خاصی وجود دارند که ساختار آن‌ها بر اساس مقدار گره‌های درخت است. این درختان کاربردهای فراوانی در علوم کامپیوتر دارند. درختانی که در اینجا بررسی خواهیم کرد عبارتند از: Heap، درخت جست‌وجوی دودویی، AVL، درخت انتخابی، درخت هافمن و درخت تصمیم.

۵-۵-۱. درخت Heap

درخت heap که به آن هرم یا کپه هم می‌گویند، به دو دسته‌ی Maxheap یا هرم حداکثر Minheap هرم حداقل تقسیم می‌شود. اگر نوع heap مشخص نشود، منظور Maxheap است. برای تعریف Maxheap و Minheap به تعریف دو درخت زیر می‌پردازیم:

☒ **Maxtree**، درختی است که مقدار هر گره بیشتر یا مساوی فرزندانش باشد. به Maxtree، درخت حداکثر هم گویند. در شکل ۴۲-۵ الف یک نمونه درخت maxtree نشان داده شده است.

☒ **Mintree**، درختی است که مقدار هر گره کوچک‌تر یا مساوی فرزندانش باشد و به Mintree، درخت حداقل هم گویند. در شکل ۴۲-۵ ب یک نمونه درخت mintree نشان داده شده است.

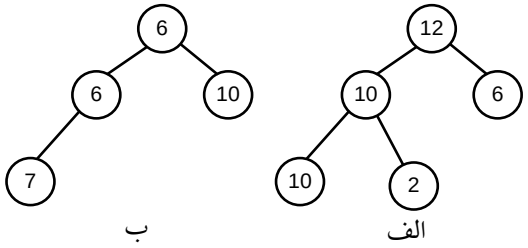


شکل ۴۲-۵ الف) درخت maxtree ب) درخت mintree.

☒ **Maxheap**، درخت دودویی کاملی است که Maxtree هم باشد. در شکل ۴۳-۵ الف یک نمونه درخت maxheap نشان داده شده است.

☒ **Minheap**، درخت دودویی کاملی است که Mintree هم باشد. در شکل ۴۳-۵ ب یک نمونه درخت minheap نشان داده شده است.

نکته: درMaxheap، بزرگ‌ترین مقدار گره‌ها در ریشه است. در Minheap، کوچک‌ترین مقدار گره‌ها در ریشه است.



شکل ۴۳-۵ الف) درخت maxheap ب) درخت .minheap

نکته: هر زیر درخت در درخت heap خود یک درخت heap است.

به طور معمول درخت heap را با استفاده از آرایه پیاده‌سازی می‌نمایند. به همان صورتی که درختان دودویی را پیاده سازی نمودیم. به عنوان مثال، شکل آرایه درخت Maxheap و Minheap شکل ۴۲-۵ به صورت شکل ۴۳-۵ خواهد بود.

0	1	2	3	4	5	6	7
	6	6	10	7			

0	1	2	3	4	5	6	7
	12	10	6	10	2		

شکل ۴۴-۵ آرایه مربوط به پیاده‌سازی درخت Maxheap و Minheap شکل ۴۲-۵.

درج گره در درخت Heap

درخت heap یک درخت کامل است. پس باید در تمام مراحل درج و حذف نیز کامل بماند. در عمل درج عنصر جدید در heap باید توجه داشت که تا زمانی که گره‌های یک سطح کامل نشده است، نمی‌توانیم به سطح بعدی برویم. همچنین اضافه کردن گره جدید در هر سطح از سمت چپ به راست است.

برای عمل درج در Maxheap به صورت زیر عمل می‌شود:

۱. گره جدید را به اولین جای خالی از سطح چپ و از آخرین سطح اضافه می‌کنیم.

۲. گره جدید را با والدش مقایسه می‌نماییم. اگر بزرگ‌تر بود، جای والد و فرزند را عوض می‌نماییم.

۳. عمل مرحله‌ی ۲ را تا زمانی که گره جدید از والدش بزرگ‌تر نباشد و یا به ریشه برسد، ادامه می‌دهیم.

عمل درج در Minheap مشابه عمل درج در Maxheap است. با این تفاوت که در صورتی که مقدار گره جدید کوچک‌تر از والدش باشد، جای آن دو عوض می‌شود.کد مربوط به پیاده سازی درج در heap به صورت شکل ۴۴-۵ خواهد بود. همان‌طور که مشاهده می‌شود، در این کد درخت heap با آرایه پیاده‌سازی شده است.

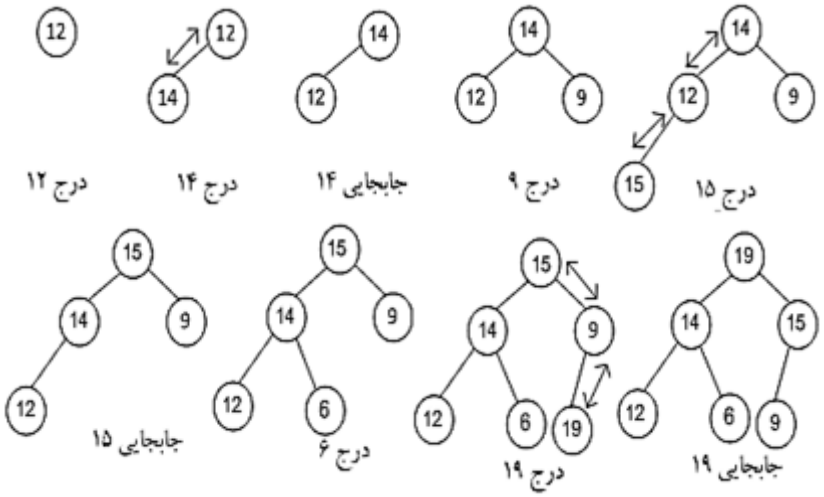
```
void insert (element item, int *n) {
  int i;
  i=++(*n);
  while((i !=1) && (item.key > heap[i/2].key))    {
      heap[i]=heap[i/2];
      i /=2;
  }
  heap[i]=item;
```

}

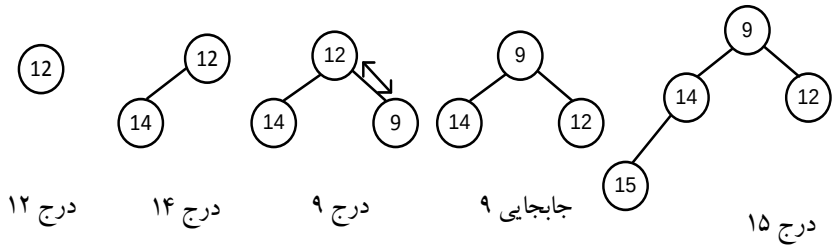
شکل ۴۵-۵ کد پیاده سازی درج در heap.

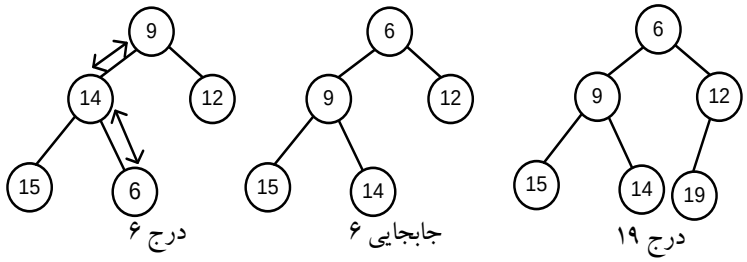
مثال۵ – ۵. اعداد مقابل را در یک درخت Maxheap و Minheap خالی درج نمایید.19 6 15 9 14 12

حل: درج در Maxheap



درج در minheap





تحلیل پیچیدگی زمانی عمل درج در heap

با درج گره جدید در heap در سطح آخر درخت، گره جدید حداکثر تا ریشه تغییر مکان می‌دهد. یعنی، حداکثر به اندازه‌ی ارتفاع درخت بالا می‌رود. از آن جا که heap یک درخت کامل است و ارتفاع درخت

کامل با n گره برابر با $[\log_2^n]+1$ است. پس، تابع زمانی عمل درج برابر $T(n)=[\log_2^n]+1$ است. در نتیجه، پیچیدگی زمانی عمل درج در heap برابر با $O[\log_2^n]$ می‌باشد.

حذف گره از درخت Heap

در حذف گره از درخت heap فقط می‌توانیم گره ریشه را حذف کنیم. در عمل حذف از درخت heap باید توجه نماییم که درخت heap بعد از عمل حذف نیز کامل باقی بماند. برای حذف گره از درخت heap به صورت زیر عمل می‌شود:

۱. گره ریشه را حذف می‌نماییم.

۲. راست‌ترین گره سطح آخر را به جای ریشه قرار می‌دهیم.

۳. درخت heap را با مقایسه گره‌ای جا به جا شده با فرزندانش دوباره تنظیم می‌نماییم.

عمل مقایسه در Maxheap بدین صورت است که اگر گره جا به جا شده تا جایی که از فرزندانش کوچک‌تر است با بزرگ‌ترین فرزندش جا به جا می‌شود. عمل مقایسه در Minheap هم به این صورت است که گره جا به جا شده تا جایی که از فرزندانش بزرگ‌تر است با کوچک‌ترین فرزندش جا به جا می‌شود. کد مربوط به پیاده سازی حذف از heap به صورت شکل ۴۵-۵ خواهد بود.

```
element delete (int *n) {
  int parent, child;
  element item, temp;
  if (heapEmpty(*n)) {
    cout << "The Heap is Empty"
     exit(1);
  }
  item=heap[1];
  temp=heap[(*n)--)];
  parent=1;
  child=2;
  while(child<=*n) {
      if((child<*n)&&(heap[child].key<heap[child+1].key)) child++;
     if(temp.key)>=heap[child].key)    break;
     heap[parent]=heap[child];
     child*=2;
  }
  heap[parent]=temp;
  return item;
}
```

شکل ۴۶-۵ کد پیاده سازی حذف از heap.

۴-۵-۵. درخت انتخابی

فرض کنید می‌خواهیم k آرایه مرتب شده صعودی را با هم ادغام نماییم تا به یک آرایه مرتب شده صعودی برسیم. برای حل این مسئله دو عمل زیر را آن قدر تکرار می‌نماییم تا هیچ عنصر در آرایه‌ها وجود نداشته باشد:

۱. خانه‌های اول همه‌ی آرایه‌ها را مقایسه کرده و کوچک‌ترین عنصر را پیدا نموده، در خروجی قرار می‌دهیم.

۲. عناصر آرایه‌ای که کوچک‌ترین عنصر از آن حذف شده را یک واحد به سمت ابتدای آرایه شیفت می‌دهیم تا جای خالی عنصر اول پوشانده شود.

به عنوان مثال، با توجه به آرایه‌های زیر ابتدا خانه اول همه‌ی آرایه‌ها مقایسه شده و کوچک‌ترین آن‌ها در خروجی قرار گرفته و آرایه ای که در آن عمل حذف انجام شده یک واحد شیفت به سمت ابتدای آرایه خواهیم داشت. به عنوان مثال، آرایه‌های مرتب شده‌ی شکل ۶۶-۵ را در نظر می‌گیریم.

1	2	3	4	5	6	7	8
19	8	1	5	11	6	3	14
22	14	10	9	17	16	8	17
25	32	17	19	18	29	15	20
30	35	24	23	31	36	27	26

شکل ۶۶-۵ آرایه‌های مرتب شده.

خانه‌های اول همه‌ی آرایه‌ها را مقایسه کرده و کوچک‌ترین یعنی ۱ از آرایه ۳ را بدست می‌آوریم. بعد از حذف ۱ و قرار دادن آن در خروجی، عناصر آرایه ۳ را یک خانه به سمت ابتدای آرایه شیفت می‌دهیم.

1	2	3	4	5	6	7	8
19	8	10	5	11	6	3	14
22	14	17	9	17	16	8	17
25	32	24	19	18	29	15	20
30	35		23	31	36	27	26

شکل ۶۷-۵ حذف کوچک‌ترین عنصر از میان اولین عناصر هر آرایه.

با تکرار این عمل مقدار ۳ از آرایه ۷ در خروجی قرار خواهد گرفت.

این عمل را تا خالی شدن همه‌ی آرایه‌ها ادامه می‌دهیم.

اگر k آرایه کوچک داشته باشیم، برای به دست آوردن کوچک‌ترین عنصر از عناصر اول آرایه‌ها نیاز به k-1 مقایسه است. اگر تعداد کل عناصر موجود در همه‌ی آرایه‌ها برابر n باشد، نیاز به n(k-1) عمل مقایسه است و پیچیدگی زمانی این روش برابر O(nk) خواهد بود. یک روش دیگر، برای انجام این کار، استفاده از درخت انتخابی[9] است که تعداد مقایسه‌ها را کاهش داده و پیچیدگی زمانی بهتری دارد.

1	2	3	4	5	6	7	8
19	8	10	5	11	6	8	14
22	14	17	9	17	16	15	17
25	32	24	19	18	29	27	20
30	35		23	31	36		26

[9]. Selection tree

شکل ۶۸-۵ حذف کوچک‌ترین عنصر از میان اولین عناصر هر آرایه.

تعریف درخت انتخابی

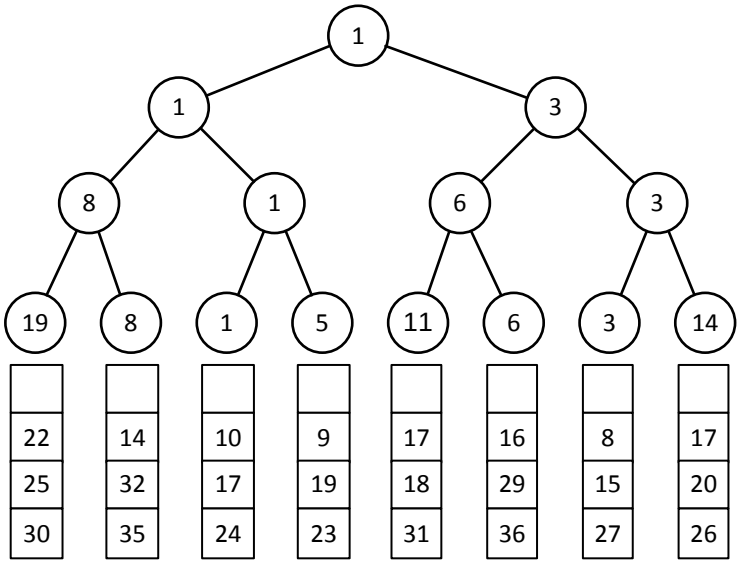
درخت انتخابی درخت دودویی کاملی است که مقدار هر گره، کوچک‌تر از مقدار فرزندانش باشد. همان‌طور که از تعریف درخت انتخابی مشخص است کوچک‌ترین گره درخت، در ریشه قرار دارد. برای ادغام آرایه‌ها با استفاده از درخت انتخابی بدین صورت عمل می‌نماییم که هر یک از آرایه‌ها را زیر یک برگ از سطح آخر درخت انتخابی قرار می‌دهیم. کوچک‌ترین عنصر هر یک از آرایه‌ها را وارد برگ‌های متناظرش در درخت می‌نماییم. از برگ‌ها شروع کرده و گره‌های همزاد را با یکدیگر مقایسه می‌نماییم و مقدار گره کوچک‌تر را در گره والد آن‌ها قرار می‌دهیم. مقایسه‌ی همزادها و قرار دادن مقدار کوچک‌تر در والد آن‌ها را تا ریشه‌ی درخت ادامه می‌دهیم. بدین صورت کوچک‌ترین مقدار در ریشه قرار می‌گیرد. مقدار ریشه را در خروجی قرار می‌دهیم. از آرایه‌ای که مقدار آن حذف شده خانه‌ی بعدی را به جای مقدار حذف شده در برگ درخت قرار داده و مقایسه با همزادش و قرار دادن کوچک‌ترین مقدار آن‌ها در والدشان را تا ریشه ادامه می‌دهیم. با این عمل کوچک‌ترین مقدار بعدی در ریشه قرار گرفته و آن را در خروجی قرار می‌دهیم و به همین صورت ادامه داده تا همه‌ی عناصر آرایه‌ها در خروجی قرار بگیرد. بدین صورت خروجی به صورت صعودی مرتب خواهد شد.

به عنوان مثال، همان آرایه‌های مرتب شده مثال قبل را در نظر می‌گیریم. ابتدا هر یک از خانه‌های ابتدایی را در برگ‌های درخت قرار می‌دهیم و برگ‌ها تا ریشه، همزادها را با یک دیگر مقایسه کرده و مقدار کوچک‌تر را در والد قرار می‌دهیم. درخت انتخابی آن در شکل ۶۸-۵ نشان داده شده است.

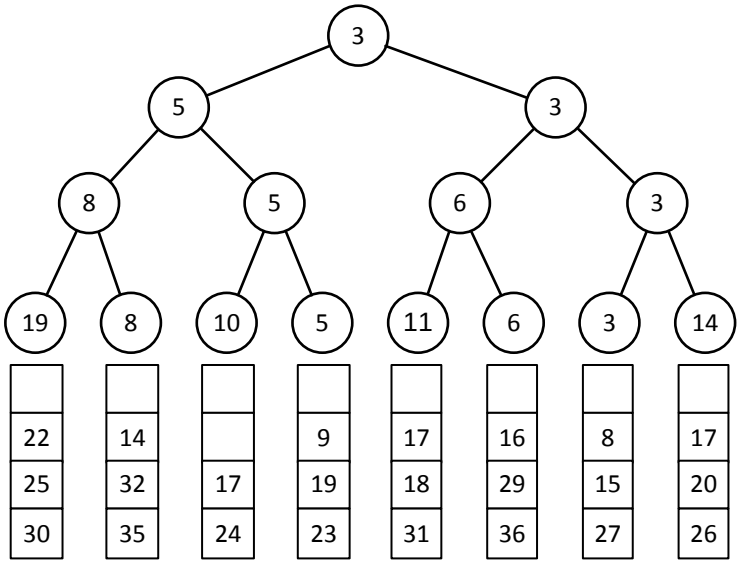
مقدار ریشه که برابر ۱ است، در خروجی قرار می‌گیرد و از آرایه‌ای که ۱ در آن بوده مقدار بعدی که ۱۰ است را به جای ۱ در برگ قرار می‌دهیم. برای ۱۰ عمل مقایسه با همزاد و قرار دادن کوچک‌تر در والد را تا ریشه انجام می‌دهیم تا ریشه بعدی مشخص شود.

کوچک‌ترین مقدار بعدی برابر ۳ است. آن را در خروجی قرار داده و مقدار بعدی از آرایه‌ای که ۳ از آن آمده (یعنی 8) را در برگ قرار داده و روند مقایسه را تا ریشه انجام می‌دهیم. این اعمال را تا زمانی که همه‌ی عناصر آرایه‌ها در خروجی قرار گیرند، تکرار می‌نماییم.

اگر k را تعداد آرایه‌ها در نظر بگیریم، زمان لازم برای ورود یک عنصر جدید به درخت و انجام اعمال مقایسه و جا به جایی تا ریشه برابر $O(\log_2 k)$ است. بنابراین، پیچیدگی زمانی ادغام n عنصر آرایه با استفاده از درخت انتخابی برابر $O(n \log_2 k)$ خواهد بود.



شکل ۶۸-۵ درخت انتخابی.



شکل ۶۹-۵ درخت انتخابی.

کاربردهای درخت انتخابی

از مهم‌ترین کاربردهای درخت‌های انتخابی می‌توان به کاربرد آن در مسابقات، بازی‌ها و مرتب‌سازی اشاره نمود. در مسابقات و بازی‌ها می‌توان از درخت انتخابی بدین صورت استفاده کرد که برنده‌ی هر مسابقه به مرحله‌ی بعدی صعود نماید. در انتها نیز برنده کل مسابقه در ریشه قرار خواهد گرفت.

یکی از روش‌های حل مسائل تقسیم و غلبه است. در روش تقسیم و غلبه، مسئله‌ای بزرگ و پیچیده را به مسائل کوچک تقسیم می‌کنند و هر کدام از مسائل کوچک را به راحتی حل می‌نماییم. در انتها نیز نتایج را

ادغام کرده تا به جواب نهایی برسیم. برای مرتب‌سازی یک لیست بسیار طولانی نیز یک روش آن است که این لیست را به آرایه‌های کوچک تقسیم نموده و هر کدام از آرایه‌ها را جداگانه مرتب نماییم. در انتها نیز با استفاده از درخت انتخابی آرایه‌های مرتب شده را با یک دیگر ادغام نموده و به یک لیست مرتب شده برسیم. مرتب-سازی ادغامی را در فصل هفتم بررسی خواهیم کرد.

۶– ۵. تست‌های ارشد درخت

۱. چند درخت دودویی متفاوت با ۵ گروه وجود دارد؟ (علوم کامپیوتر ـ دولتی ۸۱).

الف: ۴۲ ب: ۱۵ ج: ۷۰ د: ۲۸

۲. یک درخت دودویی کامل به ارتفاع h چند گره دارد؟ (مهندسی کامپیوتر ـ دولتی ۸٤).

الف: 2^h گره ب: 2^{h+1} گره ج: بین 2^h و 2^{h+1} گره د: بین $1 + 2^{h-1}$ و 2^h گره

۳. کدام گزینه نادرست است؟ (مهندسی کامپیوتر - دولتی ۸۲).

الف: تنها یک درخت دودویی کامل با n گره می‌توان رسم کرد.

ب: ارتفاع درخت کامل دودویی که n برگ دارد برابر است با $\lceil \log_2 n \rceil$

ج: اگر یک درخت کلی[۱۰] n برگ داشته باشد، تعداد کل گره‌های آن 2n-1 است.

د: اگر یک درخت دودویی پر n گره غیر برگ داشته باشد، تعداد کل گره‌های آن 2n+1 است.

٤. دو پیمایش Preorder و Postorder از یک درخت دودویی با N گره در دسترس است. کدام گزینه صحیح است؟ (مهندسی کامپیوتر ـ دولتی ۷٦).

الف: برگ‌ها و گره‌های تک فرزندی و ریشه را می‌توان تعیین کرد و تعداد درخت‌های دودویی به تعداد کل گره‌های پیمایش شده بستگی دارد.

ب: نمی‌توان درختی از روی این دو پیمایش ساخت و فقط ریشه را می‌توان تعیین کرد.

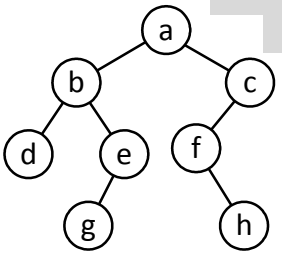
ج: برگ‌ها و گره‌های تک فرزندی و ریشه را می‌توان تعیین کرد و تعداد درخت‌های دودویی که می‌توان ساخت به تعداد گره‌های تک فرزندی بستگی دارد.

د: فقط ریشه را می‌توان تعیین کرد و نمی‌توان برگ‌ها و گره‌های تک فرزندی را تعیین کرد.

۵. الگوریتم پیمایش درخت در زمان O(d) اجرا می‌شود، d چه می‌باشد؟ (علوم کامپیوتر ـ دولتی ۸۲).

الف: تعداد برگ‌های درخت ب: عمق درخت ج: درجه درخت د: تعداد گره‌های درخت

٦. کدام گزینه پیمایش Preorder درخت زیر را مشخص می‌کند؟ (علوم کامپیوتر ـ دولتی ۸۰).



الف:abcdefgh

ب:abgeafhc

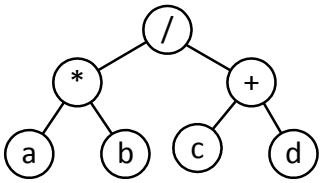
ج:abdegcfh

د:dgebhfca

---

[۱۰]. منظور از درخت کلی، درخت پر است.

۷. درخت یک عبارت جبری در شکل مقابل داده شده است. نمایش لهستانی (Polish notation) این عبارت کدام است؟ (مهندسی کامپیوتر - آزاد ۸۰).



الف: /+*abcd

ب: +*/abcd

ج: /*+abcd

د: /*ab+cd

۸. شرط اصلی برای این که بتوان از روی دو پیمایش داده شده یک درخت باینری، درخت اصلی را ساخت چیست؟ (علوم کامپیوتر ـ دولتی ۸۴).

الف: تمام داده‌ها در درخت متمایز باشند. ب: فقط ریشه از بقیه متمایز باشد.

ج: داده‌های زیر درخت ریشه از داده‌های زیر درخت راست متمایز باشند. ولی، در خود زیر درخت‌های چپ و راست می‌تواند داده‌های تکراری وجود داشته باشد.

د: داده‌های موجود در برگ‌های درخت متمایز باشند.

۹. در یک درخت باینری دلخواه پیچیدگی زمانی سه پیمایش Postorder، Preorder و Inorder به ترتیب برابر کدام است؟ (علوم کامپیوتر ـ دولتی ۸۲).

الف: O(n),O(logn),O(n) ب: $O(n^2)$,O(n),$O(n^2)$ ج: O(n),O(n),O(n) د: O(n),O(logn),$O(n^2)$

۱۰. الگوریتم زیر را در نظر بگیرید:
پیچیدگی الگوریتم برای درخت باینری
x با n گره برابر است با: (علوم کامپیوتر ـ دولتی ۸۲).

```
P(!x) {
  if (x) return(0);
  else return (1+P(left(x))+P(right(x)));
}
```

الف: $O(n^2)$ ب: O(logn)

ج: O(nlogn) د: O(n)

۷-۵. پاسخ تشریحی تست‌های ارشد درخت

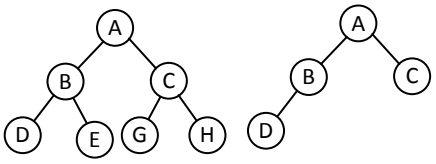
۱. گزینه (الف) صحیح است. چون تعداد درخت دودویی متفاوت با n گره برابر با $\frac{1}{n+1}\binom{2n}{n}$ است. یعنی:

$$n=5 \rightarrow \frac{1}{6}\binom{10}{5}=\frac{1\times10\times9\times8\times7\times6\times5}{6\times5\times4\times3\times2}=42$$

۲. گزینه (د) صحیح است. اگر h=3 در نظر گرفته شود، داریم:

حداقل تعداد گره‌ها در درخت دودویی کامل به ارتفاع ۳: $2^{3-1}+1=2^2+1=3$

حداکثر تعداد گره‌ها در درخت دودویی کامل به ارتفاع ۳: $2^3=2^3=8$



پس یک درخت دودویی کامل به ارتفاع h بین $2^{h-1}+1$ و 2^h گره دارد.

۳. گزینه (ب) صحیح است. چون درخت کامل با n گره $\lfloor \log_2 n \rfloor + 1$ عمق و ارتفاع دارد.

۴. گزینه (ج) صحیح است. در پیمایش Preorder، ریشه اولین گره است. اما در پیمایش Postorder ریشه در آخرین گره است. بنابراین می‌توان درخت‌های راست و چپ و محل گره‌های دو فرزندی را تعیین کرد. علاوه بر آن می‌توان فرزندان آن‌ها را نیز تعیین نمود. اما برای گره‌های که یک فرزند دارند، نمی‌توان تعیین کرد این فرزند در سمت چپ یا راست قرار می‌گیرد. بنابراین، با دو پیمایش Preorder و Postorder و n گره یک فرزندی 2^n درخت متفاوت می‌توان ایجاد کرد.

۵. گزینه (د) صحیح است.

۶. گزینه (ج) صحیح است. زیرا، در Preorder، ابتدا گره ریشه، سپس گره سمت چپ و گره سمت راست ملاقات می‌شود.

۷. گزینه (د) صحیح است. چون نمایش لهستانی همان پیشوندی (Peorder) است. طبق پاسخ سوال۶ ابتدا، گره ریشه، سپس گره سمت چپ و در پایان گره سمت راست پیمایش خواهد شد.

۸. گزینه (الف) صحیح است.

۹. گزینه (د) صحیح است.

۱۰. گزینه (د) صحیح است. چون، این تابع، بازگشتی است و برای تمام گره‌های درخت دودویی x، تابع P خودش را فراخوانی می‌کند. سپس تعداد فراخوانی‌های P برابر تعداد گره‌های همان n است.

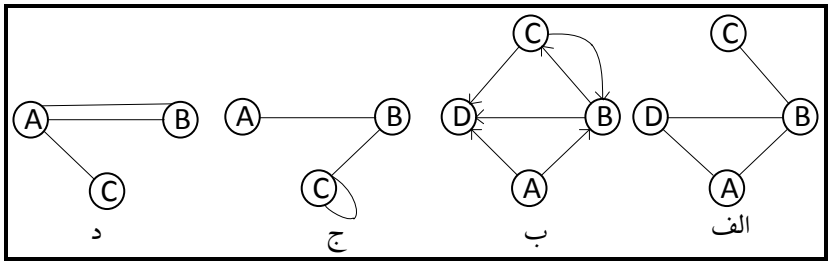
فصل ۶ گراف

در این فصل ساختمان داده‌ی گراف[11] را بررسی خواهیم کرد. گراف ساختمان داده‌ای غیر خطی و پویا است. درخت‌ها حالت خاصی از گراف‌ها هستند. در درخت، دور وجود ندارد. یعنی، از هر راس تنها از یک مسیر می‌توان به هر راس دیگر رسید. در صورتی که گراف می‌تواند دور داشته باشد. به عبارت دیگر، هر درختی یک گراف است. ولی، هر گرافی یک درخت نیست. گراف‌ها در مدل‌سازی شبکه‌های کامپیوتری و دیگر شبکه‌هایی که دارای مسیرهای مختلفی از یک راس به راس دیگر هستند،کاربرد دارند. برای شناخت و استفاده از گراف، ابتدا تعاریف و اصطلاحات مربوط به گراف را بررسی می‌نماییم.

۱-۶. تعاریف

☒ **گراف**: گراف شامل دو مجموعه V و E است که در آن V، مجموعه‌ای از عناصر هستند که به آن‌ها راس[2] گویند و E نیز مجموعه‌ای از یال‌ها[3] یا لبه‌ها است. هر یال دو راس را به یک دیگر وصل می‌نماید. یک یال به صورت $e(V_0, V_1)$ نشان داده می‌شود.

گراف‌ها به دو دسته‌ی گراف بدون جهت[4] و گراف جهت دار[5] تقسیم بندی می‌شوند. در گراف جهت‌دار یال‌ها جهت دارند. ولی، در گراف بدون جهت یال‌ها جهت ندارند. گراف‌های بدون جهت به نام گراف معروف‌اند. از این پس منظورمان از گراف، گراف بدون جهت است. در گراف یال (V_0, V_1) با یال (V_1, V_0) یکسان است، در صورتی که در گراف‌های جهت‌دار منظور از <V_0, V_1> یالی از V_0 به سمت V_1 است که با <V_1, V_0> متفاوت است. در شکل ۱-۶ الف یک گراف با مجموعه رئوس $V = \{A, B, C, D\}$ و مجموعه یال‌های $E = \{(A,B),(A,D),(D,B),(B,C)\}$ نشان داده شده است. در شکل ۱-۶ ب نیز یک گراف جهت‌دار با همان مجموعه رئوس و با مجموعه یال‌های E={<A, D>, <B, D>, <C, D>, <C, B>, <B, C>, <A, B>} نشان داده شده است.



شکل ۱-۶ الف) گراف ب) گراف جهت‌دار ج) گراف با یال حلقه د) گراف چندگانه.

---

[11]. Graph [2]. Vertex [3]. Edge [4]. Undirected graph [5]. Directed graph (digraph)

☒ **یال‌های متلاقی**: به تمامی یال‌هایی که به راس V_i متصل هستند، یال‌های متلاقی راس V_i گویند. به عنوان مثال، در شکل ۱-۶ الف، راس A دو یال متلاقی دارد.

☒ **راس منبع**[12]: در گراف جهت‌دار راسی که همه یال‌های متصل به آن خروجی هستند و درجه‌ی ورودی آن صفر است را راس منبع گویند. در شکل ۱-۶ ب، راس A منبع است.

☒ **راس مقصد یا چاه**[2]: در گراف جهت‌دار، راسی که همه‌ی یال‌های متصل به آن ورودی هستند و درجه‌ی خروجی آن صفر است را راس مقصد گویند. در شکل ۱-۶ ب، راس D مقصد است.

☒ **راس های هم‌جوار**: اگر بین دو راس یک گراف، یال وجود داشته باشد، آن دو راس را هم‌جوار یا همسایه گویند. به عنوان مثال، در شکل ۱- ۶ الف راس‌های B و D هم‌جوارند. ولی A و C هم‌جوار نیستند. در گراف‌های جهت‌دار راس Vi را هم‌جوار V_j گویند، اگر یال $<V_i,V_j>$ در گراف وجود داشته باشد. در شکل ۱-۶ ب، راس A هم‌جوار راس B می‌باشد. ولی راس B هم‌جوار راس A نیست.

☒ **مسیر**[3]: دنباله‌ای از راس‌های هم‌جوار را مسیر گویند. در شکل ۱- ۶ الف، AB، BAD و BABA چند نمونه مسیر هستند. در گراف جهت‌دار مسیر در جهت پیکان یال‌ها خواهد بود. در شکل ۱-۶ ب، CBD و ABCDB دو نمونه مسیر هستند.

☒ **طول مسیر**: تعداد یال‌های موجود در مسیر را طول مسیر گویند. طول مسیر AB، BAD و BABA به ترتیب برابر ۱، ۲ و ۳ است.

☒ **مسیر بسته**: مسیری که ابتدا و انتهای آن یکسان باشد را مسیر بسته گویند. در شکل ۱-۷ الف، BAB و ABDA دو نمونه مسیر بسته هستند.

☒ **مسیر ساده**: مسیری که تمام راس‌های آن متمایز باشد را مسیر ساده گویند. در شکل ۱-۷ الف، AB و BAD دو مسیر ساده هستند. اما، BADA و BCBAD مسیر ساده نیستند. هیچ مسیر بسته‌ای ساده نیست. چون در مسیر بسته ابتدا و انتهای مسیر یکسان است.

☒ **دور یا حلقه**[4]: یک مسیر ساده که اولین و آخرین راس آن یکسان است. در شکل ۱-۶الف، BADB و DABD حلقه هستند.

☒ **یال حلقه**: یالی که ابتدا و انتهایش یک راس باشد را یال حلقه گویند. شکل ۱-۶ ج، راس C یک یال حلقه دارد.

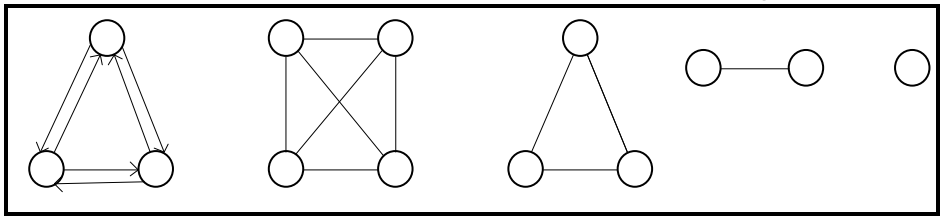
☒ **گراف چندگانه**[5]: گرافی که بین دو راس آن بیش از یک یال وجود داشته باشد را گراف چندگانه گویند. به عبارت دیگر، گراف چندگانه دارای یال‌های چندگانه است. در شکل ۱-۶ د، گراف چندگانه نشان داده شده است.

☒ **گراف ساده**[6]: گرافی که فاقد یال حلقه و یال‌های چندگانه باشد را گراف ساده گویند. شکل ۱-۶ الف، یک نمونه گراف ساده است. ولی، شکل‌های ۱-۶ ج و ۱-۶ د، گراف ساده نیستند. اغلب گراف را ساده در نظر می‌گیرند و منظور از گراف، گراف ساده است.

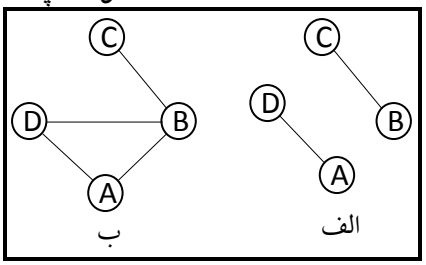
---

[12]. Source [2].Sink [3].Path [4]. Cycle [5].Multigraph [6]. simple

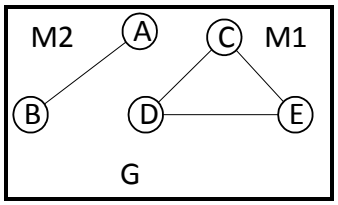
☒ **گراف کامل**: گرافی که بین هر دو راس آن یک یال وجود دارد. در شکل ۲-۶، چند نمونه گراف کامل با تعداد مختلف راس را نشان داده شده است.



شکل۲-۶ چند نمونه گراف کامل.



شکل ۳-۶ الف) گراف ناهم‌بند ب) گراف هم‌بند.



شکل ۴-۶ الف) گراف با دو مؤلفه اتصال.

نکته: تعداد یال‌های گراف کامل با n راس برابر $|E| = \frac{n(n-1)}{2}$ است. تعداد یال‌های گراف جهت‌دار کامل با n راس برابر $|E| = n(n-1)$ است. به عبارت دیگر، حداکثر تعداد یال‌های گراف و گراف جهت‌دار با n گره به ترتیب برابر $\frac{n(n-1)}{2}$ و $n(n-1)$ است.

☒ **گراف همبند**[۱۳]: گرافی که بین هر دو راس آن مسیری وجود داشته باشد را گراف هم‌بند گوییم. به گراف همبند، گراف متصل هم می‌گویند. به گرافی که هم‌بند نباشد، ناهم‌بند یا غیر هم‌بند گویند. در شکل ۳-۶ الف، یک گراف ناهم‌بند است. چون از گره C نمی‌توان به A رسید. ولی گراف شکل ۳-۶ ب، هم‌بند است.

نکته: درخت یک گراف هم‌بند بدون دور است.

نکته: گراف با n راس اگر بیش از $\binom{n-1}{2}$ یال داشته باشد، هم‌بند است.

نکته: گراف هم‌بند با n راس، حداقل n-۱ یال دارد. گراف هم‌بند با n گره و n-۱ یال، یک درخت است.

☒ **مؤلفه اتصال:** به بزرگ‌ترین زیر گراف هم‌بند مؤلفهٔ اتصال یا مؤلفه گویند. به عنوان مثال، در شکل ۴-۶، گراف G دارای دو مؤلفهٔ اتصال M1 و M2 است. مفهوم هم‌بندی در گراف‌های جهت‌دار به دو دسته‌ی هم-بندی ضعیف و هم‌بندی قوی تقسیم می‌شود.

☒ **هم‌بندی ضعیف:** در گراف جهت‌دار اگر جهت یال‌ها را در نظر نگیریم و گراف هم‌بند باشد، گوییم گراف جهت دار، هم‌بند ضعیف است. در شکل ۵-۶ الف، یک گراف جهت‌دار با هم‌بندی ضعیف نشان داده شده است.

[13]. connected

۹ – ۶. تست‌های کارشناسی ارشد گراف

۱. فرض کنید که v, u دو نود در یک گراف بدون جهت G باشند. اگر مسیر P_1, P_2 از u به v وجود داشته باشد، آنگاه (مهندسی کامپیوتر- آزاد ۷۶).

الف: v, u مجاورند. ب: G دارای سیکل است. ج: G نمی‌تواند یک گراف باشد. د: هیچ‌کدام

۲. فضای مورد نیاز برای نمایش یک گراف $G(V,E)$ به روش لیست همسایگی (Adjacency list) کدام است؟ (مهندسی کامپیوتر- سراسری ۷۸).

الف: O(|E|+|V|) ب: O(|E|) ج: O(|V|) د: O(|E|.|V|)

۳. الگوریتم زیر: (مهندسی کامپیوتر- آزاد ۷۸)

1- Initialize all nodes to the ready state (status=1)
2- Push the starting node A onto stack and change it's status to the waiting state (STATUS 2)
3- Repeat steps 4 and 5 until STACk is empty.
4- pop the top node N of STACK. Process N and change it's status to the processed state (STATUS 3)
5- push onto STACK all the neighbors of N that are still in the ready state (STATUS=1), and change thire status to the waiting state (STATUS=2) [End of step 3 loop]
6- Exit

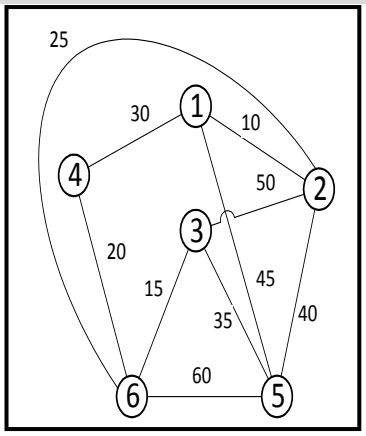
الف: الگوریتم جست‌وجوی Depth-First بر روی یک گراف است

ب: الگوریتم مرتب‌سازی Topological است.

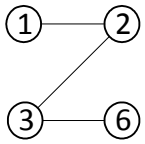
ج: الگوریتم جست‌وجوی Breadth-First است.

د: الگوریتم تبدیل یک گراف به پشته است.

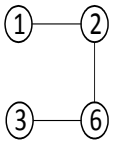
۴. اگر برای پیدا کردن درخت پوشای مینیمم گراف زیر از الگوریتم Prim استفاده کنیم. کدام یک از گزینه‌های زیر درخت حاصله در انتهای مرحله سوم این الگوریتم را به ما می‌دهد؟ (مهندسی کامپیوتر- آزاد ۷۹).



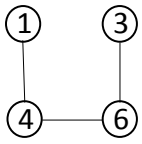
الف:



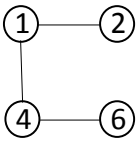
ب:



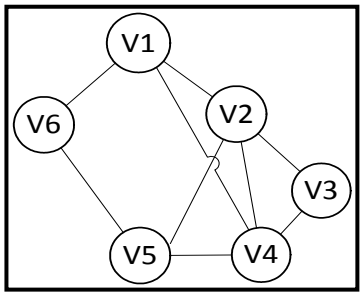
ج:



د:



۵. در گراف داده شده جست‌وجوی BFS از رأس V1 منجر می‌شود به ... (علوم کامپیوتر- سراسری ۷۹).



الف: V1, V2, V3, V4, V5, V6

ب: V1, V6, V4, V2, V5, V3

ج: V1, V6, V5, V4, V2, V3

د: V1, V6, V4, V2, V3, V5

۶. اگر رویه زیر قرار باشد، جست‌وجوی سطح اول (Breadth First Search) روی یک گراف انجام دهد، عباراتی که با شماره 1، ۲ و۳ مشخص شده‌اند چه بایستی باشند؟(مهندسی کامپیوتر- آزاد ۷۸).

```
Procedure BFS(v)
Begin
   visit V;
   1: …
    while queue is not empty do
    begin
       2: …
       for all vertices w adjacent to V
do
          if w is not visited then
          begin
             3: …
             Visit w;
          end;
    end;
end.
```

الف: 1:مورد نیاز نیست

2:AddQueue(Queue,v)

3:DeleteQueue(Queue, w)

ب: 1: AddQueue(Queue, v)

2: DeleteQueue(Queue, w)

3: AddQueue(Queue, w)

ج: 1: AddQueue(Queue, v)

2: AddQueue(Queue, v)

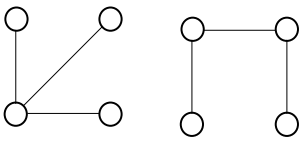
3: DeleteQueue(Queue, v)

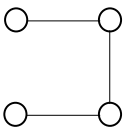
د: 1: AddQueue(Queue, v)

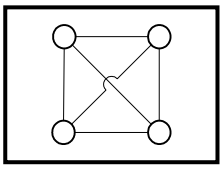
2: AddQueue(Queue, v)

3: DeleteQueue(Queue, w)

۷. گراف زیر داده شده است. کدام یک از انتخاب‌های زیر Spanning Tree این گراف است؟ (مهندسی کامپیوتر- آزاد ۷۷).

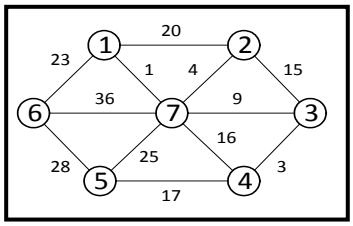
الف: 

ب: 

ج: 

د: هر سه مورد

۸. هزینه درخت پوشای مینیمم (Minimum Spanning Tree) گراف زیر چیست؟ (مهندسی کامپیوتر- آزاد ۷۷).



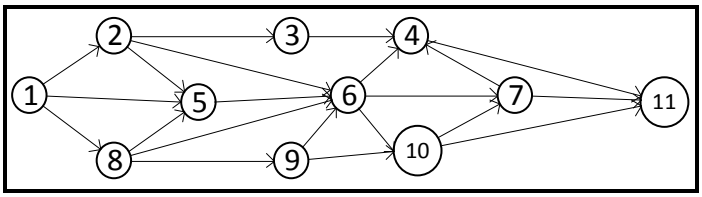
الف: ۶۸

ب: ۴۱

ج: ۸۱

د: ۵۷

۹. در جست‌وجوی اول عمق (Depth-First-Search) گراف جهت‌دار زیر، فرض کنید که گره ۱، گره شروع باشد و گره‌های مجاور یک گره به ترتیب مقدار عددی آن ملاقات شوند. کدام گزینه (از چپ به راست) ترتیب گره‌های ملاقات شده را نشان می‌دهد؟ (مهندسی کامپیوتر- دولتی ۸۱).



الف: ۹ و۸ , ۱۰ , ۷ , ۶ , ۵ , ۱۱ , ۴ , ۳ , ۲ , ۱

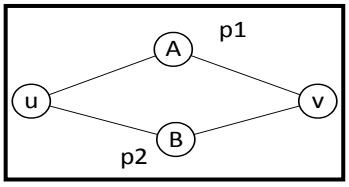
ب: ۷ و۱۰ , ۱۱ , ۴ , ۶ , ۹ , ۳ , ۵ , ۸ , ۲ , ۱

ج:۱۱ و۷ , ۱۰ , ۴ , ۶ , ۹ , ۳ , ۸ , ۵ , ۲ , ۱

د: ۱۱ و۱۰ , ۹ , ۸ , ۷ , ۶ , ۵ , ۴ , ۳ , ۲ , ۱

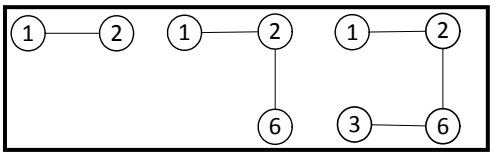
۱۰ – ۶. پاسخ تشریحی تست‌های ارشد گراف

۱. گزینه (ب) صحیح است. مانند شکل رو به رو



۲. گزینه (الف) صحیح است. برای نمایش لیست مجاورتی، برداری به تعداد گره‌های گراف یعنی $(|V|)$و هر خانه‌ی آن اشاره‌گری به ابتدای یک لیست پیوندی دارد. هر لیست پیوندی نیز به تعداد یال‌هایی که به این گره متصل است، گره دارد. یعنی $(2|E|)$ گره لیست پیوندی خواهیم داشت. پس، در مجموع تعداد عناصر حافظه از مرتبه‌ی $O(|V|+|E|)$ خواهد بود.

۳. گزینه (ب) صحیح است. این الگوریتم در قسمت مرتب‌سازی توپولوژیکی توضیح داده شده است.

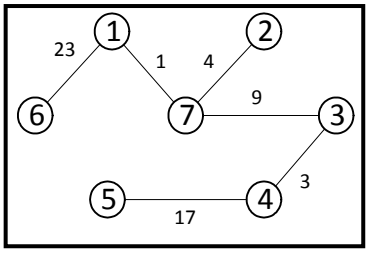


۴. گزینه(الف) صحیح است.

۵. گزینه (ب) صحیح است. ابتدا v1 به عنوان رأس شروع ملاقات می‌شود. سپس همسایه‌های v1 یعنی v2 و v4 و v6 ملاقات می‌شوند. به همین ترتیب همسایه ملاقات نشده‌ی v6 یعنی v5 دیده می‌شود. سپس، همسایه‌ی ملاقات نشده‌ی v4 یعنی v3 دیده می‌شود.

۶. گزینه (ب) صحیح است. ابتدا گره مورد نظر به انتهای صف اضافه می‌شود. سپس تا زمانی که صف خالی نشده باشد، عنصر ابتدای صف را برداشته و تمام همسایه‌های آن را که تاکنون ملاقات نشده باشند به ته صف می‌افزاییم.

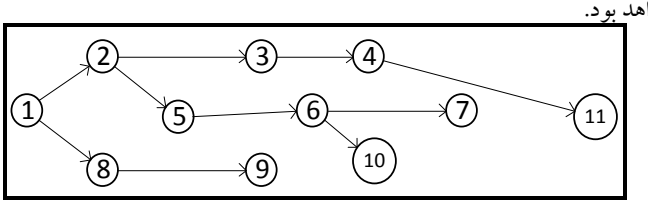
۷. گزینه (د) صحیح است. هر سه مورد از درختان پوشای گراف مورد نظر است.



۸. گزینه (د) صحیح است. درخت پوشای حداقل گراف مورد نظر به صورت زیر است و مجموع وزن یال‌ها برابر ۵۷ خواهد بود.

1+3+4+9+17+23=57

۹. گزینه (الف) صحیح است. شکل پیمایشی عمقی گراف به صورت زیر خواهد بود.



۱۰. گزینه (د) صحیح است. در متن درس توضیح داده شده است.

فصل ۷

سازی مرتب

پترین عملیاتی که بر روی لیست داده‌ها صورت می‌گیرد و کاربرد بسیار زیادی در اکثر برنامه‌های رایانه‌ای دارد، عمل مرتب‌سازی است. منظور از مرتب‌سازی، تغییر مجدد آرایش داده با یک ترتیب مشخص به صورت صعودی یا نزولی می‌باشد. داده‌ها می‌توانند از نوع عددی صحیح، عددی اعشاری، حرفی، رشته‌ای و یا رکورد باشند. داده‌های عددی به ترتیب اعداد و ارقام، داده‌های حرفی و رشته‌ای بر اساس ترتیب حروف الفبا قابل مرتب‌سازی‌اند. امّا لیستی که شامل چندین رکورد باشد، مرتب‌سازی بر روی یک یا چند فیلد رکورد انجام می‌شود. به این فیلد یا فیلدها که مرتب‌سازی بر اساس آن‌ها انجام می‌شود **کلید** گفته می‌شود. با توجه به این که در تمامی الگوریتم‌های مرتب‌سازی دو عمل مقایسه و تعویض نقش اصلی را ایفا می‌کنند. مهم‌ترین معیارهای آن سرعت و حافظه مصرفی است. امّا مسائل و مفاهیم دیگری نیز مطرح می‌شوند که می‌توان الگوریتم‌های مرتب‌سازی را بر اساس آن‌ها بررسی نمود. در این فصل ابتدا، این مسائل و مفاهیم را معرفی کرده، سپس بر اساس این مفاهیم با انواع الگوریتم‌های مرتب‌سازی آشنا می‌شویم.

۱– ۷. مفاهیم مهم در الگوریتم‌های مرتب‌سازی

در این بخش نگاهی به مسائلی که در هنگام مرتب‌سازی با آن مواجه هستیم، می‌کنیم.

☒ عناصری که مرتب می‌شوند در کدام حافظه قرار دارند؟

۱. **حافظه اصلی:** تمام عناصری که مرتب می‌شوند، اگر در حافظه اصلی قرار داشته باشند، مقایسه کلید مهم‌ترین عامل در سرعت الگوریتم است. الگوریتم‌هایی که از این روش استفاده می‌کنند، **داخلی** (Internal) نام دارند.

۲. **حافظه جانبی:** تمام عناصری که مرتب می‌شوند، در حافظه اصلی نبوده، بلکه بخشی از آن‌ها بر روی حافظه جانبی قرار دارد. علاوه بر مقایسه کلید، زمان دسترسی به عناصر مهم‌ترین عامل در سرعت الگوریتم است. الگوریتم‌هایی که از این روش استفاده می‌کنند، **خارجی** (External) نامیده می‌شوند.

☒ در الگوریتم مرتب‌سازی،آیا علاوه بر فضای حافظه‌ای که برای ذخیره‌سازی داده مورد نیاز است، از فضای حافظه کمکی نیز استفاده می‌شود؟

۱. بله، اگر الگوریتم مرتب‌سازی از فضای حافظه کمکی استفاده کند و وابسته به تعداد عناصر ورودی باشد، روش مرتب‌سازی را **غیر درجا** (Outplace) می‌گویند.

۲. خیر، اگر الگوریتم مرتب‌سازی از فضای حافظه کمکی استفاده نکند و یا وابسته به تعداد عناصر ورودی نباشد، روش مرتب‌سازی را **درجا** (Inplace) می‌گویند.

☒ آیا الگوریتم مرتب‌سازی متعادل یا پایدار (Stable) است؟

۱. الگوریتم مرتب‌سازی **پایدار** است که ترتیب عناصر با کلید مساوی را حفظ کند. یعنی، اگر داده‌های ورودی نامرتب به صورت $5, 3_{(3)}, 3_{(2)}, 4, 3_{(1)}, 7$ باشد(شماره‌های پایین عدد۳ ترتیب ورودی را نشان می‌دهد). بعد از مرتب‌سازی داده‌ها، لیست داده به صورت $3_{(3)}, 4, 5, 7, 3_{(2)}, 3_{(1)}$ مرتب شود، مرتب‌سازی پایدار است و اگر به صورت $3_{(3)}, 4, 5, 7, 3_{(2)}, 3_{(1)}$ مرتب شود، مرتب‌سازی ناپایدار می‌باشد.

☒ الگوریتم‌های مرتب‌سازی از چه روش‌هایی برای مرتب کردن استفاده می‌کنند؟

۱. **مرتب‌سازی تعویضی (Exchange)**، در این روش پس از مقایسه دو عنصر در صورتی که ترتیب آن‌ها مناسب نباشد، جای آن‌ها تعویض می‌گردد. همواره تعداد تعویض‌ها کوچک‌تر یا مساوی تعداد مقایسه‌ها است.

۲. **مرتب‌سازی انتخابی(Selection)**، در این روش در هر مرحله، عنصری از لیست انتخاب شده، در صورتی که ترتیب آن مناسب نباشد، در جای مناسب خود قرار می‌گیرد.

۳. **مرتب‌سازی درجی (Insertion)**، از روش درج عناصر در موقعیت مناسب استفاده می‌کند.

☒ پیچیدگی زمانی الگوریتم‌های مرتب‌سازی بر حسب n ورودی چیست؟

۱. $O(n^2)$: در این الگوریتم از دو حلقه تکرار استفاده می‌شود و کارایی پایین‌تری دارند.

۲. $O(nLogn)$: در این الگوریتم از روش تقسیم و غلبه استفاده می‌شود. اغلب کارایی مناسب‌تری دارند. بنابراین، بهینه‌ترین زمان برای الگوریتم‌های مرتب‌سازی مرتبه زمانی $O(nLogn)$ است.

☒ پیچیدگی حافظه، الگوریتم‌های مرتب‌سازی بر حسب n ورودی چیست؟

۱. در حالت بهینه فضای حافظه باید متناسب با اندازه لیست باشد. یعنی، پیچیدگی حافظه برای یک لیست n عنصری برابر با $O(n)$ باشد.

۲– ۷. الگوریتم‌های مرتب‌سازی

با توجه به موارد اشاره شده، به طور مطلق نمی‌توان گفت کدام الگوریتم مرتب‌سازی بهترین است. بلکه، بستگی به شرایط مسئله دارد. در ادامه، انواع الگوریتم‌های مرتب‌سازی داخلی را برای یک آرایه n عنصری که در حافظه اصلی قرار دارند، مطرح نموده و پیچیدگی زمانی آن‌ها را با یک دیگر مقایسه می‌نماییم.

۱–۲– ۷. مرتب‌سازی حبابی

یکی از ساده‌ترین و ابتدایی‌ترین روش‌های مرتب‌سازی، روش حبابی است. در این روش برای مرتب کردن یک لیست یا همان آرایه n عنصری به صورت صعودی نیاز به چندین گذر یا مرحله است که با عملیات زیر انجام می‌شود:

در گذر اول، اولین مقدار با دومین مقدار آرایه مقایسه می‌شود. اگر ترتیب آن‌ها درست نبود، جای آن‌ها با یک دیگر عوض می‌گردد و سپس هر عنصر با عنصر بعدی مقایسه می‌شود. این کار تا انتهای لیست ادامه می‌یابد. در پایان گذر اول، بزرگ‌ترین عنصر در انتهای لیست قرار می‌گیرد. در گذرهای بعدی نیز همین کار را از ابتدای آرایه مجدداً تکرار کرده، با این تفاوت که هر بار لیست از انتها کوچک‌تر می‌شود. در این روش حداکثر به n-1 گذر و در هر گذر نیز نیاز به n-pass مقایسه است(pass، همان شماره گذر است). الگوریتم مرتب‌سازی حبابی در زیر آمده است:

مثال ۱-۷. لیست A={9, 7, 5, 1} شامل ۵ عنصر که به صورت نزولی مرتب شده است. با استفاده از مرتب‌سازی حبابی آن را به صورت صعودی مرتب نمایید.

```
void bubbleSort (int A [ ] , int n)  {
 int pass,j,temp;
 for (pass=1; pass<=n-1 ; pass ++)  {
  for (j=0; j<n-pass ; j++)
     if (A[j]>A[j+1])     {
         temp=A[j];
         A[j]=A[j+1];
         A[j+1]=temp;
       }
  }
}
```

برای درک بهتر مرتب‌سازی حبابی لیست به صورت نزولی که در بدترین وضعیت است، در نظر گرفته شده است تا تمام حالت‌های آن بررسی شود.

در گذر اول: تعداد عناصر۴، تعداد مقایسه ۳

آرایه A با موقعیت عناصر آن در لیست	A[3]	A[2]	A[1]	A[0]
داده‌های اولیه در آرایه A	1	5	7	9
مقایسه ۷ و ۹، تعویض آن‌ها	1	5	7	9
مقایسه ۵ و ۹، تعویض آن‌ها	1	5	9	7
مقایسه ۱ و۹، تعویض آن‌ها	1	9	5	7
۹ بزرگ‌ترین عنصر لیست به انتهای لیست منتقل شد	9	1	5	7

در گذر دوم: تعداد عناصر۴، تعداد مقایسه ۲

آرایه A با موقعیت عناصر آن در لیست	A[3]	A[2]	A[1]	A[0]
داده‌های لیست در پایان گذر اول	9	1	5	7
مقایسه ۷ و۵، تعویض آن‌ها	9	1	5	7
مقایسه ۱ و۷، تعویض آن‌ها	9	1	7	5
مقایسه ۷ و۹ لازم نیست. چون در گذر اول مقایسه شدند.	9	7	1	5

در گذر سوم: تعداد عناصر۴، تعداد مقایسه ۱

آرایه A با موقعیت عناصر آن در لیست	A[3]	A[2]	A[1]	A[0]
داده‌های لیست در پایان گذر اول	9	7	1	5
مقایسه ۱ و ۵، تعویض آن‌ها	9	7	1	5
لیست به صورت صعودی مرتب شده است	9	7	5	1

سوال۱: چه رابطه‌ای بین عناصر لیست و تعداد گذرها وجود دارد؟

اگر لیست شامل n عنصر باشد، آنگاه تعداد گذرها برابر با n-1 خواهد بود. یعنی، لیست ۴ عنصری ۳ (۱ – ۴) گذر رخ داد.

سوال۲: چه رابطه‌ای بین عناصر لیست و تعداد مقایسه در هر گذر وجود دارد؟

اگر لیست شامل n عنصر باشد، آنگاه تعداد مقایسه در هر گذر برابر با n-pass است (pass، شماره گذر است). به عنوان مثال، در لیست ۴ عنصری در گذر۳ تعداد ۱ (۳-۴) مقایسه انجام می‌شود.

سوال۳: چه رابطه‌ای بین تعداد عناصر لیست و تعداد مقایسه در کل گذرها وجود دارد؟

چون در گذر اول n-1 مقایسه، در گذر دوم، n-2 مقایسه و در گذر (n-1)ام، ۱ مقایسه انجام می‌شود. پس مجموع کل مقایسه‌ها را به صورت زیر می‌توان نوشت:

حداکثر تعداد مقایسه مرتب‌سازی حبابی $= \frac{n(n-1)}{2}$

به عنوان مثال، در لیست ۴ عنصری حداکثر تعداد مقایسه‌ها برابر با ۶=۱+۲+۳ است که از رابطه فوق نیز بدست می‌آید:

حداکثر تعداد مقایسه مرتب‌سازی حبابی $= \frac{n(n-1)}{2} = \frac{4(4-1)}{2} = 6$

خلاصه عملکرد مرتب‌سازی حبابی در جدول ۷-۱ آمده است.

جدول ۷-۱ خلاصه عملکرد مرتب‌سازی حبابی.	
روش مرتب‌سازی	تعویضی (Exchange)
وضعیت استفاده از فضای حافظه کمکی	درجا (In place)
حالت پایداری	متعادل (پایدار)
مرتبه زمانی	$O(n^2)$
بهترین حالت	زمانی که لیست به صورت صعودی مرتب باشد.
بدترین حالت	زمانی که لیست به صورت نزولی مرتب باشد.
تعداد مقایسه در بهترین و بدترین حالت	تفاوت دارد

بهبود الگوریتم مرتب‌سازی حبابی

یکی از مشکلاتی که الگوریتم مرتب‌سازی حبابی ارائه شده دارد، این است که اگر لیست در طی یک یا چند گذر مرتب شود، آنگاه اجرای الگوریتم تا خاتمه گذر (n-1) ادامه می‌یابد. برای حل این مشکل می‌توان متغیر منطقی flag با مقدار اولیه صفر را در نظر گرفت که اگر در طی یک گذر عمل جا به جایی صورت گیرد، مقدار آن ۱ می‌شود و اگر عمل جا به جایی صورت نگیرد، مقدار آن صفر باقی می‌ماند. پس، اگر در گذری flag برابر با صفر باقی بماند، اجرای الگوریتم خاتمه می‌یابد (الگوریتم مقابل):

```
void bubbleSort (int A [ ] , int n) {
  int pass, j, temp, flag=1;
  for (pass=1; (pass<=n-1) && flag ; pass+ +) {
      f lag=0;
     for (j=0; j<n-pass ; j++)
     if (A[j]>A[j+1]) {
      temp=A[j];
      A[j]=A[j+1];
      A[j+1]=temp;
      flag=1;
     }
  }
}
```

	بهترین حالت	حالت متوسط	بدترین حالت
تعداد جا به جایی	صفر	$\frac{n(n-1)}{4}$	$\frac{n(n-1)}{2}$
تعداد مقایسه	n-1	$\frac{(n-1)(n-2)}{2}$	$\frac{n(n-1)}{2}$
مرتبه	$O(n)$	$O(n^2)$	$O(n^2)$

جدول ۲ - ۷ پیچیدگی زمانی الگوریتم مرتب سازی حبابی $O(n^2)$ است. چون دو حلقه for دارد.

۷-۲-۲ مرتب‌سازی انتخابی

در این روش در هر مرحله، عنصری از لیست انتخاب شده در صورتی که ترتیب آن مناسب نباشد، در جای مناسب خود قرار می‌گیرد. الگوریتم روش مرتب‌سازی انتخابی (Selection Sort) به شرح زیر است:

۱. از ابتدای لیست تا انتهای لیست را جست‌وجو کرده، مکان کوچک‌ترین عدد را پیدا می‌کنیم.

۲. کوچک‌ترین عدد لیست را با اولین عدد لیست جا به جا کرده تا کوچک‌ترین عدد به ابتدای لیست برود.

۳. مراحل ۱ و۲ را تکرار می‌کنیم. با این تفاوت که هر بار به اندیس ابتدای لیست یک واحد اضافه می‌کنیم تا به انتهای لیست نزدیک‌تر شویم. این الگوریتم در رو به رو آمده است:

```
void selectionSort (int A[ ] , int n) {
int pass, j, minPost , temp;
for (pass=0;  pass<n-1;pass++)   {
  minPos =pass;
 for (j=pass+1; j<n; j++)
     if (A[j]<A[minPos])  minPos=j;
   temp=A[minPos];
   A[minPos]=A[pass];
   A[pass]=temp;
   }
}
```

پیچیدگی زمانی الگوریتم مرتب‌سازی انتخابی $0(n^2)$ است. چون دو حلقه for دارد.

خلاصه عملکرد مرتب‌سازی انتخابی در جدول ۳ - ۷ و پیچیدگی زمانی آن ۴ – ۷ آمده است.

باید در نظر داشت که به جای کوچک‌ترین عنصر می‌توان بزرگ‌ترین عنصر را پیدا کرد. ولی آن را باید در آخر لیست قرار داد تا مرتب‌سازی به صورت صعودی شود. اگر در ابتدای لیست قرار گیرد، مرتب‌سازی به صورت نزولی می‌شود.

جدول ۳ – ۷ خلاصه عملکرد مرتب سازی انتخابی.

روش مرتب‌سازی	انتخابی
وضعیت استفاده از فضای حافظه کمکی	درجا
حالت پایداری	ناپایدار
مرتبه زمانی	$O(n^2)$
بهترین حالت	زمانی که لیست به صورت صعودی مرتب باشد.
بدترین حالت	زمانی که لیست به صورت نزولی مرتب باشد.
تعداد مقایسه در بهترین و بدترین حالت	تفاوتی وجود ندارد

جدول ۴ – ۷ پیچیدگی زمانی الگوریتم مرتب‌سازی انتخابی است. $0(n^2)$

	بهترین حالت	حالت متوسط	بدترین حالت
تعداد جا به جایی	صفر	$\frac{n}{2}$	n-1
تعداد مقایسه	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$
مرتبه	$0(n^2)$	$0(n^2)$	$0(n^2)$

مثال ۲-۷. لیست A={4,6,7,1,5} شامل ۵ عنصر است. با استفاده از مرتب‌سازی انتخابی آن را به صورت صعودی مرتب نمایید.

در گذر اول: ابتدای لیست خانه صفر در نظر گرفته می‌شود. پس اولین عنصر آن که (۴) با کوچک‌ترین عنصر لیست (۱) جا به جا می‌شود. یعنی، جا به جایی بین A[0] و A[3] صورت می‌گیرد.

آرایه A با موقعیت عناصر آن در لیست	A[4]	A[3]	A[2]	A[1]	A[0]
داده‌های اولیه در آرایه A از اندیس ۰ تا ٤	5	1	7	6	4
اولین عنصر ٤،کوچک‌ترین عنصر ۱	5	1	7	6	4
تعویض مقادیر مکان ۰ و ۳ (۱ و ٤)	5	4	7	6	1

در گذر دوم: ابتدای لیست خانه یک در نظر گرفته می‌شود. پس اولین عنصر لیست جدید عدد (۶) با کوچک‌ترین عنصر لیست جدید(۴) جا به جا می‌شود. یعنی، جا به جایی بین A[1] و A[3] صورت می گیرد.

آرایه A با موقعیت عناصر آن در لیست	A[4]	A[3]	A[2]	A[1]	A[0]
داده‌های اولیه در آرایه A از اندیس ۱ تا ٤	5	4	7	6	1
اولین عنصر ٦،کوچک‌ترین عنصر ٤	5	4	7	6	1
تعویض مقادیر مکان ۱ و ۳ (٦ و٤)	5	6	7	4	1

در گذر سوم: ابتدای لیست خانه ۲ در نظر گرفته می‌شود. پس اولین عنصر لیست جدید عدد(۷) با کوچک-ترین عنصر لیست جدید یعنی ۵ جا به جا می‌شود. یعنی، جا به جایی بین A[2] و A[4] صورت می گیرد.

آرایه A با موقعیت عناصر آن در لیست	A[4]	A[3]	A[2]	A[1]	A[0]
داده‌های در آرایه A از اندیس ۲ تا ٤	5	6	7	4	1
اولین عنصر۷،کوچک‌ترین عنصر۵	5	6	7	4	1
تعویض مقادیر مکان ۲ و ٤ (۵ و۷)	7	6	5	4	1

در گذر چهارم: لیست جدید فقط شامل خانه ۳ و۴ می‌باشد. بنابراین، ابتدای لیست خانه ۳ در نظر گرفته می-شود. پس اولین عنصر لیست جدید عدد ۶ می‌باشد با کوچک‌ترین عنصر لیست جدید باید جا به جا شود، چون وجود ندارد، جا به جایی صورت نمی گیرد.

آرایه A با موقعیت عناصر آن در لیست	A[4]	A[3]	A[2]	A[1]	A[0]
داده‌ها در آرایه A از اندیس۳ تا ٤	7	6	5	4	1
لیست به صورت صعودی مرتب شده است	7	6	5	4	1

سوال۱: چه رابطه‌ای بین عناصر لیست و تعداد گذرها در مرتب‌سازی انتخابی وجود دارد؟

اگر لیست شامل n عنصر باشد، آنگاه تعداد گذرها برابر با n-1 می‌باشد. یعنی، لیست ۵ عنصری ۴ (۱-۵) دارد.

سوال۲: چه رابطه‌ای بین عناصر لیست و تعداد مقایسه در هر گذر وجود دارد؟

اگر لیست شامل n عنصر باشد، آنگاه تعداد مقایسه برای پیدا کردن کوچک‌ترین عنصر در هر گذر برابر با n-pass می‌باشد(pass، شماره گذر است).

سوال۳: چه رابطه‌ای بین تعداد عناصر لیست و تعداد مقایسه در کل گذرها وجود دارد؟

چون برای پیدا کردن کوچک‌ترین عنصر درگذر اول n-1 مقایسه، در گذر دوم، n-2 مقایسه و در گذر (n-1)ام، ۱ مقایسه انجام می‌شود، پس مجموع کل مقایسه‌ها را به صورت زیر می‌توان نوشت:

حداکثر تعداد مقایسه مرتب‌سازی انتخابی= $\frac{n(n-1)}{2}$ = 1+2+...+(n-1)

به عنوان مثال، در لیست ۵ عنصری حداکثر تعداد مقایسه‌ها برابر با ۶ (۱+۲+۳) است که از رابطه زیر نیز به دست می‌آید:

حداکثر تعداد مقایسه مرتب‌سازی انتخابی $10=\frac{5(5-1)}{2}=\frac{n(n-1)}{2}=$

۲-۳- ۷. مرتب سازی درجی

مرتب‌سازی درجی(Insertion Sort)، یک الگوریتم مرتب‌سازی ساده بر مبنای مقایسه است که پیاده‌سازی آن راحت است. این الگوریتم برای داده‌های زیاد، کارآمد نیست، امّا برای $n \leq 20$ سریع‌ترین روش مرتب‌سازی است. برای داده‌های با تعداد کم تقریباً مرتب شده کارآمدتر از روش انتخابی و حبابی عمل

می‌کند. در این روش، ابتدا عنصر اول را در نظر گرفته، عنصر دوم به گونه‌ای درج می‌شود تا لیست مرتب دو عنصری تشکیل گردد. سپس، عنصر سوم را در جای مناسبی قرار داده تا لیست ۳ عنصری مرتب ایجاد شود و همین عمل تا مرتب شدن کامل لیست ادامه می‌یابد. خلاصه عملکرد الگوریتم مرتب‌سازی درجی در جدول ۵ - ۷ آمده است و الگوریتم مرتب‌سازی درجی را در زیر می‌بینید:

```
void insertionSort(int * A, int n){
for (int i = 1; i< n; i++){
 int temp = A [i];
 int position = i;
 while (position > 0 && A [position-1] > temp){
    A [position] = A [position-1];
     position--;
  }
  A [position] = temp;
  }
}
```

همان طور که در این الگوریتم مشاهده می‌شود، پیچیدگی الگوریتم $O(n^2)$ می‌باشد. امّا اگر آرایه مرتب باشد، حلقه while اجرا نشده و فقط حلقه for حداکثر n-1 اجرا می‌شود. بنابراین، درجه آن $O(n)$ است (جدول ۶- ۷).

جدول ۵ - ۷ خلاصه عملکرد الگوریتم مرتب‌سازی درجی

روش مرتب‌سازی	**درجی**
وضعیت استفاده از فضای حافظه کمکی	**درجا**
حالت پایداری	**پایدار**
مرتبه زمانی	**$O(n^2)$**
بهترین حالت	**زمانی که لیست به صورت صعودی مرتب باشد.**
بدترین حالت	**زمانی که لیست به صورت نزولی مرتب باشد.**

مثال ۳-۷. لیست A={17,16,14,15,11} شامل ۵ عنصر است. با استفاده از مرتب‌سازی درجی آن را به صورت صعودی مرتب نمایید.

با توجه به الگوریتم درگذر اول i=1، temp=A[1]=16، position=1 و شرط while درست است، پس محتویات A[0] (عدد۱۷) در A[1] قرارگرفته و محتویات A[1] (۱۶) که در شروع کار در temp قرار داشت در A[0] قرار می‌گیرد و این روند تا پایان لیست ادامه دارد. مراحل کار به طور خلاصه در جدول زیر نشان داده شده است:

آرایه A با موقعیت عناصر آن در لیست	A[4]	A[3]	A[2]	A[1]	A[0]
داده‌های اولیه در آرایه A	11	15	14	16	17
گذر اول درج ۱۷ در A[1] و۱٦ در A[0]	11	15	14	17	16
گذر دوم درج ١٤ قبل از١٦	11	15	17	16	14
گذر سوم درج ١٥ قبل از١٦	11	17	16	15	14
گذر چهارم درج ١١ قبل از١٤	17	16	15	14	11

سوال۱: چه رابطه‌ای بین عناصر لیست و تعداد گذرها در مرتب‌سازی درجی وجود دارد؟

اگر لیست شامل n عنصر باشد، آنگاه تعداد گذرها برابر با n-1 می‌باشد.

سوال ۲: چه حالتی پیچیدگی زمانی الگوریتم مرتب‌سازی درجی O(n) است؟

اگر لیست تقریباً به صورت صعودی مرتب شده باشد.

سوال ۳: در چه حالتی مرتبه تعویض‌ها در الگوریتم مرتب‌سازی درجی O(1) است؟

اگر لیست در ابتدا کاملاً به صورت صعودی مرتب شده باشد. تعداد مقایسه در حلقه while برابر با O(n) می‌شود. امّا شرط حلقه while در تمامی مراحل نادرست است، هیچ‌گاه وارد حلقه while نمی‌شود تا عمل تعویض و جا به جایی در نتیجه، درجه تعویض‌ها O(1) می گردد.

	بهترین حالت	حالت متوسط	بدترین حالت
تعداد انتساب	2(n-1)	$\frac{(n-1)(n+8)}{4}$	$\frac{(n-1)(n+4)}{2}$
تعداد مقایسه	n-1	$\frac{(n-1)(n+2)}{4}$	$\frac{n(n-1)}{2}$
مرتبه	$0(n)$	$0(n^2)$	$0(n^2)$

جدول ٦ - ٧ پیچیدگی زمانی الگوریتم مرتب‌سازی درجی است. $0(n^2)$

۲-۴- ۷. مرتب‌سازی درجی Shell

این الگوریتم یکی از روش‌های مرتب‌سازی درجی است که نام این الگوریتم از نام ابداع کننده آن گرفته شده است. در این روش، در هر مرحله لیست به قسمت‌های مجزایی تقسیم شده که لیست‌های کوچک‌تری را تشکیل می‌دهند، این لیست‌های کوچک اغلب به روش درجی ساده مرتب می‌شوند. برای تقسیم لیست به لیست‌های کوچک‌تر از پارامتر افزایشی مانند k استفاده می‌شود که kامین عنصر از لیست انتخاب می‌شود. در هر مرحله مقدار k نیز بر اساس یک مجموعه اولیه کاهش می‌یابد. خلاصه عملکرد الگوریتم مرتب‌سازی درجی Shell در جدول زیر آمده است و الگوریتم مرتب‌سازی درجیShell را در زیر می‌بینید:

```
void ShellSort(int *A, int n,int *inck,int nk ){
for (int pass = 1; pass<nk; ++pass){
    k=inck[pass];
 for (int j =k; j<n;j++){
int temp = A [j];
      for (s=j-k;(s>0)
&&(temp<A[s]);s-=k)
          A[s+k]=A[s];
       A[s+k]=temp;
     }
}
```

مرتب‌سازی درجی Shell	
روش مرتب‌سازی	**درجی**
حالت پایداری	**پایدار**
مرتبه زمانی	**$O(n(\log n)^2)$**

الگوریتم ShellSort آرایه A را که شامل n عنصر است، با دنباله افزایشی که در آرایه inck با nk عنصر قرار دارد، در نظر می گیرد و به شیوه درجی ساده مجموعه‌های کوچک‌تر را مرتب می کند. در شروع کار چون مقداری که برای k در نظر گرفته می‌شود، بزرگ، امّا لیست‌های ایجاد شده کوچک است، بنابراین سرعت مرتب‌سازی لیست‌های کوچک با روش درجی ساده سریع است و همچنین در مراحل آخر اندازه k کوچک شده، اما لیستی که باید مرتب شود، بزرگ است. ولی، چون در مراحل قبلی تقریباً مرتب‌تر شده‌اند، باز هم مرتب‌سازی درجی خوب جواب می‌دهد.

محاسبه پیچیدگی این الگوریتم، به طول آرایه افزایشی و مقادیر مختلف آن بستگی دارد. اگر دنباله مناسبی از افزایش‌ها در نظر گرفته شود، دارای مرتبه زمانی $O(n(\log n)^2)$ و در غیر این صورت، $O(n\sqrt{n})$ می‌باشد که برای nهای بسیار بزرگ از $O(n^2)$ بهینه‌تر است. یکی از حالت‌های مناسب دنباله افزایشی این است که مقادیر آن نسبت به هم اول باشند. به عنوان نمونه، inck={5,3,1} که ۵، ۳ و ۱ نسبت به هم اول هستند.

مثال ۴-۷. لیست {30,28,25,27,26,29}=A شامل۶ عنصر است. با استفاده از مرتب‌سازی Shell با دنباله افزایشی {۳،۲،۱} آن را به صورت صعودی مرتب نمایید.

با توجه به دنباله افزایشی پس این لیست باید در سه گذر مرتب شود. در گذر اول k=3، در گذر دوم k=2 و در گذر سوم k برابر با ۱ در نظر گرفته شده و لیست‌ها بر اساس آن‌ها ایجاد می‌شوند.

شروع گذر اول: k=3 یعنی عناصر لیست با فاصله ۳ از یک دیگر، مقایسه شده و مرتب می‌شوند:

آرایه A با موقعیت عناصر آن در لیست	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]
داده‌های اولیه در آرایه A	29	26	27	25	28	30
لیست اوّل، ۲۷ و ۳۰ باید مرتب شوند			27			30
لیست دوم، ۲۶ و ۲۸ باید مرتب شوند		26			28	
لیست سوم، ۲۹ و ۲۵ باید مرتب شوند	29			25		
پایان گذر اوّل	29	28	30	25	26	27

شروع گذر دوم: k=2 یعنی عناصر لیست با فاصله ۲ از یک دیگر، مقایسه شده و مرتب می‌شوند:

آرایه A با موقعیت عناصر آن در لیست	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]
محتویات آرایه A در پایان گذر اوّل	29	28	30	25	26	27
لیست اول، ۲۷، ۲۵ و ۲۸ باید مرتب شوند		28		25		27
لیست دوم، ۲۶، ۳۰ و ۲۹ باید مرتب شوند	29		30		26	
پایان گذر دوم	30	28	29	27	26	25

شروع گذر سوم: با k=1 یعنی عناصر لیست با فاصله ۱ با یک دیگر مقایسه شده و مرتب می‌شوند:

آرایه A با موقعیت عناصر آن در لیست	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]
محتویات آرایه A در پایان گذر دوم	30	28	29	27	26	25
پایان مرتب‌سازی	30	29	28	27	26	25

۵-۲-۷. مرتب‌سازی جا به جایی

الگوریتمی غیر کارآمد امّا با پیاده‌سازی ساده است. در این روش در گذر اول، اولین عنصر لیست، با عنصر دوم تا آخرین عنصر لیست مقایسه می‌شود و اگر از هر عنصر بزرگ‌تر بود با آن جا به جا می‌شود. بنابراین، در انتهای گذر اول کوچک‌ترین عنصر لیست در ابتدای لیست قرار می‌گیرد. در گذر دوم، دومین عنصر لیست با عنصر سوم تا آخرین عنصر لیست مقایسه می‌شود و اگر از هر عنصر بزرگ‌تر بود با آن جا به جا می‌شود و این روند تا n-1 گذر ادامه دارد. الگوریتم مرتب‌سازی جا به جایی (Exchange Sort) در مقابل آمده است:

```
void ExchangeSort (int A[], int n) {
 for (i=0;i<n-1;i++)
for (j=i+1;j<n;j++)
        if (A[i]>A[j]){
           temp=A[i];
           A[i]=A[j];
A[j]=temp;
}
}
```

مثال ۵-۷. لیست {34,37,32,46,21}=A شامل ۵ عنصر است. با استفاده از الگوریتم مرتب‌سازی جا به جایی آن را به صورت صعودی مرتب نمایید.

گذر	i	j	توضیحات	A[4]	A[3]	A[2]	A[1]	A[0]
	0	0	عناصر نامرتب آرایه A	21	46	32	37	34
اول	0	1	A[1]>A[0] نیست جا به جایی صورت	21	46	32	37	34

			نمی‌گیرد						
	0	2	A[2]>A[0] است جا به جایی صورت می‌گیرد	21	46	34	37	32	
	0	3	A[3]>A[0] نیست جا به جایی صورت نمی‌گیرد	21	46	34	37	32	
	0	4	A[4]>A[0] است جا به جایی صورت می‌گیرد	34	46	32	37	21	
دوم	1	2	A[2]>A[1] است جا به جایی صورت می‌گیرد	34	46	37	32	21	
	1	3	A[3]>A[1] نیست جا به جایی صورت نمی‌گیرد	34	46	37	32	21	
	1	4	A[4]>A[1] نیست جا به جایی صورت نمی‌گیرد	34	46	37	32	21	
سوم	2	3	A[3]>A[2] نیست جا به جایی صورت نمی‌گیرد	34	46	37	32	21	
	2	4	A[4]>A[2] است جا به جایی صورت می‌گیرد	37	46	34	32	21	
چهارم	3	4	A[4]>A[3] است جا به جایی صورت می‌گیرد	46	37	34	32	21	

۶ – ۲– ۷. مرتب‌سازی سریع

مرتب‌سازی سریع(Quick Sort) از روش تقسیم و غلبه برای مرتب‌سازی استفاده می‌کند. در این الگوریتم در هر مرحله یکی از عناصر لیست به عنوان محور یا لولا (Pivot) انتخاب شده، سپس سایر عناصر لیست به گونه‌ای جا به جا می‌شوند که کلیه عناصر بزرگ‌تر از محور در سمت راست و تمامی عناصر کوچک‌تر از محور در سمت چپ آن قرار گیرند تا عنصر محور در مکان مناسب خود قرار گیرد. در ادامه لیستی که در سمت چپ و راست عنصر محور ایجاد شد به صورت بازگشتی به همین روش مرتب می‌شوند.

```
void quickSort(int A[], int first, int last) {
int p;
if (first < last)    {
  p= partition(A, first, last); //پیدا کردن موقعیت عنصر محور
  quicksort(A,first,p-1);    //مرتب سازی لیست سمت چپ
     quicksort(A, p+1,last);   //مرتب سازی لیست سمت راست
   }
}
```

بنابراین، برای نوشتن الگوریتم بازگشتی مرتب‌سازی سریع باید سه عمل زیر انجام شود:

۱. انتخاب عنصر محور (Pivot)، موقعیت اولیه عنصر محور را می‌توان میانه عناصر، عنصر وسط آرایه یا اولین عنصر آرایه در نظر گرفت برای کاهش هزینه معمولاً عنصر محور را اولین عنصر در نظر می‌گیرند، امّا انتخاب میانگین سه عنصر اول، وسط و آخر انتخاب مناسب‌تری است.

۲. تقسیم آرایه با استفاده از تابع partition، لیست بر مبنای عنصر محور به گونه‌ای تغییر می‌کند که تمامی عناصر کوچک‌تر یا مساوی محور در سمت چپ آن، و تمامی عناصر بزرگ‌تر در سمت راست آن قرار می‌گیرند. بنابراین، ابتدا موقعیت عنصر محور را پیدا کرده، مثلاً p، سپس لیست سمت چپ از first تا p-1 و لیست سمت راست از p+1 تا last در نظر گرفته می‌شود.

۳. مرتب‌سازی بازگشتی، لیست سمت چپ و راست به روش مرتب‌سازی سریع مجدداً مرتب می‌شوند.

این الگوریتم با دو تابع quickSort و partition به صورت زیر پیاده‌سازی می‌شود:

تابع partition که در مرتب‌سازی QuickSort فراخوانی شده است برای تقسیم کردن آرایه یا لیست به کار می‌رود. هدف اصلی آن پیدا کردن مکان مناسب عنصر محور در آرایه می‌باشد.

```
int partition(int*A, int first, int last){
  int  i= first , j = last , pivot = A[first], temp,  pos;
  while (i <j){
  while (A[j] >pivot) j--;
  while (i<j&& A[i] <= pivot) i++;
  if (i <j){
    temp=A[i];
    A[i]=A[j];
   A[j]=temp;
   }
   pos=j;
   A[first]=A[pos]
    A[pos]=pivot;
    return pos;
}
```

این الگوریتم در بهترین حالت دارای پیچیدگی زمانی O(nlogn) و در بدترین حالت دارای پیچیدگی زمانی $O(n^2)$ می‌باشد. بدترین حالت زمانی است که لیست به صورت صعودی مرتب باشد. امّا با توجه به بازگشتی بودن الگوریتم به فضای اضافی حافظه برای پشته نیاز دارد که در بهترین حالت دارای حافظه O(logn) و در بدترین حالت دارای حافظه O(n) می‌باشد. خلاصه عملکرد مرتب سازی سریع در جدول ۸ – ۷ آمده است. الگوریتم تابع partition را در زیر می‌بینید:

مثال ۶–۷. لیست A={9,18,5,3,20} شامل، ۵ عنصر است. با استفاده از الگوریتم مرتب‌سازی سریع به صورت صعودی مرتب نمایید.

با توجه به الگوریتم باید به نکات زیر برای حل مسئله توجه داشت:

۱. در پیدا کردن محل مناسب pivot باید i کوچک‌تر و مساوی j باشد تا حلقه‌های while تکرار شوند، در غیر این صورت، محتویات محلی که j به آن اشاره می‌کند با محتویات pivot تعویض گشته، محل آن به تابع quickSort برگشت داده می‌شود.

۲. تا زمانی که A[j] بزرگ‌تر از pivot است، مقدار j کاهش می‌یابد.

۳. تا زمانی که A[i] کوچک‌تر و مساوی pivot است، مقدار i افزایش می‌یابد.

٤. چنانچه شرط ۲ و ۱ برقرار نباشد، محتویات A[i] و A[j] جا به جا می‌شوند.

آرایه A با موقعیت عناصر آن در لیست	A[8]	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]

محتویات آرایه A قبل از مرتب‌سازی	3	8	4	18	9	30	6	13	12
انتخاب عنصر اول به عنوان محور	3	8	4	18	9	30	6	13	12

↑pivot ↑i ↑j

چون i < j و ۱۳ از محور (۱۲) بزرگ‌تر و۳ از محور کوچک‌تر است، پس ۱۳ و۳ جا به جا می‌شوند. بعد از جا به جایی اندیس i و j تغییر می‌کند.

جا به جایی ۳ و۱۳	13	8	4	18	9	30	6	3	12

↑pivot ↑i ↑j

چون ۶ از محور کوچک‌تر است، فقط به i اضافه می‌شود. اندیس j تغییر نمی‌کند. چون، زمانی j تغییر می‌کند که از محور بزرگ‌تر باشد یا جا به جایی صورت گیرد.

اضافه شدن به متغیر i	13	8	4	18	9	30	6	3	12

↑pivot ↑i ↑j

چون i < j و ۳۰ از محور (۱۲) بزرگ‌تر و ۸ از محور کوچک‌تر است، پس ۳۰ و ۸ جا به جا می‌شوند. بعد از جا به جایی اندیس i و j تغییر می‌کند.

جا به جایی ۳۰ و۸	13	30	4	18	9	8	6	3	12

↑pivot ↑i ↑j

چون ۹ از محور کوچک‌تر است، فقط به i یک واحد اضافه می‌شود. اندیس j تغییر نمی‌کند.

اضافه شدن به متغیر i	13	30	4	18	9	8	6	3	12

↑pivot ↑i ↑j

چون i < j و ۱۸ از محور یعنی (۱۲) بزرگ‌تر و۴ از محور کوچک‌تر است، پس ۴ و ۱۸جا به جا می‌شوند. بعد از جا به جایی اندیس i و j تغییر می‌کند.

جا به جایی ١٤ و١٨	13	30	18	4	9	8	6	3	12

↑pivot ↑j ↑i

چون j < i، پس مکان مناسب محور(۱۲) مشخص شده و آرایه به دو لیست جداگانه تقسیم می‌شود.

جا به جایی ١٤ و ١٨	13	30	18	12	9	8	6	3	4
	لیست سمت راست		لیست سمت چپ						

مجدداً این الگوریتم را برای لیست چپ {۴،۳،۶،۸،۹} و راست {۱۸،۳۰،۱۳} اجرا می‌کنیم تا در نهایت لیست به صورت زیر مرتب شود.

آرایه A با موقعیت عناصر در لیست	A[8]	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]
محتویات آرایه A بعد از مرتب سازی عناصر آرایه	30	18	13	12	9	8	6	4	3

جدول ۹ – ۷ پیچیدگی زمانی الگوریتم مرتب سازی سریع است. O(nlogn)			
	بهترین حالت	حالت متوسط	بدترین حالت
مرتبه زمانی	O(nlogn)	O(nlogn)	$O(n^2)$

۷ – ۲– ۷. مرتب‌سازی ادغامی

مرتب‌سازی ادغامی از ادغام و الگوریتم تقسیم و غلبه برای مرتب کردن عناصر استفاده می‌کند. در این الگوریتم لیست به دو قسمت مساوی تقسیم می‌شود. هر قسمت مجزا به صورت بازگشتی مرتب شده، با ادغام آن‌ها لیست به صورت کامل مرتب می‌شود. اما احتمال دارد، پس از یک مرحله تقسیم باز هم لیست‌های ایجاد شده بزرگ باشند، باید برای هر لیست ایجاد شده بزرگ مراحل بالا را دوباره به صورت بازگشتی انجام داده تا به زیر لیست‌هایی با تنها ۱ عنصر برسیم. چون لیست تک عنصری خود به خود مرتب است.

☒ **تابع mergeSort**، این تابع بازگشتی با فراخوانی مکرر خود و تابع merge عمل مرتب‌سازی را انجام می‌دهد. تابع mergeSort را در زیر می‌بینید:

```
void mergeSort(int*A, int * tempA, int low, int high, int size)  {
if (low < high)   {
    int middle = (low + high) / 2;
    mergeSort(A, tempA, low, middle, size);
    mergeSort(A, tempA, middle+1, high, size);
    merge(A, tempA, low, middle, high, size);
  }
}
```

☒ **ادغام زیر لیست‌های مرتب شده:** ادغام دو آرایه در هر مرحله یک زیر لیست مرتب از لیست اولیه را تشکیل می‌دهد. در این تابع دو لیست{A[low]...A[middle]} و{A[middle]...A[high]}را در آرایه {Atemp[low]...Atemp[high]} قرار داده، سپس نتیجه ادغام در {A[low]...A[high]} قرار می‌گیرد.

```
void merge(int*A, int*tempA, int low, int middle, int high, int size) {
 int i, j, k;
for (i = low; i <= high; i++)  {
      tempA[i] = A[i]; }
i = low; j = middle+1; k = low;
while ((i <= middle) && (j <= high))  {
  if (tempA[i] <= tempA[j])A[k++] = tempA[i++];
   else   A[k++] = tempA[j++];
 }
while (i <= middle)    A[k++] = tempA[i++];
  }
}
```

این الگوریتم در بهترین حالت، بدترین حالت و حالت متوسط دارای پیچیدگی زمان O(nlogn) است. این الگوریتم نیز به فضای اضافی حافظه که یک آرایه کمکی به طول n است، نیاز داشته، پس روش مرتب-سازی درجا نیست. همچنین از این روش نه تنها برای مرتب‌سازی آرایه‌ها می‌توان استفاده کرد، بلکه برای فایل‌ها نیز می‌توان استفاده نمود. بنابراین، این روش به دلیل استفاده از فضای حافظه اصلی برای آرایه، به

صورت داخلی و به دلیل استفاده از فضای حافظه جانبی برای فایل به صورت خارجی قابل استفاده است. خلاصه عملکرد مرتب‌سازی ادغامی خارجی و داخلی را در جدول ۱۰ - ۷ می‌بینید.

جدول ۱۰ – ۷ خلاصه عملکرد مرتب‌سازی ادغامی خارجی و داخلی.	
روش مرتب‌سازی	تعویضی
وضعیت استفاده از فضای حافظه کمکی	غیر در جا - استفاده از حافظه اضافی با مرتبه O(n)
حالت پایداری	پایدار
مرتبه زمانی	O(nlogn)
بهترین حالت	O(nlogn) (زمانی که لیست نا مرتب باشد).

مثال ۷-۷. لیست A={20,14,17,13,16} شامل ۵ عنصر است. با استفاده از الگوریتم ادغامی به صورت صعودی مرتب نمایید.

A[0]	A[1]	A[2]	A[3]	A[4]
20	13	17	16	14

ابتدا با استفاده از تابع بازگشتی آرایه زیر را به قسمت‌های کوچک‌تر تقسیم کرده، سپس آن‌ها را مرتب می‌نماییم:

تقسیم به قسمت‌های کوچک‌تر که یک لیست با ۳ عنصر {A[0],A[1],A[2]} و لیست دیگر با دو عنصر {A[3]} و {A[4]} تشکیل می‌شود:

A[0]	A[1]	A[2]
20	13	17

A[3]	A[4]
16	14

تقسیم به قسمت‌های کوچک‌تر که لیست ۳ عنصری قبلی به لیست ۲ و ۱ عنصری {A[0],A[1]} و {A[2]} و لیست دو عنصری را به لیست یک عنصری {A[3]} و { A[4]}تبدیل می‌کند:

A[0]	A[1]
20	13

A[2]
17

A[3]
16

A[4]
14

تنها لیست دو عنصری را به لیست یک عنصری {A[1]}و{ A[0]} تبدیل می‌کند:

آرایهA با موقعیت عناصر آن در لیست	A[4]	A[3]	A[2]	A[1]	A[0]
تقسیم به ۲ لیست	14	16	17	13	20

برای ادغام باید عکس مراحل انجام شده به صورت زیر انجام شود:

ادغام {A[1]} و{ A[0]} و جا به جایی عناصر

tempA[0]	tempA[1]
20	13

➔

A[0]	A[1]
13	20

ادغام {A[2]} و{ A[1] ,A[0]}، چون tempA[i] کوچک‌تر از tempA[j] است، پس جا به جایی صورت نمی‌گیرد:

tempA[0]	tempA[1]
13	20
i	

tempA[1]
17
j

➔

A[0]
13
K

ادغام { A[1] ,A[0], A[2]}، چون tempA[i] کوچک‌تر از tempA[j] نیست، پس جا به جایی صورت می‌گیرد و جای ۲۰ و۱۷ تعویض شده، در آرایه A قرار می‌گیرد:

tempA[0]	tempA[1]
13	20
	I

tempA[2]
17
j

➔

A[0]	A[1]
13	17
	K

ادغام کامل {A[2]} , { A[1] ,A[0]}:

tempA[0]	tempA[1]
13	20
	I

tempA[2]
17
j

➔

A[0]	A[1]	A[2]
13	17	20
		K

ادغام { A[3] ,{A[4]}و جا به جایی عناصر :

tempA[3]
16

tempA[4]
14

➔

A[3]	A[4]
14	16

در نهایت ادغام کل عناصر به همان روش فوق:

tempA[0]	tempA[1]	tempA[2]
13	17	20
i		

tempA[3]	tempA[4]
14	16
j	

با مقایسه و جا به جایی عناصر و قرار گرفتن نتیجه در آرایه A لیست به صورت زیر ادغام می‌شود:

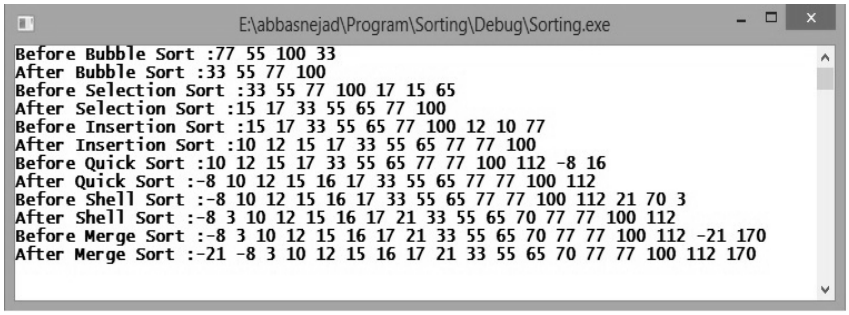
A[0]	A[1]	A[2]	A[3]	A[4]	آرایه A با موقعیت عناصر آن در لیست
13	14	16	17	20	محتویات آرایه A بعد از مرتب-سازی

پیاده سازی ۱ – ۷. برنامه‌ای که مرتب‌سازی‌های حبابی (bubble)، انتخاب (selection)، درجی (insertion)، سریع (Quick)، ادغامی (merge) و Shell را انجام می‌دهد.

```
#include "stdafx.h"
#include <iostream>
#include <vector>
using namespace std;
class Array {
 private:
  vector<double> v;
  int count;
  void swap(int i, int j)  {
   double temp = v[i];
   v[i] = v[j];
   v[j] = temp;
  }
 public:
  Array(int max) : count(0)  {   v.resize(max);  }
  void insert(double value)  {
   v[count] = value;
   count++;
  }
  void display()  {
   for(int j=0; j<count; j++)   cout << v[j] << " ";
   cout << endl;
  }
  void bubbleSort()  {
   for(int i=count-1; i>0; i--)
    for(int j=0; j<i; j++)  if( v[j] > v[j+1] )  swap(j, j+1);
  }
 void selectionSort()  {
for( int i = 0; i< count - 1 ; i++){
        int min = i ;
        for( int j = i + 1; j < count ; j ++)if(v[min] > v[j])  min = j ;
        if( min != i) swap(i, min);
       }
 }
```

```
 void insertionSort()  {
  for(int i=1; i<count; i++)  {
   double temp = v[i];
   int j = i;
   while(j>0 && v[j-1] >= temp)   {
    v[j] = v[j-1];
    --j;
   }
   v[j] = temp;
  }
 }
 void quickSort()  {   QuickSort(0, count-1);  }
 void QuickSort(int left, int right)  {
int i = left, j = right;
int tmp;
int pivot = v[abs((left + right) / 2)];
 while (i <= j) {
         while (v[i] < pivot)
         i++;
      while (v[j] > pivot)      j--;
      if (i <= j) {
           swap(i, j);
           i++;
           j--;
         }
       }
if (left < j) QuickSort(left, j);
if (i< right) QuickSort(i, right);
 }
 void shellSort()  {
  int m=count,  flag;
  do
  {
    flag=0;
   m=(m+1)/2;
   for(int i=0;(i+m)< count;i++) {
     if(v[i]>v[i+m])   {
       swap(i, i+m);
          flag=1;
     }
      }
} while(flag!=0 || m>1);
 }
 void mergeSort() {
  vector<double>(workSpace);
  workSpace.resize(count);
  MergeSort(workSpace, 0, count-1);
 }
 void MergeSort(vector<double> workSpace,int low, int up)  {
  if(low == up)    return;
  else
  {
   int mid = (low+up) / 2;
   MergeSort(workSpace, low, mid);  //sort low half
   MergeSort(workSpace, mid+1, up);//sort high half
   merge(workSpace, low, mid+1, up); //merge them
  }
 }
 void merge(vector<double> workSpace, int lowPtr,int highPtr, int up)
```

```
 {
  int j = 0;
  int low = lowPtr;
  int mid = highPtr-1;
  int n = up-low+1; //# of items
  while(lowPtr <= mid && highPtr <= up)
   if( v[lowPtr] < v[highPtr] )  workSpace[j++] = v[lowPtr++];
   else  workSpace[j++] = v[highPtr++];
  while(lowPtr <= mid)  workSpace[j++] = v[lowPtr++];
  while(highPtr <= up)
  workSpace[j++] = v[highPtr++];
  for(j=0; j<n; j++) v[low+j] = workSpace[j];
 } //end merge
};
  int main()  {
  int maxSize = 30;
  Array arr(maxSize);
  arr.insert(77); arr.insert(55);    arr.insert(100);  arr.insert(33);
  cout << "Before Bubble Sort :";    arr.display();
  arr.bubbleSort();
  cout  << "After Bubble Sort :";    arr.display();
  arr.insert(17); arr.insert(15);    arr.insert(65);
  cout << "Before Selection Sort :";  arr.display();
  arr.selectionSort();
  cout << "After Selection Sort :";   arr.display();
  arr.insert(12);  arr.insert(10);    arr.insert(77);
  cout << "Before Insertion Sort :";  arr.display();
  arr.insertionSort();
  cout << "After Insertion Sort :";   arr.display();
  arr.insert(112);  arr.insert(-8);   arr.insert(16);
  cout << "Before Quick Sort :";      arr.display();
  arr.quickSort();
  cout << "After Quick Sort :";      arr.display();
  arr.insert(21);  arr.insert(70);   arr.insert(3);
  cout << "Before Shell Sort :";     arr.display();
  arr.shellSort();
  cout << "After Shell Sort :";      arr.display();
  arr.insert(-21);  arr.insert(170);
  cout << "Before Merge Sort :";     arr.display();
  arr.mergeSort();
  cout << "After Merge Sort :";      arr.display();
  getchar();
 return 0;
}
```



این برنامه شامل کلاس Arrayو تابع main: است که عملکرد آن‌ها را در زیر می‌بینید

۱. **کلاس Array()**، اعضای داده‌ای v (عناصر آرایه از نوع double)، count (تعداد عناصر فعلی آرایه) و اعضای تابعی زیر را دارد:

☒ **تابع swap()**، i و j را به عنوان پارامتر دریافت کرده، مقادیر عناصر v[i] و v[j] را تعویض می‌کند.

☒ **تابع Array()**، سازنده کلاس است که max (حداکثر تعداد عناصر آرایه) را به عنوان پارامتر دریافت کرده، آرایه‌ای به اندازه max ایجاد می‌کند و تعداد عناصر فعلی (count) را برابر صفر قرار می‌دهد.

☒ **تابع insert()**، پارامتر value از نوع double را دریافت کرده، آن را به آرایه اضافه می‌کند.

☒ **تابع display()**، عناصر آرایه را نمایش می‌دهد.

☒ **تابع bubleSort()**، مرتب‌سازی حبابی را برای آرایه انجام می‌دهد.

☒ **تابع selecttionSort()**، مرتب‌سازی انتخابی را بر روی آرایه انجام می‌دهد.

☒ **تابع insertionSort()**، مرتب‌سازی درجی را بر روی آرایه انجام می‌دهد.

☒ **توابع quickSort() و QuickSort()**، مرتب‌سازی سریع را بر روی آرایه انجام می‌دهند.

☒ **تابع shellSort()**، مرتب‌سازی shell را بر روی آرایه انجام می‌دهند.

☒ **توابع mergeSort()، MergeSort() و Merge()**، مرتب‌سازی ادغامی را بر روی آرایه انجام می‌دهند.

۲. **تابع main()**، شامل دستوراتی است که عناصری را به آرایه اضافه می‌کند و سپس هر یک از مرتب‌سازی‌ها را آزمایش می‌نماید.

۳-۷.تست‌های ارشد مرتب‌سازی

۱. دنباله‌ای از اعداد وجود دارد. اگر برخی از این اعداد تکراری باشد و در کل k عدد غیر تکراری باشد، آن‌گاه با چه مرتبه زمانی مناسبی می‌توان این دنباله را مرتب کرد؟(علوم کامپیوتر – دولتی ۹۰).

الف: O(nlogn)　　ب: O(nlogk)　　ج: O(klogn)　　د: O(klogk)

۲. فرض کنید تعدادی داده داریم که به صورت نزولی مرتب شده‌اند. با اجرای یک الگوریتم مرتب‌سازی صعودی روی آن، بعد از دو مرحله اجرا داده‌ها به صورت زیر درآمده‌اند:

48, 59, 61, 77, 11, 15, 19, 26, 1, 5

الگوریتم اجرا شده چه نوع Sortای می‌باشد؟(علوم کامپیوتر – دولتی ۸۸).

الف: Insertion Sort　　ب: Merge Sort　　ج: Quick Sort　　د: Heap Sort

۳. الگوریتم Radix Sort را روی n عدد در فاصله $[0, n^2-1]$ اجرا می‌کنیم. پایه استفاده شده در Radix Sort برابر n است. متوسط زمان اجرای این الگوریتم چقدر است؟(فناوری اطلاعات – دولتی ۸۸).

الف: $\theta(n^2)$　　ب: $\theta(n)$　　ج: $\theta(nlogn)$　　د: $\theta(n^2 logn)$

٤. می‌دانیم هزینه‌های مرتب‌سازی درجی (Insertion Sort) برای مرتب‌سازی یک آرایه‌ی A با n عنصر مناسب با تعداد "وارونگی"(Inversion)های عناصر آن آرایه است. زوج (i, j) را یک عدد وارونگی می‌گوییم، اگرj > i و A[j]<A[i] با فرض احتمال این که یک زوج اندیس دلخواه از یک وارونگی باشد، برابر $\frac{1}{2}$ است، میانگین تعداد وارونگی‌های یک آرایه‌ی A با عناصر متمایز چقدر است؟(مهندسی کامپیوتر – دولتی ۸۹).

الف: $\frac{n^2-n}{2}$　　ب: $\frac{n^2}{2}$　　ج: $\frac{n^2}{4}$　　د: $\frac{n^2-n}{4}$

۵. در الگوریتم مرتب‌سازی آرایه A با n فرض کنید b>1 یک عدد ثابت است. همچنین فرض کنید هزینه مقایسه‌ی دو عنصرA[i] و A[j] با تعویض آن‌ها، اگر $|j-1| \leq b$ برابر صفر (خیلی کم)، و در غیر این صورت، برابر ۱(خیلی زیاد) است. توجه کنید که این فرض، هزینه مرتب‌سازی درجی، حبابی(Bubble Sort) برابر O(1) می‌شود. با این فرض

هزینه‌ی مرتب‌سازی ادغامی (Merge Sort)، A در بدترین حالت چقدر است؟(بهترین جواب را انتخاب کنید). بدیهی است که اگر T(n) زمان اجرا باشد، داریم: b > n و 1= T(n)(مهندسی کامپیوتر – دولتی ۸۹).

الف: O(nlog(n/b))ب: O(n/blog(n/b)) ج: O(nlogn) د: O(nlogn/b)

٦. کدام یک از الگوریتم‌های زیر دارای بیشترین پیچیدگی حافظه است؟(علوم کامپیوتر – دولتی ۸۹).

الف: Quick Sort ب: Insertion Sort ج: Heap Sort د: Merge Sort

۷. اگر برای الگوریتم مرتب‌سازی آرایه به صورت صعودی از الگوریتم Bubble Sort استفاده شود، چند عمل مقایسه و چند عمل جا به جایی صورت می‌گیرد؟{3,5,2,1,4,7} (علوم کامپیوتر – دولتی ۸۹).

الف: ۳ عمل جا به جایی و ۳۰ عمل مقایسه صورت می‌گیرد. ب: ۳ عمل جا به جایی و۱۵ عمل مقایسه صورت می‌گیرد.

ج: ۶ عمل جا به جایی و ۱۴ عمل مقایسه صورت می‌گیرد. د: ۳۰ عمل جا به جایی و ۳۰ عمل مقایسه صورت می‌گیرد.

۸. در الگوریتم Merge Sort، اگر به جای آن که هر بار لیست به دو قسمت تقسیم شود، به چهار قسمت مساوی تقسیم گردد و در هر مرحله ترکیب این چهار لیست در یک دیگر ادغام شوند، پیچیدگی زمانی الگوریتم چه خواهد شد؟(مهندسی کامپیوتر – آزاد۸۰).

الف: $\theta(n^{\frac{3}{4}})$ ب: $\theta(nlogn)$ ج: $\theta(n^2)$ د: $\theta(n^2\ log_4^n)$

۴ – ۷. پاسخ تشریحی تست‌های ارشد مرتب‌سازی

۱. گزینه (ب) صحیح است.

۲. گزینه (ب) صحیح است. چون آرایه اولیه مرتب شده نزولی بوده است. پس، آرایه اولیه به صورت زیر بوده است:

77, 61, 59, 48, 26, 19, 15, 11, 5, 1

) آرایه به شکل زیر تغییر می‌یابد(هر دو تا Merge Sortبا اجرای اولین مرحله الگوریتم مرتب‌سازی ادغامی(جداگانه مرتب می‌شوند):

61, 77, 48, 59, 19, 26, 11, 15, 1, 5

با اجرای مرحله دوم الگوریتم مرتب‌سازی آرایه به صورت زیر تغییر می‌یابد(چهار عنصر، چهار عنصر مرتب خواهند شد):

48, 59, 61, 77, 11, 15, 19, 26, 1, 5

۳. گزینه (ب) صحیح است. چون مرتبه اجرایی الگوریتم مبنا (پایه یا Radix Sort)، $\theta(s\times n)$ است که در آن s تعداد ارقام اعداد و n تعداد اعداد موجود در آرایه می‌باشد. از آنجای که اعداد در فاصله $[0, n^2\text{-}1]$ قرار دارند، پس تعداد ارقام عدد کم می‌باشد. به عنوان مثال، n =100 تعداد ارقام عدد 4 می‌باشد (مقادیر[0, 9999]). پس $\theta(4 \times n)$، معادل $\theta(n)$ است.

٤. گزینه (د) صحیح است. در حالتی که هزینه‌ی الگوریتم مرتب‌سازی درجی $\frac{n(n-1)}{2} = \frac{n^2-n}{2}$ است که نصف آن‌ها وارونگی دارند. پس، $\frac{n^2-n}{4}$ صحیح است.

۵. گزینه (الف) صحیح است. همان طور که می‌دانید در مرتب‌سازی ادغامی اندازه آرایه در هر مرحله نصف می‌گردد. چون فرض شده است که آستانه نصف شدن آرایه b باشد. بنابراین، برای این مرحله $O(\ log^{\frac{n}{b}})$ زمان نیاز است. از

طرف دیگر، چون برای ادغام دو آرایه مرتب شده با n عنصر به n-1 مقایسه نیاز است که همان O(n) می‌باشد. بنابراین، زمان کل اجرای این الگوریتم $O(nlog\frac{n}{b})$ خواهد بود.

٦. گزینه (د) صحیح است. فضای استفاده شده مرتب‌سازی‌های Quick (سریع)، Insertion(درجی) و هرمی (Heap) تابعی از اندازه آرایه نیست. امّا فضای استفاده شده مرتب‌سازی Merge (ادغامی)، تابعی از اندازه آرایه می‌باشد.

۷. گزینه (ج) صحیح است. حداکثر تعداد مقایسه‌ها در مرتب‌سازی حبابی برابر با $\frac{n(n-1)}{2}$ $(\frac{(6*5)}{2}=۱۵)$ است. پس گزینه‌های (الف) و (د) نادرست هستند. حال اگر الگوریتم مرتب‌سازی حبابی را اجرا کنید وتعداد جا به جایی‌ها را بشمارید، به گزینه (ج) می‌رسید.

۸. گزینه (ب) صحیح است.