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	:	QA۷٦/۹/س۲ع۲۳ 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 ٦٦٤۰۰۲۲۰-٦٦٤۰۰۱٤٤

تلفکس:

فهرست مطالب

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

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، پاییز۱۳۹۲

مولفین

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 الگوریتم‌ها مورد استفاده قـرار 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:

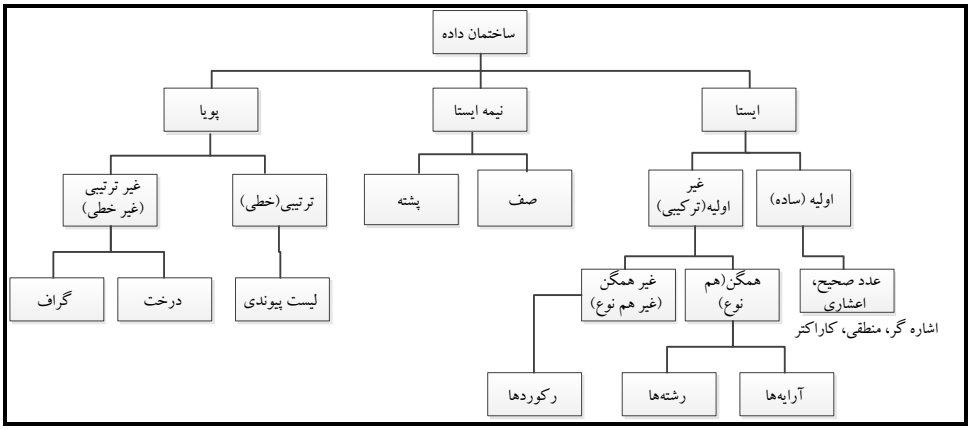
☒**ساختمان داده‌های اولیه،** ساختمان داده‌هایی از قبیل داده‌های عـددی صـحیح (int، long، unsigned int و ...)، عددی اعشاری (float و double)، کاراکتری (char)، منطقی (bool) و اشاره‌گر، ساختمان داده اولیـه هستند. برای تعریف متغیری از این نوع ساختمان داده به صورت زیر عمل می‌شود:

نوع داده لیست متغیرها ;

به عنوان مثال، دستورات زیر را ببینید:

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

دستور اول، متغیرهـای x و y را از نـوع صـحیح تعریـف مـی‌کنـد، دستور دوم، متغیر q را از نوع منطقـی تعریـف مـی‌نمایـد و دسـتور 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];
```

۲. **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:

```
struct نام ساختمان {
   نوع فیلد۱ نام فیلد۱;
   نوع فیلد۲ نام فیلد ۲;
   …
   نوع فیلد n نام فیلد n ;
}
```

دستورات زیر را در نظر بگیرید:

```
struct Book {
   char ISBN[14];
   char bookName[50];
   long price;
}
```

این دستور، ساختمانی به نام Book تعریف می‌کند که دارای فیلدهای شابک کتاب (از نوع کاراکتری شامل ۱۴ کاراکتر)، نام کتاب (از نوع کاراکتری با ۳۰ کاراکتر) و قیمت از نوع صحیح (با طول بلند) است (شکل زیر):

ISBN	bookName	Price

۳. **رشته**، مجموعه‌ای از کاراکترهای پشت سرهم حافظه که با '\0' خاتمه می‌یابند. در زبان C++، رشته به صورت زیر تعریف می‌شود:

```
char نام رشته [تعداد];
```

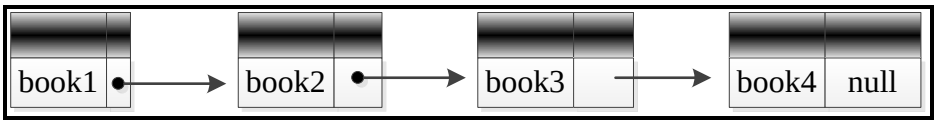
به عنوان مثال، دستور زیر رشته‌ای به نام s با ۸۰ کاراکتر تعریف می‌کند:

```
char s[80];
```

۲-۱-۱. ساختمان داده‌های پویا

ساختمان داده‌هایی هستند که تعداد عناصر آن‌ها در هنگام ترجمه مشخص نیست. به عنوان مثال، اگر بخواهید ساختمان داده‌ای داشته باشید که تعدادی شابک، نام کتاب و قیمت را نگهداری کند، باید از ساختمان‌های داده پویا استفاده کنید. چون، تعداد کتاب‌ها در زمان ترجمه مشخص نیست. انواع ساختمان داده-های پویا عبارت انداز:

☒**ساختمان داده‌های پویای ترتیبی (خطی)**، در این نوع ساختمان داده‌ها هر عنصر به عنصر قبلی یا بعدی‌اش اشاره می‌کند. نمونه‌ای از این ساختمان داده می‌توان **لیست پیوندی** را نام برد. لیست پیوندی انواع مختلف دارد که در فصل ۴ به آن‌ها می‌پردازیم. ساده‌ترین لیست پیوندی، لیست تک پیوندی است. در این نوع لیست پیوندی هر عنصر از دو بخش **داده**[1] و **پیوند**[2] تشکیل می‌شود. پیوند به داده بعدی لیست اشاره می‌کند. اما، مقدار پیوند آخرین عنصر لیست NULL است. نمونه‌ای از لیست پیوندی که اطلاعات کتاب‌ها را نگهداری می‌کند، در شکل زیر آمده است:



تعریف این ساختار به صورت زیر است:

```
struct ListBook {
  char ISBN[14];
  char bookName[30];
  long price;
  ListBook *Link;
}
```

☒**ساختمان داده‌های غیر ترتیبی پویا(غیر خطی)**، در این ساختمان داده، عناصر به عنصر بعدی یا قبلی‌شان اشاره نمی‌کنند. به عنوان نمونه‌هایی از این نوع ساختمان داده می‌توان **درخت** و **گراف** را نام برد. در فصل‌های ۵ و ۶ با مفاهیم و پیاده‌سازی درخت و گراف آشنا خواهیم شد.

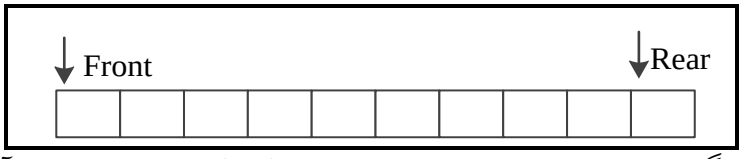
۳-۱-۱. ساختمان داده‌های نیمه ایستا

ساختمان داده‌های نیمه ایستا، ساختمان داده‌هایی هستند که گاهی اوقات از طریق ساختمان داده‌های ایستا و گاهی از طریق ساختمان داده‌های پویا قابل پیاده‌سازی هستند. در این ساختمان داده‌ها، اگر تعداد عناصر در هنگام ترجمه مشخص باشد، از طریق ساختمان داده‌های ایستا مانند آرایه می‌توان آن‌ها را پیاده‌سازی نمود. اما، اگر تعداد عناصر آن‌ها در زمان ترجمه مشخص نباشد، باید آن‌ها را از طریق ساختمان داده‌های پویا نظیر لیست پیوندی پیاده‌سازی نمود. این نوع ساختمان داده‌ها در زیر آمده‌اند:

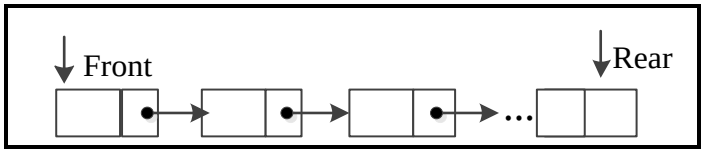
---

[1].Data [2].Link

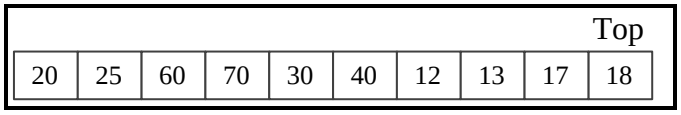
☒ **صف**[1]، ساختمان داده‌ای که در آن اولین ورودی اولین خروجی[2] باشد. یعنی، عناصر در سر[3] صف اضافه می‌شوند و از انتهای[4] آن حذف می‌شوند. اگر در هنگام ترجمه تعداد عناصر صف مشخص باشد، از طریق آرایه می‌توان آن را پیاده‌سازی نمود. این پیاده‌سازی را در شکل زیر می‌بینید:



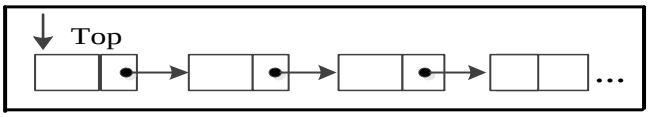
اما، اگر در هنگام ترجمه تعداد عناصر صف مشخص نباشد، از طریق لیست پیوندی می‌توان آن را پیاده-سازی نمود (شکل زیر):



☒ **پشته**[5]، ساختمان داده‌ای که اولین ورودی آن، آخرین خروجی باشد[6] یا آخرین ورودی آن، اولین خروجی است[7]. 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.

۲-۱. الگوریتم

الگوریتم، مجموعه‌ای از دستورالعمل‌ها که مراحل انجام کاری را به **زبان دقیق**، با **جزئیات کافی** بیان نماید، **ترتیب اجرای دستورات** و **شرط خاتمه** آن مشخص باشد.

همان‌طور که در این تعریف می‌بینید، برخی از ویژگی‌های الگوریتم عبارت‌اند از:

---

[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)

۱. **قطعیت[1](عدم ابهام)**، دستورالعمل‌ها هیچ گونه ابهامی نداشته باشند. دستورالعمل‌ها به زبان دقیق بیان شوند.
۲. **پایان پذیر باشد[2]**، الگوریتم باید پس از انجام مراحل خاصی خاتمه یابد.
۳. **ترتیب اجرای آن مشخص باشد،** مراحل انجام دستورالعمل‌ها در الگوریتم باید مشخص باشد.
۴. **ورودی[3]**، الگوریتم می‌تواند ورودی داشته باشد یا ورودی نداشته باشد.
۵. **خروجی[4]**، الگوریتم باید حداقل یک خروجی داشته باشد.
۶. **کارائی**، الگوریتم باید قابل پیاده‌سازی باشد. یعنی، بتوان دستورالعمل‌های آن را به صورت دستی و گام به گام دنبال کرد.

الگوریتم‌ها توسط زبان‌های برنامه‌سازی پیاده‌سازی می‌شوند. با پیاده‌سازی الگوریتم توسط زبان‌های برنامه سازی، **برنامه[5]** ایجاد می‌شود. تفاوت بین برنامه و الگوریتم در زیر آمده است:

۱. الگوریتم باید خاتمه پذیر باشد. اما، برنامه می‌تواند خاتمه نداشته باشد (مانند سیستم عامل‌ها که هیچ گاه خاتمه نمی‌یابند).
۲. برنامه‌ها توسط رایانه‌ها قابل فهم و اجرا می‌باشند. اما، الگوریتم‌ها توسط رایانه‌ها قابل اجرا نیستند.

۱-۲-۱. کارایی الگوریتم

برای حل یک مسئله ممکن است الگوریتم‌های مختلفی وجود داشته باشد. به عنوان مثال، فرض کنید بخواهید الگوریتمی بنویسید که یک عدد را از ورودی خوانده، تعیین کند اول است یا خیر. برای تعیین عدد اول تعاریف زیر وجود دارد:

☒ عددی اول است که مجموع مقسوم علیه‌های آن برابر با یک باشد.
☒ عددی اول است که بر هیچ عدد کوچک‌تر از خودش به جز یک بخش پذیر نباشد.
☒ عددی اول است که بر هیچ عدد اول کوچک‌تر از خودش بخش‌پذیر نباشد.

یا فرض کنید بخواهید محتوی متغیرهای x و y را تعویض کنید که به ترتیب مقادیر ۷ و ۱۰ دارند. برای حل این مسئله روش‌های مختلفی وجود دارد که عبارت‌اند از:

۱. **استفاده از متغیر کمکی برای تعویض مقادیر x و y**، این روش به صورت مقابل انجام می‌شود.

```
t = x;  ⇨  t = 7
x= y;  ⇨  x = 10
y = t;  ⇨  y = 7
```

۲. **استفاده از عملگرهای + و – برای تعویض مقادیر x و y**، این روش به صورت زیر انجام می‌شود:

```
x = x + y;  ⇨ x = 7 + 10 = 17
y = x - y;  ⇨   y = 17 – 10 = 7
x = x - y;  ⇨ x = 17 – 7 = 10
```

۳. **استفاده از عملگرهای * و تقسیم برای تعویض مقادیر x و y**، در این روش به صورت زیر عمل می‌شود:

```
x = x * y;  ⇨   x = 7 * 10 = 70
y = x / y;  ⇨   y = 70  / 10 = 7
x = x / y;  ⇨   x = 70 / 7 = 10
```

همان‌طور که در روش‌های مختلف حل مسائل بیان شده مشاهده کردید، کارائی الگوریتم به دو عامل اصلی بستگی دارد. ۱.**پیچیدگی حافظه**، مقدار حافظه‌ای که برای اجرای الگوریتم مورد نیاز است. به عنوان مثال، در روش

---
[1].Definiteness [2].Finitness [3].Input [4].Output [5]. Program

اول برای تعویض مقدار دو متغیر x و y به یک متغیر کمکی مانند t نیاز است. پیچیدگی حافظه یک الگوریتم با حرف S نمایش داده می‌شود. ۲. **پیچیدگی زمانی**، مدت زمانی که برای اجرای الگوریتم مورد نیاز است. به عنوان مثال، هر یک از روش‌های تعیین عدد اول، دارای پیچیدگی زمانی متفاوتی هستند. پیچیدگی زمانی الگوریتم با حرف T نمایش داده می‌شود.

الگوریتمی کاراتر است که فضای حافظه کمتری را اشغال کند و جهت اجرا به زمان کمتری از پردازشگر نیاز داشته باشد. پیچیدگی الگوریتم را به دو روش می‌توان محاسبه کرد که عبارت‌اند از:

۱. **محاسبه دقیق زمان اجرا و حافظه مصرفی الگوریتم**، این روش نه تنها مشکل است، بلکه به نوع ماشینی که الگوریتم بر روی آن اجرا می‌شود، بستگی دارد. به عنوان مثال، یک نوع int در رایانه‌های ۱۶ بیتی، دو بایت از حافظه را اشغال می‌کند، ولی همین نوع int در رایانه‌های ۳۲ بیتی، چهار بایت از حافظه را اشغال می‌نماید. از طرف دیگر، همیشه به محاسبه دقیق زمان اجرا و حافظه مصرفی الگوریتم نیازی نیست.

۲. **محاسبه پیچیدگی الگوریتم به صورت تابعی از ورودی**، این روش نه تنها به نوع ماشین بستگی ندارد، بلکه بدون این که زمان اجرا و حافظه مصرفی الگوریتم به طور دقیق محاسبه شود، می‌تواند برای سنجش کارائی الگوریتم‌های مختلف به کار رود.

دستورالعمل‌های الگوریتم به دستورات برنامه تبدیل می‌شوند تا بتوانند توسط رایانه قابل اجرا باشند. دستورات برنامه دو نوع‌اند که عبارت‌اند از:

۱. **دستورات غیر اجرائی**، دستوراتی هستند که زمان پردازشگر را مصرف نمی‌کنند. این دستورات دارای گام صفر هستند. نمونه‌ای از این دستورات عبارت‌اند از: ۱. **دستورات تعریف تابع** ۲. **توضیحات برنامه** ۳. **دستورات تعریف متغیر** که مقدار اولیه در متغیر قرار نمی‌دهند. ۴.**بلاک { و }** ۵.**دستورات تعریف کلاس‌ها.**

مثال۱ – ۱. دستورات زیر در C++ دارای گام صفر هستند.

```
// Declare x, i;
int x, i;
void f1(int x, int y);
class Square    {
public:
    ...
}
```

```
1)  int  sum = 0;
2)  x = y * 3 + 5;
3)  for (int  i = 1; i<=n; i++)
    {
      x++;
    }
```

i = 1; ⇨ گام 1

i++; ⇨ گام n

۲. **دستورات اجرائی**، هر دستور اجرائی به ازای هر بار اجرا یک گام است. یعنی، اگر دستور اجرائی در یک حلقه تکرار قرار گیرد که n بار تکرار می‌شود، n گام را می‌گیرد.

مثال۲– ۱. گام‌های اجرای دستورات زیر را بدست آورید:

دستور اول، دارای گام ۱ است. چون، علاوه بر تعریف متغیر sum، مقدار صفر را در آن قرار می‌دهد. دستور دوم، نیز گام یک را دارد. چون، یک دستور اجرائی است. اما، گام دستور سوم به صورت زیر حساب می‌شود:

پس for (int i = 0; i<=n; i++) دارای n + 1 گام است. از طرف دیگر، x++ نیز n بار اجرا می‌شود، پس دارای n گام است. بنابراین، این حلقه تکرار دارای (n + 1 + n) = 2n + 1 گام است.

در دستورات شرطی(if)، عبارت شرطی دارای گام یک است. اما با توجه به نتیجه شرط (درست یا نادرست بودن) یکی از بخش‌های if یا else اجرا می‌شود و با توجه به تعداد دستورات if یا else گام‌های دستور محاسبه می‌گردد.

مثال۳- ۱. گام‌های اجرای دستورات زیر را بدست آورید:

```
int y, x = 5; ⇨(1)
if (x < 10)⇨ (1)
{
   x++;⇨(1)
   y = x--;⇨(1)
}
else
{
   x--;
}
```

گام این دستورات ۴ است. زیرا، دستور اول و دوم هر کدام دارای گام ۱ هستند. چون x کوچک‌تر از ۱۰ است، پس دستورات ;x++ و ;y = x-- اجرا می‌شوند که هر کدام دارای گام ۱ هستند.

مثال۴- ۱. گام‌های اجرای حلقه‌های تو در تو زیر را بدست آورید:

برای محاسبه گام این دستورات به صورت زیر عمل می‌شود:

☒ گام اجرای حلقه خارجی (n + 1) است.

☒ گام اجرای حلقه داخلی (m + 1) است. چون این حلقه n بار اجرا می‌گردد، پس n * (m +1) گام اجرای این حلقه است.

☒ دستور x++ دارای گام اجرای n * m است. بنابراین، گام اجرای این حلقه برابر است با :

(n +1) + n(m + 1) + nm = n + 1 + nm + n + nm = 2nm + 2n + 1

مثال۵- ۱. گام‌های اجرای دستورات زیر را بدست آورید:

```
for (int i = 1; i < n; i++) {
 for(int j = 1; j < m - 1; j++) {
   int x;
   x = y ++;
   y --;
}
```

گام این دستورات به صورت زیر بدست می‌آید:

☒ گام حلقه خارجی، n است. چون، i = 1 یک بار و i++، n – 1 مرتبه (زیرا، از ۱ تا n – 1 تغییر می‌یابد) اجرا می‌شود (n =1 -1 + 1).

☒ گام حلقه داخلی (m -1) * (n – 1) است. چون حلقه خارجی (n – 1) بار حلقه داخلی را اجرا می‌کند و حلقه داخلی نیز m – 1 مرتبه اجرا می‌شود. زیرا، j = 1، یک بار اجرا می‌شود و j از ۱ تا m – 2 تغییر می‌کند و j++ می‌شود (یعنی j++، m – 2 بار اجرا می‌شود).

☒ گام اجرای دستور int x;، صفر است. چون تعاریف متغیرها دارای گام صفر هستند. اما، دستورات x = y++ و y--; هر یک به اندازه (m – 2) * (n – 1) مرتبه اجرا می‌شوند. پس داریم:

```
for (int i = 1; i <= n; i++)   {
 for (int j = 1; j <= m; j++)  {
   x++;
 }
}
```

2 * (n – 1) * (m – 2)

بنابراین، گام کل دستورات برابر است با:

n + (n – 1) (m – 1) + 2 (n – 1) (m – 2) = n + nm – n – m +1+ 2 (nm – 2n – m + 2) =
nm – m + 1 + 2nm – 4n – 2m + 4 = 3nm – 3m – 4n + 3

مثال۶- ۱. گام اجرای تابع زیر را بدست آورید:

```
int calculateSum (int n)  {
 int sum = 0;
 for (int i = 1; i <= n; i++)   sum += i;
 return sum;
}
```

گام این تابع به صورت زیر محاسبه می‌شود:

☒ دستور اول، دارای گام صفر است. چون تعاریف توابع گام صفر دارند.

☒ دستور دوم، int sum = 0; دارای گام یک است. چون علاوه بر تعریف متغیر sum، مقدار 0 را در آن قرار داده است.

☒ حلقه تکرار دارای گام (n + 1) است.

☒ دستور sum += i; دارای گام n است. چون n بار اجرا می‌شود. بنابراین، گام کل دستورات برابر 2n + 2 (1 + n + 1 + n) است.

مثال۷ – ۱. گام اجرای دستورات زیر را بدست آورید:

```
int sum = 0;
for (int  i = n; i  > 1; i--)
   sum += i;
```

دستور int sum = 0; دارای گام یک است، حلقه تکرار دارای گام n + 1 می‌باشد و دستور sum +=i; دارای گام n است. بنابراین، گام کل دستورات برابر با 2n + 2 = (1 + n + 1 + n) است.

مثال ۸ – ۱. گام اجرای دستورات زیر را بدست آورید:

```
int sum = 0;
for (int  i=0; i < n; i *=2)
  for (int  j=m; j >=1; j /= 3)
    sum +=i;
```

دستور int sum = 0; دارای گام یک است، حلقه تکرار for با شمارنده i دارای گام $log_2^n + 1$ است، حلقه تکرار for با شمارنده j دارای گام $(log_2^n)(log_3^m + 1)$ است و دستور sum += i; دارای گام $(log_2^n)(log_3^m)$ می‌باشد. بنابراین گام کل دستورات برابر است با:

$$T(n) = 1 + (log_2^n + 1) + \big((log_2^n)(log_3^m + 1)\big) + log_2^n(log_3^m)$$

مثال ۹ – ۱. گام اجرای قطعه برنامه زیر چند است:

```
int sum = 0;
for (int i=0; i< n; i +=2)
  for (int j=m; j >=1; j -= 3)
    sum +=1;
```

دستور int sum = 0; دارای گام یک است، دستور for با شمارنده i دارای گام $\frac{n}{2} + 1$ می‌باشد، دستور for با شمارنده j دارای گام $\left(\frac{n}{2}\right)\left(\frac{m}{3} + 1\right)$ است و دستور sum+=i; دارای گام $\left(\frac{n}{2}\right) * \left(\frac{m}{3}\right)$ است. پس گام کل دستورات برابر با عبارت $T(n) = 1 + \left(\frac{n}{2} + 1\right) + \left(\left(\frac{n}{2}\right)\left(\frac{m}{3} + 1\right)\right) + \left(\frac{n}{2}\right)\left(\frac{m}{3}\right)$ است.

مثال ۱۰ – ۱. گام اجرای دستورات زیر را بدست آورید:

```
int x = 0;
for (int i=2; i <= n+5; i++)
  for (int j=n; j >3; j -=2)
    for (int k=1; k <2*n + 1; k*=3)
        x++;
```

دستور int x = 0; دارای گام یک است، حلقه for با شمارنده i دارای گام n + 5 می‌باشد، دستور for با شمارنده j، دارای گام $\left(\frac{n-3}{2} + 1\right)$ (خود حلقه) ضرب در n + 4 (حلقه بالائی) است، حلقه for با شمارنده k دارای گام $log_3^{2n+1} + 1$ (خود حلقه) * $\frac{n-3}{2}$ (برای حلقه با شمارنده j) *n + 4 (برای حلقه با شمارنده i) است و دستور x++; دارای گام $(log_3^{2n+1}) * \frac{n-3}{2} * (n + 4)$ است. پس گام کل دستورات برابر است با:

$$T(n) = 1 + (n + 5) + (n + 4)\left(\left(\frac{n-3}{2} + 1\right) + \frac{n-3}{2}\left((log_3^{2n+1} + 1) + log_3^{2n+1}\right)\right)$$

همان‌طور که مشاهده کرده‌اید، محاسبه گام اجرای دستورات کار سختی است. بنابراین، بهتر است از مرتبه اجرائی الگوریتم استفاده شود. مرتبه اجرائی را در ادامه می‌آموزیم.

۲-۲-۱. مرتبه اجرائی

برای مقایسه الگوریتم‌ها، سه نماد O و Ω و θ وجود دارد که در ادامه هر کدام از آن‌ها را توضیح خواهیم داد. از میان این نمادها به طول معمول از نماد O استفاده می‌شود. تعریف نماد O به صورت زیر است:

$$f(n) \in O(g(n)) \Leftrightarrow \exists n_0 > 0, c > 0 \quad \forall n \geq n_0 : f(n) \leq cg(n)$$

در این رابطه n_0 و c مقادیر ثابت هستند. $\exists c$، یعنی حداقل یک c وجود دارد و $\forall n$ به معنای همه مقادیر n است. اگر $f(n) \in O(g(n))$، آن گاه g(n) یک کران بالا برای f(n) خواهد بود.

قضیه:	اگر $f(n) = a_m n^m + \cdots + a_1 n + a_0$ باشد، آنگاه $f(n) \in O(n^m)$ خواهد بود.

به عنوان مثال، تابع f(n) مقابل را در نظر بگیرید: $f(n) = 4n + 4$

$f(n) \in O(n)$ است. زیرا: $\forall n \geq 3 : 4n + 4 \leq 6n$

پس $n_0 = 3$ و $c = 5$ و $g(n) = n^2$ می‌باشد. بنابراین می‌توان نتیجه گرفت که جمله غالب f(n) (جمله‌ای که رشد آن از دیگر جملات بیشتر باشد) برابر با $g(n)$ است. یعنی داریم:

$f(n) = 4n + 4 \in O(n)$
$f(n) = 10n^3 + 5n^2 + 6n + 4 \in O(n^3)$
$f(n) = n + log_2^n \in O(n)$
$f(n) = log_2^n + 5 \in O(log_2^n)$
$f(n) = 5 \in O(1)$

به عنوان مثال، تابع f(n) زیر را ببینید:

$f(n) = 5n^2 + 2n + 3$

$f(n) \in O(n^2)$ است. زیرا: $\forall n \geq 3 : 5n^2 + 2n + 3 \leq 5n^2$

پس $n_0 = 3$ و $c = 6$ و $g(n) = n$ است.

مثال ۱۱ - ۱. مرتبه اجرائی تابع زیر چیست:

$$f(n) = \frac{n}{3}\left(\frac{n}{2} - 1\right) = \frac{n^2}{6} - \frac{n}{3} \in O(n^2)$$

مرتبه اجرائی برخی از توابع مهم به ترتیب از چپ به راست بر اساس میزان زمانی که مصرف می‌کنند، در جدول ۱-۱ آمده‌اند.

اگر $f(n) \in O(g(n))$ باشد، آن‌گاه مرتبه اجرائی f(n)، برابر تمام توابع بزرگ‌تر از $g(n)$ نیز خواهد بود. به عنوان مثال، مرتبه اجرائی توابع زیر دارای این شرایط هستند:

$$f(n) = 4n^3 + n^2 + 6 \Rightarrow f(n) \in O(n^3) \Rightarrow f(n) \in O(n^4) => f(n) \in O(n^5)$$
...

جدول ۱-۱ مرتبه اجرائی برخی از توابع مهم.

توانی	فاکتوریل	نمایی	چند جمله‌ای	-	خطی	لگاریتمی	ثابت	تابع
$O(n^n)$	O(n!)	$O(a^n)$	$O(n^a)$	O(nlogn)	O(n)	O(logn)	O(1)	مرتبه اجرائی

برای محاسبه مرتبه اجرائی (O) به نکات زیر توجه کنید:

☒ دستورات انتساب، محاسباتی، رابطه‌ای، منطقی شرطی (نظیر if، switch) و دستورات ورودی و خروجی **پیچیدگی زمانی ثابت** دارند.

☒ دستورات تکرار مانند for، while و do while در صورتی که تعداد تکرار مستقل از اندازه ورودی باشد، پیچیدگی زمانی ثابتی دارند، در غیر این صورت، پیچیدگی زمانی آن‌ها به تعداد تکرار بستگی دارد.

☒ دستورات تکرار متداخل پیچیدگی زمانی آن‌ها در هم ضرب می‌شود.

☒ تعاریف توابع، متغیرها و توضیحات پیچیدگی زمانی صفر دارند.

مثال ۱۲- ۱. مرتبه اجرائی دستورات زیر را بدست آورید:

```
int sum = 0;
for (int i=1; i <= n; i++)
  sum += i;
```

مرتبه اجرائی این دستورات به صورت زیر است:

چون، دستورات sum = 0 و sum += i دارای O(1) هستند و حلقه for، دارای O(n) می‌باشد. بنابراین، O(n) + O(1) = O(n) است.

مثال ۱۳ – ۱. مرتبه اجرائی دستورات زیر را بدست آورید:

مرتبه اجرائی } و { برابر O(0) است. اما، دستورات int sum = 0، int k =0 و k++ دارای O(1) هستند. چون دو حلقه (متداخل)داریم، پس $O(n^2)$ را برای حلقه‌ها خواهیم داشت. بنابراین، $O(n^2) + O(1)$ برابر با $O(n^2)$ خواهد بود.

```
int sum =0;
while (n > 0) {
  sum +=n % 10;
   n /=10;
}
```

مثال ۱۴ – ۱. پیچیدگی زمانی دستورات زیر را بدست آورید:

پیچیدگی زمانی این الگوریتم به تعداد تکرار while بستگی دارد. همان‌طور که می‌بینید، n در داخل حلقه تکرار تقسیم بر ۱۰ می‌گردد. پس اگر while، k بار اجرا شود، آنگاه n >= 10^k خواهد شد. بنابراین، تعداد تکرار این الگوریتم برابر O(k) است:

$$n \geq 10^k \Longrightarrow k = log_{10}^n \Longrightarrow O(log_{10}^n)$$

```
int sum = 0;
for (int i=1; i <= n; i++) {
  int k = 0;
  for (int j=i; j < n; j++)
    k++;
}
```

مثال ۱۵ – ۱. پیچیدگی الگوریتم زیر را تعیین کنید:

پیچیدگی دستورات int x = 0; و int i = n; برابر O(1) است. اما، پیچیدگی دستورات داخل حلقه تکرار به تعداد تکرار آن‌ها بستگی دارد. چون شرط حلقه i > 1 است. از طرف دیگر، دستورات i %= 2; نتیجه آن 0 یا 1 است. پس حلقه تکرار حداکثر یک بار اجرا خواهد شد و پیچیدگی زمانی این دستورات نیز یک است. بنابراین، پیچیدگی زمانی کل دستورات O(1) می‌باشد.

```
int x =0;
int i =n;
while(i > 1)
{
   x++;
   i %=2;
}
```

نماد Ω

نماد Ω عبارت است از: $\mathbf{f(n) \in \Omega(g(n)) \leftrightarrow \exists c, n_0 > 0, \forall_n > n_0 \quad f(n) \geq cg(n)}$.

اگر f(n) ∈ Ω(g(n)) باشد،آن گاه g(n) یک کران پایین برای f(n) خواهد بود.

قضیه: اگر $f(n) = a_m n^m + a_{m-1} n^{m-1} + \cdots + a_1 n + a_0$،آن گاه $f(n) \in \Omega(n^m)$ خواهد بود.

به عنوان مثال، در تابع $f(n) = 6n^3 + 2n^2 + 5$ داریم: $f(n) \in \Omega(n^3)$. چون، $\forall_n \geq 1: f(n) \geq 6n^3$. پس، $g(n) = n^3, c = 6, n_0 = 1$ را خواهیم داشت.

یا اگر $n^2 + n\log n$ باشد، آن گاه $f(n) \in \Omega(n^2)$ را داریم. چون، $\forall_n > 1: n^2 + n\log n \geq n^2$ است. پس، $g(n) = n^2, c = 1, n_0 = 1$ را خواهیم داشت. مثال‌های زیر را ببینید:

$f(n) = 10n^3 + 5n^2 + 6n \in \Omega(n^3)$, $f(n) = n + \log_2^n \in \Omega(n)$,
$f(n) = \log_2^n + 5 \in \Omega(\log_2^n)$, $f(n) = 5 \in O(1)$

نکته: اگر $f(n) \in \Omega(g(n))$ باشد، آن گاه مرتبه اجرایی $f(n)$ برابر تمام توابع کوچک‌تر از $g(n)$ نیز خواهد بود(مانند):

$f(n) = 14n^6 + 5n^4 + 6n^3 => f(n) \in \Omega(n^6) => f(n) \in \Omega(n^5) => f(n) \in \Omega(n^4) => \cdots$

نماد θ

نماد Θ عبارت است از: $f(n) \in \Theta(g(n)) <=> \exists c_1, c_2 n_0 > 0, \forall_n c_1 g(n) \leq f(n) \leq c_2 g(n)$.

نماد Θ تابع $f(n)$ را از بالا و پایین محدود می‌نماید. درجه رشد تابع $f(n)$ و $g(n)$ با هم برابر است.

قضیه: اگر $f(n) = a_m n^m + a_{m-1} n^{m-1} + \cdots + a_1 n + a_0$ باشد، آن گاه $f(n) \in \Theta(n^m)$ خواهد بود. به عنوان مثال، تابع $f(n) = 4n^2 + 1$ باشد، آن گاه $f(n) \in \Theta(n^2)$ خواهد بود.

چون $\forall_n > 1: 4n^2 \leq f(n) \leq 5n^2$ است. پس، $g(n) = n^2, c_2 = 5, c_1 = 4$ است.

یا تابع $f(n) = 7n^5 + n^2$ باشد، آن گاه $f(n) \in \Theta(n^5)$ خواهد بود. چون $\forall_n >: 7n^5 \leq f(n) \leq 8n^5$ است. پس، $g(n) = n^5, c_2 = 8, c_1 = 7, n_0 = 0$ است. مثال‌های زیر را ببینید:

$f(n) = 10n^3 + 5n^2 + 6n \in \Theta(n^3)$, $f(n) = n\log_2^n \in \Theta(n)$,
$f(n) = \log_2^n + 5 \in \Theta(\log_2^n)$, $f(n) = 5 \in \Theta(1)$

نکته: $f(n) \in \Omega(g(n))$، $f(n) \in O(g(n))$ است، اگر و تنها اگر $f(n) \in \Theta(g(n))$ باشد.

نکته: $f(n) \in \Theta(g(n))$ است، اگر و تنها اگر $g(n) \in \Theta(f(n))$ باشد.

نکته: $f(n) \in O(g(n))$ است، اگر و تنها اگر $g(n) \in \Omega(f(n))$ باشد.

۳-۱. توابع بازگشتی

توابع بازگشتی، توابعی هستند که خودشان را به طور مستقیم یا غیر مستقیم فراخوانی می‌کنند. معمولاً مسائل تکراری را به دو روش می‌توان حل نمود:

۱. استفاده از دستورات تکرار for، while، do while.

۲. استفاده از روش بازگشتی

معمولاً کارائی روش بازگشتی از لحاظ زمان اجرا از روش تکرار کم‌تر است. زیرا، در روش بازگشتی با هر فراخوانی تابع، سربار اضافی ایجاد خواهد شد (آدرس بازگشت و پارامترها در پشته نگهداری می‌شوند). با وجود این، در بسیاری از موارد، روش‌های بازگشتی راه حل‌های ساده‌تر و راحت‌تری نسبت به راه‌حل‌های تکراری ارائه می‌کنند. به همین دلیل، روش بازگشتی ابزار مفیدی برای حل مسائل برنامه‌نویسی محسوب

می‌شود. روش بازگشتی در حل مسائل غیر عددی نظیر نوشتن کامپایلرها، جست‌وجو و مرتب کردن در الگوریتم‌ها به کار می‌رود.

نکته:	همان‌طور که بیان گردید، الگوریتم‌های بازگشتی حافظه بیشتری مصرف می‌کنند و سرعت آن‌ها پایین‌تر است. اما، کد پیاده‌سازی آن‌ها کم‌تر است و راحت‌تر می‌توان آن‌ها را پیاده‌سازی نمود.

الگوریتم بازگشتی از دو بخش تشکیل شده است که عبارت‌اند از:

۱. **قانون بازگشتی**، فرمولی است که بر اساس آن الگوریتم بازگشتی خودش را فراخوانی می‌نماید.

۲. **شرط خاتمه**، شرطی که به فراخوانی متد بازگشتی خاتمه می‌دهد.

اکثر مسائل ریاضی را می‌توان به روش بازگشتی حل نمود. به عنوان مثال، روش بازگشتی !n به صورت زیر است:

$$n! = n * \underbrace{(n-1) * (n-2) * \dots * 2 * 1}_{(n-1)!} = n * (n-1)!$$

$$(n-1)! = (n-1) * \underbrace{(n-2) * \dots * 2 * 1}_{(n-2)!} = (n-1) * (n-2)!$$

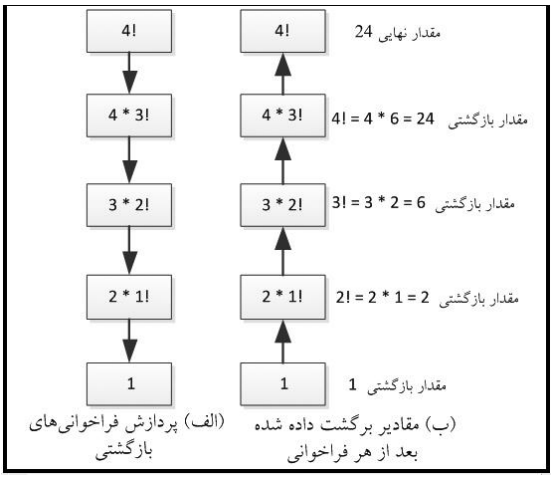
...

$$1! = 1 * 0!$$
$$0! = 1$$

اگر به این تعاریف دقت کنید، قانون بازگشتی !n برابر با $n * (n-1)!$ است و شرط خاتمه روش بازگشتی !n، اگر n برابر صفر شود، می‌باشد که یک را برمی‌گرداند. بنابراین، فرمول بازگشتی !n را می‌توان به صورت زیر نوشت:

$$n! = \begin{cases} 1 & \text{اگر } n == 0 \text{ باشد} \\ n * (n-1)! & \text{وگرنه} \end{cases}$$

روش اجرای !4 را در شکل ۲-۱ می‌بینید.



شکل ۲–۱ روش محاسبه !4 را نمایش می‌دهد.

به عنوان مثال، a و b دو عدد صحیح بزرگ‌تر از صفر باشند، فرمول محاسبه a^b به صورت زیر می‌باشد:

$$a^b = \underbrace{a * a * a * \dots * a}_{b \text{ مرتبه}} = a * \underbrace{a * a * \dots a}_{b-1 \text{ مرتبه}} = a * a^{b-1}$$

$$a^{b-1} = \underbrace{a * a * a * \dots * a}_{b-1 \text{ مرتبه}} = a * \underbrace{a * a * \dots * a}_{b-2 \text{ مرتبه}} = a * a^{b-2}$$

...

$$a^1 = a * 1 = a * a^0$$

$$a^0 = 1$$

پس، داریم:

$$a^b = \begin{cases} 1 & \text{اگر } b == 0 \text{ باشد} \\ a * a^{b-1} & \text{وگرنه} \end{cases}$$

یا فرمول بازگشتی مجموع اعداد یک تا n به صورت زیر است:

$$Sum = n + \underbrace{(n-1) + (n-2) + \dots + 1}_{Sum(n-1)} = n + Sum(n-1)$$

$$Sum(n-1) = (n-1) + \underbrace{(n-2) + (n-3) + \dots + 1}_{Sum(n-2)} = (n-1) + Sum(n-2)$$

...

$$Sum(1) = 1 + Sum(0)$$

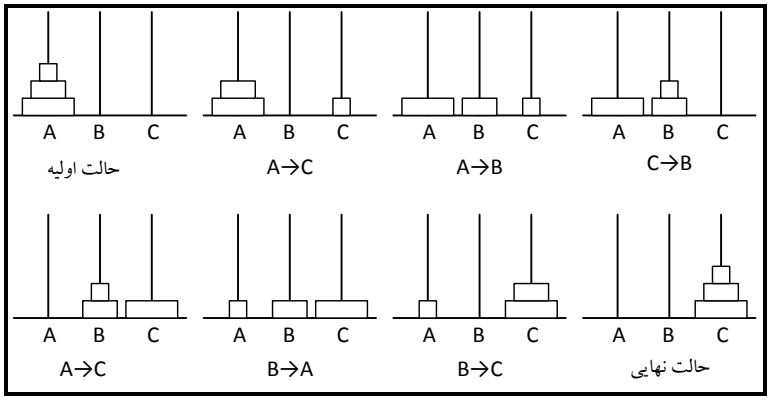
$$Sum(0) = 0$$

پس داریم:

$$Sum(n) = \begin{cases} 0 & \text{اگر } n == 0 \text{ باشد} \\ n + Sum(n-1) & \text{وگرنه} \end{cases}$$

برج‌های هانوی

یکی از مسائل معروف بازگشتی، برج‌های هانوی است. در برج‌های هانوی سه برج (میله) و n حلقه (دیسک) داریم.حلقه‌ها به ترتیب نزولی از پایین به بالای میله‌ی اول قرار دارند. هدف، انتقال حلقه‌ها از میله‌ی اول به میله‌ی سوم به کمک میله‌ی دوم است. در این جا به جایی‌ها باید دو شرط رعایت شود: ۱ - در هیچ مرحله‌ای حلقه‌ی کوچک‌تر، پایین‌تر از حلقه‌ی بزرگ‌تر قرار نگیرد. ۲- در هر مرحله، تنها یک حلقه جا به جا شود. در شکل۳-۱ جا به جایی‌های لازم برای برج‌های هانوی با سه حلقه نشان داده شده است.

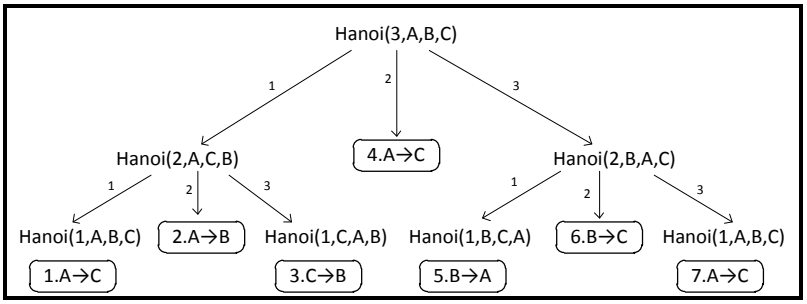


شکل۳-۱ جا به جایی‌های حلقه‌ها در برج‌های هانوی با سه حلقه.

برای حل مسئله برج‌های هانوی باn حلقه نیاز به$2^n - 1$ جا به جایی است. پس، پیچیدگی آن $O(2^n)$ است که برای nهای بزرگ بسیار وقت گیر خواهد بود. برج‌های هانوی را می‌توان با استفاده از الگوریتم زیر پیاده-سازی نمود:

```
void   Hanoi(int n, char  A,char B,char C) {
    if (n==1)
      cout<<"move a disk from" ,A ,"to",C  //  انتقال یک حلقه از مبدأ به مقصد
    else {
       Hanoi(n-1, A, C, B);        //    انتقال n-1 حلقه از مبدأ به کمکی
       cout << "move a disk from", A, "to", C;  //     انتقال یک حلقه از مبدأ به مقصد
      Hanoi(n-1,B,A,C);            //     انتقال n-1حلقه از کمکی به مقصد
     }
   }
```

در این الگوریتم n تعداد حلقه و میله‌ها بدون در نظر گرفتن نام آن‌ها، به ترتیب میله مبدأ، کمکی و مقصد هستند. همان‌طور که در الگوریتم نشان داده شده است، در هر مرحله به صورت بازگشتی سه عمل انجام می‌شود: ۱ . انتقال n-1 حلقه از مبدأ به کمکی ۲. انتقال حلقه باقی‌مانده از مبدأ به مقصد ۳. انتقال n-1 حلقه از کمکی به مقصد. در شکل۴-۱ مراحل اجرای برج‌های هانوی با سه حلقه نشان داده شده است.



شکل ٤-١ مراحل اجرای برج‌های هانوی با سه حلقه.

حل روابط بازگشتی با استفاده از روش تکرار با جای گذاری

یکی از روش‌های حل بعضی از روابط بازگشتی، روش تکرار با جای گذاری است. در این روش، با استفاده از رابطه‌ی بازگشتی و جای گذاری nهای مختلف، در نهایت جای گذاری مقدار ثابت به جواب نهایی خواهیم رسید.

مثال۱۶-۱. رابطه‌ی بازگشتی زیر را به روش تکرار با جای گذاری حل نمایید:

$$\mathbf{T(n)} = \begin{cases} \mathbf{a} & \mathbf{n = 2} \\ \mathbf{T(n-b)} & \mathbf{n > 2} \end{cases}$$

$T(n) = T(n-b)+c$

$= (T(n-b-b)+c = T(n-2b)+2c$

…

$= T(n-ib)+ic$

به فرمولی برای جمله‌ی iام رسیدیم. حال آن را برابر مقدار آستانه قرار می‌دهیم تا i به دست آید:

$$n - ib = 2 \rightarrow i = \frac{n-2}{b}$$

اکنون i را در جمله‌ی عمومی قرار می‌دهیم تا به فرمول T(n) برسیم:

$$T(n) = T(n - ib) + ic \quad \rightarrow \quad T(n) = T(2) + \frac{n-2}{b} = a + \frac{n-2}{b} \in O(n)$$

مثال۱۷-۱. رابطه‌ی بازگشتی زیر را به روش تکرار با جای گذاری حل کنید:

$$\mathbf{T(n)} = \begin{cases} \mathbf{a} & \mathbf{n = 1} \\ \mathbf{bT\left(\frac{n}{b}\right) + d} & \mathbf{n > 1} \end{cases}$$

$T(n) = bT(\frac{n}{b})+d$

$= b(bT(\frac{n}{b^2})+d = b^2\, T(\frac{n}{b^2})+(b+1)d$

$= b^2(bT(\frac{n}{b^3})+d)+(b+1)d = b^3\, T(\frac{n}{b^3})+(b^2+b+1)d$

…

$$= b^i T(\frac{n}{b^i})+(b^{i-1} + \cdots + b + 1)d = b^i\, T\left(\frac{n}{b^i}\right) + (b^i - 1)d$$

$$\frac{n}{b^i} = 1 \rightarrow n = b^i \rightarrow i = \log_b^n$$

i را در جمله عمومی قرار می‌دهیم:

$$T(n) = b^i\, T\left(\frac{n}{b^i}\right) + (b^i - 1)d$$

$$\Rightarrow T(n) = b^{\log_b^n}\, T(1) + \left(b^{\log_b^n} - 1\right)d$$

$$\Rightarrow T(n) = an + (n-1)d$$

$$\Rightarrow T(n) \in O(n)$$

مثال۱۸–۱. رابطه بازگشتی زیر را به روش تکرار با جای گذاری حل نمایید:

$$T(n) = \begin{cases} a & n = 2 \\ b \quad T\left(\frac{n}{b}\right) + nd & n > 1 \end{cases}$$

$$T(n) = b\,T\left(\frac{n}{b}\right) + nd$$

$$= b\left(b\,T\left(\frac{n}{b^2}\right) + \frac{n}{b}d\right) + nd = b^2\,T\left(\frac{n}{b^2}\right) + 2nd$$

$$= b^2\left(b\,T\left(\frac{n}{b^3}\right) + \frac{n}{b^2}d\right) + 2nd = b^3\,T\left(\frac{n}{b^2}\right) + 3nd$$

$$= b^i\,T\left(\frac{n}{b^i}\right) + ind$$

$$\rightarrow \frac{n}{b^i} = 1 \rightarrow n = b^i \rightarrow i = \log_b^n$$

$$\rightarrow \mathbf{T(n) = b^i T\left(\frac{n}{b^i}\right) + ind \rightarrow T(n) = n\,T(1) + (\log_b^n)nd}$$

$$\mathbf{=> T(n) = an + dn(\log_b^n) \rightarrow T(n) \epsilon O(n \log n)}$$

مثال۱۹–۱. تابع زمانی و پیچیدگی زمانی تابع زیر را محاسبه نمایید.

```
int func(int  n) {
  if (n==1)    return 1;
  else return  n+func(n-1);
}
```

رابطه بازگشتی این تابع به صورت زیر است:

$$\mathbf{T(n) = \begin{cases} 1 & n = 1 \\ n + T(n-1) & n > 1 \end{cases}}$$

$$\mathbf{T(n) = T(n-1) + n}$$

$$\mathbf{= T\big((n-1) + n - 1\big) + n = T(n-1) + n + (n-1)}$$

$$\mathbf{= T\big((n-2) + n - 2\big) + n + (n-1) = T(n-1) + n + (n-1) + (n-2)}$$

$$\mathbf{\cdots}$$

$$\mathbf{T\big((n-(n-1))\big) + n + (n-1) + \cdots + (n-(n-1))}$$

$$\mathbf{T(1) + n + n + (n-1) + \cdots + 1}$$

$$\mathbf{T(n) = \frac{n(n+1)}{2} + 1 \rightarrow T(n)\,\epsilon\,O(n^2)}$$

۱–۴. مسائل حل شده

مثال ۱. تابع جست‌وجوی یک مقدار در یک آرایه نامرتب به طول n عنصر در زیر آمده است. پیچیدگی زمانی این تابع را در بهترین حالت، بدترین حالت و حالت میانگین بدست آورید. میزان حافظه مصرفی را نیز بدست آورید.

```
int IndexOf(int a[], const int n, int value)
{
 int i;
 for (int i=0; i< n; i++)
   if (a[i] == value) return i;
 return – 1;
}
```

بهترین حالت زمانی اتفاق می‌افتد که اولین عنصر آرایه با مقدار value برابر باشد. پس، پیچیدگی زمانی آن O(1) است. چون فقط یک مقایسه انجام می‌شود. اما، بدترین حالت زمانی رخ می‌دهد که مقدار value آخرین عنصر آرایه باشد یا مقدار value در آرایه موجود نباشد. در این حالت n مقایسه انجام خواهد شد. پس، پیچیدگی زمانی آن O(n) است. ولی، اگر value در وسط آرایه باشد، حالت میانگین اتفاق می‌افتد. در این حالت n / 2 مقایسه انجام می‌شود. پس، پیچیدگی زمانی آن O(n / 2) خواهد شد. پیچیدگی زمانی O(n / 2) همان پیچیدگی O(n) است.

پیچیدگی حافظه به صورت زیر محاسبه می‌شود:

۱. متغیرهای n، value و i قسمت ثابت حافظه مصرفی هستند که دارای پیچیدگی $S(1) = O(1)$ هستند.

۲. آرایه a قسمت متغیر حافظه مصرفی است که دارای طول n می‌باشد. پس، پیچیدگی حافظه آن $S(n) = O(n)$ می‌باشد.

مثال ۲. پیچیدگی زمانی و مقدار حافظه مصرفی تابع Reverse() زیر که عدد n را به عنوان پارامتر دریافت می‌کند و مقلوب آن را برمی‌گرداند، بدست آورید:

```
int Reverse(int n)
{
 int r =0;
 while (n > 0)
 {
  r =r * 10 + n % 10;
  n /=10;
 }
 return r;
}
```

پیچیدگی زمانی دستورات int r = 0; و r = r * 10 + n % 10; و n /= 10;، برابر $O(1)$ است. پس پیچیدگی زمانی این تابع به تعداد تکرار دستورات while بستگی دارد. در داخل while، هر بار n بر ۱۰ تقسیم می‌گردد (پس در بدترین حالت، حلقه while به تعداد ارقام n تکرار می‌شود). پس در بدترین حالت $n >= 10^m$ است که m تعداد تکرار while است. بنابراین، پیچیدگی زمانی تابع برابر با $O(log_{10}^{n})$ است. در این تابع متغیرهای ثابت n و r استفاده شده‌اند. پس پیچیدگی حافظه آن $S(1) = O(1)$ می‌باشد.

مثال ۳. پیچیدگی زمانی و میزان حافظه مصرفی تابع مرتب سازی bubbleSort() زیر را بدست آورید:

```
void bubbleSort(int a[], const int n)
{
  int i, j, temp;
  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;
     }
}
```

پیچیدگی زمانی این تابع برابر با تعداد تکرار حلقه‌های for است. دستور for خارجی دقیقاً $n - 1$ بار تکرار می‌شود. حلقه for داخلی نیز در بدترین حالت $n - 1$ مرتبه تکرار خواهد شد. پس پیچیدگی زمانی آن برابر با:

$$T(n) = O((n-1)(n-1)) = O(n^2 - n - n + 1)$$
$$= O(n^2) - O(2n) + O(1) = O(n^2)$$

میزان حافظه مصرفی این تابع از دو بخش زیر تشکیل شده است:

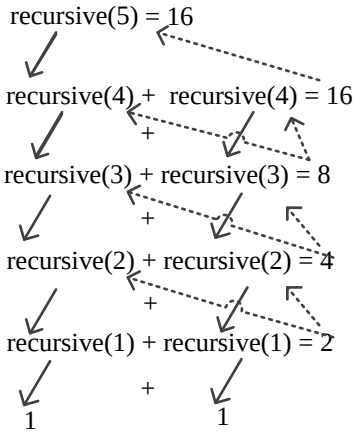
۱. بخش ثابت حافظه مصرفی، شامل متغیرهای n، i، j و temp می‌باشد که پیچیدگی آن‌ها $S(1) = O(1)$ است.

۲. بخش متغیر حافظه مصرفی، شامل آرایه a است که دارای n عنصر می‌باشد. پس، پیچیدگی حافظه آن $S(n) = O(n)$ می‌باشد.

مثال ۴. مقدار recursive(5) تابع بازگشتی زیر را بدست آورید:

```
int recursive(int n) {
   if (n == 1) return (1);
   else return (recursive(n – 1) + recursive(n – 1));
}
```

نمودار درختی اجرای این تابع به صورت زیر است:



مثال ۵. فرض کنید n توانی از ۲ باشد، تابع زیر چه عددی را به عنوان جواب بر می‌گرداند:

```
int    test(int n)   {
  if (n == 1) return (1);
  return( 2 * test(n / 2) + 6 * n – 1);
}
```

$$T(n) = 2T\left(n/2\right) + 6n - 1, \quad T(n) = 6n\log_2^n + 1$$

$$T(n) = 2\left[\left(6\frac{n}{2}\right)\log_2\frac{n}{2} + 1\right] + 6n - 1$$

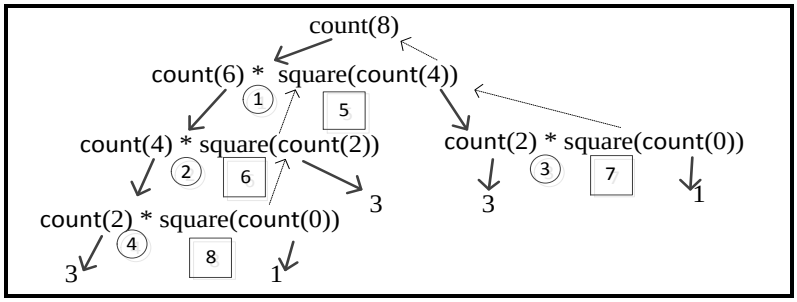
$$T(n) = 6n\left(\log_2^{n-1}\right) + 2 + 6n - 1$$

$$T(n) = 6n\log_2^n - 6n + 2 + 6n - 1 = 6n\log_2^n + 1$$

مثال ۶. در فراخوانی تابع زیر برای n = 8 چند عمل ضرب انجام می‌شود؟ فرض کنید هر عمل square نیز به یک عمل ضرب نیاز دارد.

```
int count(int n)   {
  if (n <= 0) return(1);
  if (n == 1) return(2);
  if (n == 2) return(3);
  return (count(n – 2) * square(count(n – 4)));
}
```

درختی بازگشتی به صورت زیر است:

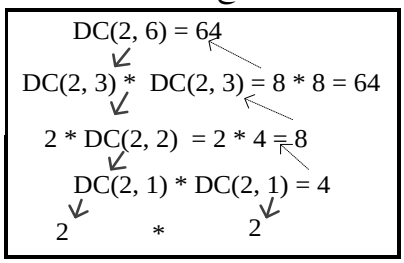


همان‌طور که در شکل می‌بینید، ۴ عمل ضرب انجام شده است(شماره‌های موجود در دایره) و چهار مرتبه square تکرار شده است(شماره‌های موجود در مربع). از آن جایی که هر square یک عمل ضرب می‌باشد. بنابراین، ۸ عمل ضرب انجام شده است.

مثال ۷. تابع زیر برای مقادیر a = 2 و n = 6 چه مقداری را برمی‌گرداند:

```
int Dc(int a, int n)
{
 if (n == 1) return(a);
 if (n % 2 == 0) return(DC(a, n / 2) * DC(a, n/2));
 return (a * DC(a, n – 1));
}
```

درخت بازگشتی تابع به صورت زیر است:



مثال ۸. تابع زیر را در نظر بگیرید:

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

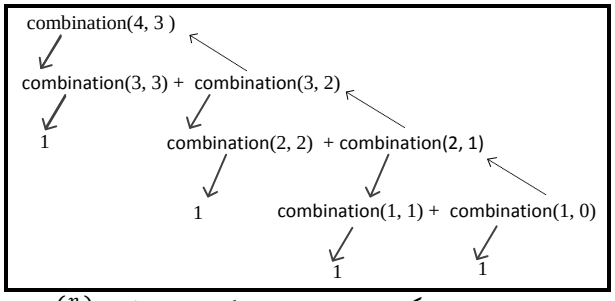
برای n > m دلخواه عمل + چند بار اجرا می‌شود.

فرمول بازگشتی این تابع به صورت زیر است:

$$\binom{n}{0} = \binom{n}{n} = 1 \qquad \text{اگر } m == n \text{ یا } m == 0 \text{ باشد،}$$

$$\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1} \qquad \text{وگرنه}$$

بنابراین این تابع را به مسائل کوچک‌تری تقسیم می‌کنیم تا به حالتی برسیم که m == n شود یا m == 0 شود. در این حالت تابع ۱ را برمی‌گرداند، در نهایت یک‌های بدست آمده را جمع می‌کنیم. به عنوان مثال، اگر n=4 و m=3 باشد، درخت بازگشتی به صورت زیر بدست می‌آید:



همان طور که مشاهده می‌شود، مجموع یک‌ها برابر با ۴ است که برابر با $\binom{n}{m} - 1$ است.

مثال ۹. تابع زیر را در نظر بگیرید. تعداد فراخوانی‌های بازگشتی چقدر است؟

```
int C(int n, int k)  {
    if ((k == 0)|| (k == n) return(1);
     else return (C(n – 1, k-1)+ C(n–1, k));
}
```

این مثال مانند مثال ۸ است و به همان روش حل می‌گردد.

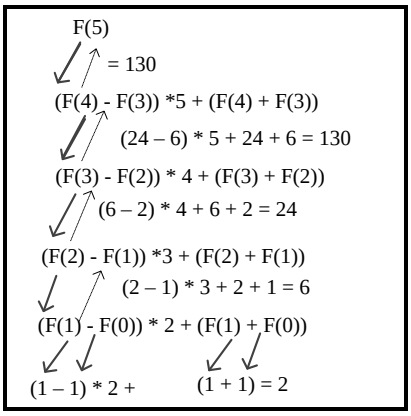
مثال ۱۰. مقدار بازگشتی F(5) چیست؟

فرمول تابع به صورت زیر می‌باشد:

```
long F(int n)   {
   if (n < 2) return(1);
   else return ((F(n – 1) - F(n - 2))*n +(F(n–1)+ F(n-2)));
}
```

$$F(n)\begin{cases}1 & \text{اگر } n<2\\ ((F(n-1)-F(n-2))*n+(F(n-1)+F(n-2)) & \text{وگرنه}\end{cases}$$

اکنون برای $f(5)$ درخت فراخوانی‌های بازگشتی را رسم می‌کنیم:

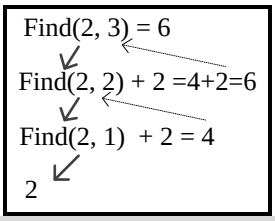


پس، مقدار بازگشتی ۱۳۰ است.

مثال ۱۱. تابع زیر را در نظر بگیرید. اگر a و b اعداد صحیح باشند، تابع چه مقداری را برمی‌گرداند؟

```
int    Find(int a, int b))   {
   if (b == 1) return(a);
   else return (Find(a, b – 1) + a);
}
```

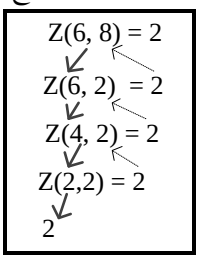
این تابع همان a * b (a را b بار با خودش جمع می‌کند) است. یعنی، اگر a = 2 و b = 3 باشد، درخت بازگشتی تابع به صورت زیر است:



مثال ۱۲. تابع زیر با مقدار n = 8 و k = 6 چه مقداری را برمی‌گرداند؟

```
int    Z(int k, int n)
{
 if (n == k)   return(k);
 else if (n> k)   return (Z(k, n – k));
 else return Z(k – n, n);
}
```

درخت بازگشتی تابع به صورت زیر است:



مثال۱۳. در قطعه برنامه زیر مقدار محاسبه شده برای total را تعیین کنید:

```
int  total =0, k, i;
for (i=1; i<= n; i++)  {
  k=n;
  while (k != 1)  {
    k /=2;
    total++;
  }
}
```

مقدار total برابر است با تعداد دفعاتی که دستور total++; اجرا می‌شود. دستور total++; در داخل حلقه while قرار دارد که این حلقه $\log_2^k$ (همان $\log_2^n$، چون k در ابتدای حلقه while برابر n است) مرتبه اجرا می‌شود. چون، این حلقه در داخل حلقه for قرار دارد و حلقه for، به تعداد n مرتبه اجرا می‌شود، پس دستور total++; به تعداد $n * \log_2^n$ مرتبه اجرا خواهد شد. از آنجائی که هر مرتبه به total یک واحد اضافه می‌شود، پس مقدار total برابر با $n * \log_2^n$ خواهد شد.

مثال۱۴. در قطعه برنامه زیر t++ چند بار اجرا می‌شود:

```
for (i=1; i<= m; i++)
   for (j=1; j <= (m –i); j++)    t++;
```

تعداد اجرای $t++$ برابر با تعداد تکرار حلقه داخلی است و حلقه داخلی به صورت زیر تکرار می‌شود:

i	1	2	…	m
تعداد تکرار حلقه داخلی	m – 1	m – 2		0

$$(\boldsymbol{m-1})+(\boldsymbol{m-2})+(\boldsymbol{m-3})+(\boldsymbol{m-4})+\ \ldots \boldsymbol{2+1}=\frac{(\boldsymbol{m-1})\times(\boldsymbol{m-1+1})}{\boldsymbol{2}}=\frac{(\boldsymbol{m-1})\boldsymbol{m}}{\boldsymbol{2}}$$

مثال۱۵. برای m > 0، پس از اجرای قطعه برنامه زیر، مقدار x چند خواهد شد؟

```
int  x=0;
for (i=0; i <= m; i++)
  for (j=0; j <= i; j++)
    for (k=1; k <= m; k++) x++;
```

مقدار x برابر با تعداد اجرای x++ است. از آن جایی که x++، در حلقه داخلی قرار دارد، بنابراین تعداد تکرار آن برابر است با:

(تعدد تکرار حلقه با شمارنده k) * (تعداد تکرار حلقه با شمارنده j)

حلقه با شمارنده j، برای i = 0، یک مرتبه اجرا می‌شود، برای i=1، دو مرتبه اجرا خواهد شد و برای i=m، m+1 مرتبه اجرا می‌گردد. پس، تعداد تکرار حلقه با شمارنده j به صورت زیر حساب می‌شود:

$$1+\ \ 2+\cdots m+(m+1)=\frac{(m+1)(m+2)}{2}$$

چون، حلقه با شمارنده k نیز m مرتبه اجرا می‌شود. پس تعداد تکرار دستور x++ برابر است با:

$$\frac{(m+1)(m+2)m}{2}$$

مثال۱۶. درجه قطعه برنامه زیر چند است؟

```
for (i=0; i<= n; i++)  {
  j=i;
  while (j != 0)
    j /=2;
}
```

چون قطعه برنامه از دو حلقه متداخل for و while مستقل تشکیل شده است، مرتبه اجرائی هر یک از حلقه‌ها را بدست می‌آوریم و در هم ضرب می‌کنیم. یعنی، داریم:

مرتبه اجرائی $for(n+1)$ بار اجرا می‌شود) $= O(n+1) = O(n) + O(1) = O(n)$

مرتبه اجرائی $while = O(log_2^n)$

چون، j در بدترین حالت برابر n است و هر بار j بر ۲ تقسیم می‌گردد. پس مرتبه اجرائی کل برنامه برابر با $O(nlog_2^n)$ می‌باشد.

مثال۱۷. مرتبه زمانی قطعه برنامه زیر چیست؟

```
j=n;
while (j >= 1)    {
   for (i=1; i<= n; i++)
        x++;
   j /= 2;
}
```

مرتبه اجرائی این قطعه برنامه نیز برابر با مرتبه اجرائی while ضرب در مرتبه اجرای for می‌باشد. مرتبه اجرائی while در بدترین حالت برابر با $O(log_2^n)$ است و مرتبه اجرائی for برابر با O(n) می‌باشد. بنابراین، مرتبه اجرائی این قطعه برنامه برابر با $O(nlog_2^n)$ است.

مثال۱۸. پیچیدگی زمانی دستورات زیر را بدست آورید:

```
int a=n;
while (a >1)   {
 a /=2;
 b=n;
 while (b > 1)  {
  b /=3;
  x++;
  }
}
```

پیچیدگی زمانی این دستورات برابر با پیچیدگی زمانی while (a > 1) * پیچیدگی زمانی while (b > 1) است. چون در داخل حلقه داخلی هر مرتبه b تقسیم بر ۳ می‌شود، پیچیدگی زمانی الگوریتم آن $O(log_3^n)$ می‌باشد و در حلقه while (a >1) چون، هر مرتبه a تقسیم بر ۲ می‌شود، پیچیدگی آن $O(log_2^n)$ خواهد شد. پس، پیچیدگی زمانی کل برابر با $O(log_2^n * log_3^n)$ می‌باشد.

مثال ۱۹. ثابت کنید عبارت زیر برقرار است:

$$7n^2 - 6n + 2 \in \theta(n^2)$$

به ازای $\forall n \geq n_0$ به طوری که n_0، c_2 و $\exists c_1$ داریم: $C_2 n^2 < 7n^2 - 6n + 2 < C_1 n^2$

این عبارت را تقسیم بر n^2 می‌کنیم تا عبارت زیر به دست آید:

$$C_2 < 7 - \frac{6}{n} + \frac{2}{n^2} < C_1$$

در این رابطه به ازای $n_0 = 3$، $C_1 = 9$ و $C_2 = 6$ رابطه مقابل را داریم: $T(n) \in T(n^2)$

مثال ۲۰. تعیین کنید آیا عبارت زیر برقرار است یا خیر؟

$$n^5 + 14n^3 \in \theta(n^3)$$

با توجه به تعریف θ، $\exists c_1, c_2, n_0 \geq 0, \forall n > n_0 \Rightarrow c_1 n^3 \leq n^5 + 14n^3 \leq c_2 n^3$ برقرار است. چون هیچ گاه رابطه $n^5 + 14n^3 \leq c_2 n^3$ برقرار نیست؛ زیرا، اگر طرفین را بر n^3 تقسیم کنیم، رابطه زیر بدست می‌آید:

$$n^2 + 14 \leq c_2 \Rightarrow n^2 \leq c_2 - 14$$

که هیچ گاه رابطه $n^2 \leq c_2 - 14$ برقرار نیست. پس، رابطه $T(n) \in \theta(n^3)$ نادرست خواهد بود.

مثال ۲۱. ثابت کنید عبارت زیر برقرار است:

$$n! + 7n^5 \in O(n^n) \Rightarrow T(n) = n! + 7n^5$$

رابطه $\exists C, n_0 \geq 0, \forall n > n_0 \Rightarrow T(n) \leq Cn^n$ برقرار است. پس، داریم:

$$T(n) = n! + 7n^5 \leq n^n + 7n^5 \leq n^n + 7n^n = 8n^n$$

پس به ازای C = 8 رابطه $T(n) \in O(n^n)$ برقرار خواهد بود.

مثال ۲۲. تابع بازگشتی که n را به عنوان پارامتر دریافت کرده، مجموع ارقام آن را برمی‌گرداند.

```
int sumDigit(int n) {
   if (n == 0) return 0;
   else return(n % 10 + sumDigit(n / 10));
}
```

مثال ۲۳. تابع بازگشتی که n را به عنوان پارامتر دریافت کرده، حاصل ضرب ارقام فرد آن را برمی‌گرداند.

```
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));
}
```

مثال ۲۴. تابع بازگشتی که n را به عنوان پارامتر دریافت کرده، حاصل جمع ارقام بالای ٦ آن را برمی‌گرداند.

```
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));
}
```

مثال ۲۵. تابع بازگشتی که n را به عنوان پارامتر دریافت کرده، nامین جمله فیبوناچی را برمی‌گرداند.

```
int Fibo(int n) {
   if (n == 1 || n == 2) return 1;
   else return(Fibo(n -1) + Fibo(n – 2));
}
```

مثال ۲۶. تابع بازگشتی که m و n را به عنوان پارامتر دریافت کرده، $\binom{n}{m} = C(n,\ m)$ را برمی‌گرداند.

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

مثال ۲۷. تابع بازگشتی که a و n را به عنوان پارامتر دریافت کرده، حاصل عبارت $\sqrt{a+\sqrt{a+\sqrt{a+\cdots}}}$ را برای n مرتبه برمی‌گرداند.

```
double S(double a, int n) {
   if (n == 1) return(sqrt(a));
   else return(sqrt(a + S(a, n -1));
}
```

مثال ۲۸. تابع بازگشتی که a و b را به عنوان پارامتر دریافت کرده، a تقسیم بر b را برمی‌گرداند.

```
int Div(int a, int b) {
   if (a < b) return 0;
   else return(1 + Div(a – b, b));
}
```

مثال۲۹. تابع بازگشتی که n را به عنوان پارامتر دریافت کرده، حاصل جمع اعداد فرد ۱ تا n را برمی‌گرداند.

```
int sumOdd (int n) {
   if (n < 1) return 0;
   else if (n % 2 == 1) return(n + sumOdd(n – 2));
  else return(sumOdd(n - 1));
}
```

مثال ۳۰. تابع بازگشتی که n را به عنوان پارامتر دریافت کرده، حاصل ضرب اعداد زوج ۲ تا n را برمی‌گرداند.

```
long MulEven(int n) {
   if (n < 2) return 1;
```

```
    else if (n % 2 == 0) return(n * MulEven(n – 2));
    else return(MulEven(n - 1));
}
```

مثال ۳۱. تابع بازگشتی که اعداد صحیح a و b بزرگ‌تر از صفر را دریافت کرده، a ضرب در b را برمی‌گرداند.

```
int Mul(int a, int b) {
   if (b == 1) return a;
   else return(a + Mul(a, b – 1));
}
```

مثال ۳۲. تابع بازگشتی که a و b (اعداد صحیح) را به عنوان پارامتر دریافت کرده، باقی مانده تقسیم صحیح a بر b را برمی‌گرداند.

```
int Mod(int a, int b) {
   if (a < b) return a;
   else return(Mod(a – b, b));
}
```

مثال ۳۳. تابع بازگشتی که n را به عنوان پارامتر دریافت کرده، مجموع تمام اعداد مضرب ۳ کوچک‌تر یا مساوی n را برمی‌گرداند.

```
int M3(int n) {
  if (n < 3) return 0;
  else if (n % 3 == 0) return(n + M3(n – 3));
  else return(M3(n – 1));
}
```

مثال ۳۴. تابع بازگشتی که n را به عنوان پارامتر دریافت کرده، تعداد صفرهای آن را برمی‌گرداند.

```
int countZero(int n) {
  if (n == 0) return 0;
  else if ( n % 10 == 0) return(1 + countZero(n / 10));
  else return(countZero(n / 10));
}
```

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 $C(^n_m)$ 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 $n <= 2^L$ 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. کدام یک از مجموعه توابع زیر بر حسب افزایش مرتبه (order) از چپ به راست مرتب هستند؟ (علوم کامپیوتر- دولتی ۷۹).

الف: !n، n^{1000}، $(1.005)^n$ ب: !n، $(1.005)^n$، n^{1000}
ج: n^{1000}، !n، $(1.005)^n$ د: $(1.005)^n$، !n، n^{1000}

۸. در زیر برنامه زیر مقدار F(6, 3) برابر است با: (مهندسی کامپیوتر- آزاد ۸٤).

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

الف: ۲۰ ب: ۱۰
ج: ۱۸ د: ۱۵

۹. فرض کنید که a و b اعداد صحیح مثبت بوده و تابع Q به صورت زیر به شکل بازگشتی تعریف شده باشد: مقادیر Q(2, 3) و Q(14, 3) را پیدا کنید (مهندسی کامپیوتر – آزاد ۷۸).

$$Q(a,b) = \begin{cases} 0 & if\ a < b \\ Q(a-b,b)+1 & if\ b \le a \end{cases}$$

الف: Q(14, 3) = 0، Q(2, 3) = 0 ب: Q(2, 3)= 0, Q(14, 3) = 4
ج: Q(2, 3) = 0, Q(14, 3) = 3 د: Q(2, 3) = 4, Q (14, 3) = 3

۱۰. کدام گزینه صحیح است؟ (علوم کامپیوتر- دولتی ۸۲).

الف: $n^a = O(logn)^b$ اگر a > b ب: $n^a \neq O((logn)^b)$ اگر a > b
ج: $n^a = O((logn)^b)\ \forall b,\ a > 0$ د: $n^a \neq O((logn)^b)\ \forall b,\ a > 0$

۱۱. مرتبه زمانی الگوریتم زیر چیست؟ (علوم کامپیوتر- دولتی ۸۵).

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

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

۱۲. کدام یک از تساوی‌های زیر درست است؟ (مهندسی کامپیوتر – آزاد ۸۵).

الف: $n^2 logn = \theta(n^2)$ ب: $6n^3/(logn+1) = O(n^3)$

ج: $n^2/logn = \theta(n^2)$ د: $n^3 2^n + 6n^2 3^n = O(n^3 2^n)$

۱۳. گزینه صحیح را انتخاب کنید (علوم کامپیوتر- دولتی ۸۰).

الف: $n^{\frac{1}{10}} \in \Omega(logn)$ ب: $\log(n!) \in O(\log(n))$

ج: $8n^2 + 3n - 4 \in O(nlogn)$ د: $1^n + 2^n + \cdots + n^n \in O(n^n)$

۱۴. تابع بازگشتی زیر را در نظر بگیرید: (مهندسی کامپیوتر- دولتی ۷۵).

```
int test (int n) {
  if (n <= 2) return 1;
  else return test(n – 2)*test(n – 2);
}
```

زمان اجرای تابع فوق برابر است با:

الف: $O(n^2)$ ب: $O(nlogn)$

ج: $O(2^{n/2})$ د: $O(2^n)$

۱۵. تابع $f(n)=4^{\log n}$، $g(n) = (logn)^{\log n}$ و $h(n) = log^2 n$ را در نظر بگیرید. کدام یک از گزاره‌های زیر صحیح است؟ (مهندسی کامپیوتر- دولتی ۸۵).

الف: $f(n) \in O(g(n)), f(n) \in \Omega(h(n))$ ب: $g(n) \in \Omega(h(n)), h(n) \in \Omega(f(n))$

ج: $f(n) \in O(h(n)), g(n) \in \Omega(f(n))$ د: $h(n) \in O(g(n)), f(n) \in \theta(g(n))$

۱۶. کدام یک از عبارات زیر غلط است (مهندسی کامپیوتر- دولتی ۸۱).

الف: $10^n + n^{20} \notin \theta(n^n)$ ب: $log_2^n \in \theta(log_{10}^n)$

ج: $(log_2^n)! \in \Omega(n!)$ د: $4n^3 + 5n^2 + 7n \in \Omega(log_2^n)$

۱۷. می‌خواهیم n خط را در یک صفحه رسم کنیم. با فرض این که هیچ دو خطی موازی هم دیگر نیستند و همچنین بیشتر از دو خط هم دیگر را در یک نقطه قطع نمی‌کنند، تعداد نواحی تولید شده توسط این خطوط چیست؟ (مهندسی کامپیوتر- آزاد ۷۹).

الف: $2n+1$ ب: $\frac{n(n+1)}{2} + 1$ ج: $3n-1$ د: n^2

۱۸. پیچیدگی زمانی الگوریتم زیر کدام است (علوم کامپیوتر- دولتی ۸۱).

```
sum = 0;
for (i = 0; i < n; i++)
  for (j = 0; j < i; j ++)
    for (k = 0; k < 3; k ++)    sum++;
```

الف: $O(n^3)$ ب: $O(n)$

ج: $O(nlogn)$ د: $O(n^2)$

۱۹. مرتبه زمانی شبه کد زیر چیست (مهندسی فناوری اطلاعات- دولتی ۸٤).

```
for (i = 1; i <= n; i++)  {
   for (j = 1; j <= n; j ++)
      x--;
   n --;
}
```

الف: n ب: n^2

ج: logn د: nlogn

۲۰. کدام یک از عبارات زیر صحیح است:(علوم کامپیوتر- دولتی ۸۱).

الف: $\sum_{i=0}^{n} i^2 = O(n^3)$ ب: $3^n = O(2^n)$

ج: $n^2 logn = O(n^2)$ د: $n^2/logn = O(n^2)$

۲۱. n دانشجو به صورت تصادفی در یک ردیف نشسته‌اند. فرض کنید دانشجوی i در مختصات (X_i, 0) قرار دارد. مربی می‌خواهد در نقطه‌ای باشد که مجموع فواصل‌اش تا همه دانشجویان کمینه شود. با داشتن n مختصات، محل قرار گرفتن مربی را در چه زمانی می‌توان بدست آورد؟ (مهندسی کامپیوتر- دولتی ۸۳).

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

۲۲. تابع زیر را در نظر بگیرید: (مهندسی کامپیوتر- دولتی ۸۱).

کدام گزینه بهترین نمایش برای تابع F(n) بر حسب n است؟

الف: خطی ب: نمایی

ج: درجه ۲ د: لگاریتمی

```
int f(int x)  {
   if(x < 1) return 1;
  else return f(x – 1) + g(x);  }
 int g(int x)  {
   if (x < 2) return 1;
   else return f(x – 1) + g (x / 2);
  }
```

```
int Fun2 (int n)  {
   if (n <= 1) return 1;
   else return Fun2(n/2) + n/2;
}
```

۲۳. تابع بازگشتی زیر را در نظر بگیرید (n توانی از ۲ است): (مهندسی فناوری اطلاعات - آزاد ۸۵). زمان اجرای تابع فوق چیست؟

الف: $O(log_2^n)$ ب: $O(n)$

ج: $O(nlogn)$ د: $O\left(\frac{n}{2}\right)+1$

۲۴. کدام یک از روابط ذیل درست است؟ (مهندسی کامپیوتر- آزاد ۸۱).

الف: $O(\log n) < O(\sqrt{n})$ ب: $O(\log n) > O(\sqrt{n})$

ج: $O(\mathrm{n!}) < O(a^n)$ د: $O(\sqrt{\mathrm{n}^3}) < O(n)$

۲۵. کدام گزینه صحیح می‌باشد؟ (علوم کامپیوتر – دولتی ۸٤).

الف: $\mathrm{n}^3 logn = O(n^{3+\varepsilon}) \quad 0 < \varepsilon < 0.1$ ب: $\sqrt{\mathrm{n}} = O(logn)$

ج: $n^{1+\varepsilon} = O(nlogn) \quad 0 < \varepsilon < 0.1$ د: $\mathrm{n}^2 = O(\frac{n^2}{logn})$

۲۶. خروجی برنامه زیر برای دو عدد صحیح مثبت x و y چیست؟ (علوم کامپیوتر-دولتی ۸٤).

```
int test (int x, int y) {
  if ( x ==  0) return y;
  else return (--x, y++);
}
```

الف: x ب: y

ج: x – y د: x + y

۲۷. فرض کنید a و b نمایش دو عدد صحیح مثبت باشند. فرض کنید تابع Q به شکل زیر به صورت بازگشتی تعریف شده است (مهندسی کامپیوتر-آزاد ۸۲):

$$Q(a,\ b) = \begin{cases} 0 & if(a < b) \\ Q(a-b,\ b)+1 & if\ (b \le a) \end{cases}$$

مقدار Q(5861, 7) برابر است با:

الف: 837 ب: 843 ج: 850 د: 858

۲۸. در تکه برنامه زیر Process a چند بار اجرا می‌شود؟ (علوم کامپیوتر- دولتی ۸٤).

```
for (int i = 0; i< N –i; i++)
   for (int j = 0; j < N –i; j++) {
    /*process a */
 }
```

الف: $\frac{N}{2}$ ب: $\frac{N^2}{4}$

ج: $\frac{N+1}{2}$ د: $\frac{(N+1)^2}{4}$

۲۹. در برنامه زیر تعداد دفعات تکرار دستورالعمل شماره ۳ برابر است با ... (علوم کامپیوتر- دولتی ۸۰).

```
1) for (int k = 0; k <= n; k++)
2)   for (i = 1; i <= n - k; i++)
3)     a[i][i + k] = k;
```

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

۳۰. کدام گزینه تعداد مراحل برنامه زیر را به درستی بیان می‌کند؟ (مهندسی کامپیوتر- دولتی ۸۳).

```
void sum (int m, int n, folat s[][]) {
  int i, j;
  for (j = 0; j < m; j++)    {
    s[n–1][j] =0;
    for (i =0; i < n; i++)
   s[n–1][j] += s[i][j];
    }
}
```

الف: 2m + 2n
ب: $mn^2 + m^2n$
ج: m(2n + 1) + 1
د: m (2n + 1) – 1

۳۱. برای محاسبه عبارت زیر، حداقل چند عمل ضرب می‌توان استفاده کرد؟ (مهندسی کامپیوتر- دولتی ۸۱).

$$a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0$$

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

۳۲. کدام عبارت صحیح است؟ (علوم کامپیوتر- دولتی ۸۲).

الف: $(n+1)(n^2-2n+1) \in O(2^n)$ ب: $(n+1)(n^2-2n+1) \in \theta(n)$
ج: $(n+1)(n^2-2n+1) \in \Omega(n^4)$ د: $(n+1)(n^2-2n+1) \in O(n^2 logn)$

۳۳. پیچیدگی زمانی اجرای حلقه زیر چه می‌باشد؟ (علوم کامپیوتر- دولتی ۸٤).

```
while (n > 0)  {
  n = n /  10;
}
```

الف: $O(1)$ ب: $O(\frac{n}{10})$
ج: $O(n)$ د: $O(\log n)$

۳۴. میزان زمان لازم برای اجرای قطعه برنامه زیر چگونه برآورد می‌شود؟ (مهندسی فناوری اطلاعات- دولتی ۸۳).

```
for i ← 1 to n
  for j ← 1 to i
    for k ← 1 to n²
      Sum← sum + A
```

الف: $O(n^3)$ ب: $\theta(n^3)$
ج: $\Omega(n^3)$ د: $\theta(n^4)$

۳۵. فرض کنید زمان اجرایی الگوریتمی روی n ورودی، T(n) بوده که به صورت زیر تعریف می‌شود. زمان اجرای الگوریتم مزبور برابر کدام گزینه است؟ (مهندسی کامپیوتر- دولتی ۸٤).

$$T(n) = \begin{cases} 1 & n = 1 \\ n + T(n-1) & n \geq 2 \end{cases}$$

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

۳۶. مرتبه اجرایی تابع زیر کدام است (مهندسی کامپیوتر – دولتی ۸۰).

```
int gcd(int a, int b)  {
  if (b == 0)  return a;
  else return gcd (b, a % b);
}
```

الف: $O(log_2^{\frac{a}{b}})$ ب: $O(log_b^a)$
ج: $O(log_2^a)$ د: $O(log_2^{(a-b)})$

۳۷. بهترین الگوریتم بازگشتی برای محاسبه x^y دارای چه زمانی است؟ (علوم کامپیوتر- دولتی ۸٤، آزاد ۹۱ و ۹۲).

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