بسم ا... الرحمن الرحیم

درس و کنکور

طراحی الگوریتم

ویرایش سوم

ویژه‌ی : داوطلبان کنکورهای کارشناسی ارشد کامپیوتر

تمامی گرایش‌های سخت‌افزار، نرم‌افزار، هوش مصنوعی، علوم کامپیوتر و IT

قابل استفاده: کنکورهای دکتری (کامپیوتر، بیوانفورماتیک و ..)

- ☑ **خلاصه سه کتاب آقایان نیپولیتان، هورویتز و قلی‌زاده**
- ☑ **تدریس کامل و خودآموز درس مطابق سرفصل‌های وزارت علوم**
- ☑ **حل تشریحی تمامی کنکورهای رسمی برگزار شده دولتی و آزاد تا سال جاری**
- ☑ **حل تمامی کنکورهای سخت‌افزار، نرم‌افزار، هوش مصنوعی، علوم کامپیوتر و مهندسی IT**
- ☑ **قابل استفاده به عنوان مرجع درسی برای دوره کارشناسی**

مؤلف : حمیدرضا مقسمی

عضو هیأت علمی دانشگاه آزاد اسلامی واحد تهران غرب

سرشناسه : مقسمی، حمیدرضا، ۱۳۴۹ -
عنوان و نام پدیدآور : درس وکنکورطراحی الگوریتم(ارشد) ویژه ی : داوطلبان کنکورهای کارشناسی ارشد کامپیوتر .....
وضعیت ویراست : ویراست سوم
مشخصات نشر : تهران: گسترش علوم پایه، ۱۳۹۷.
مشخصات ظاهری : ۵۰۴ ص.: مصور
شابک : 978-964-490-628-2
وضعیت فهرست نویسی : فیپای مختصر
یادداشت : این مدرک در آدرس http://opac.nlai.ir قابل دسترسی است.
یادداشت : چاپ بیستم.
شماره کتابشناسی ملی : ۳۶۲۷۱۴۲

نام کتاب : ..... **درس و کنکور طراحی الگوریتم (ارشد)**
مؤلف : ..... **حمیدرضا مقسمی**
ناشر : ..... **انتشارات گسترش علوم پایه**
مدیر فنی : ..... **مهدی زنگنه**
حروفچینی : ..... **سیما ابویی**
طرح جلد : ..... **حاتمی‌کیا**
لیتوگرافی : ..... **مهرشاد**
چاپخانه : ..... **مهر**
سال نشر : **۱۳۹۷**
نوبت چاپ : ..... **بیستم**
شمارگان : ..... **۵۰۰ جلد**
قیمت : ..... **۳۵۰۰۰۰ ریال**

شابک : ۲-۶۲۸-۴۹۰-۹۶۴-۹۷۸ **ISBN : 978-964-490-628-2**

دفتر انتشارات و پخش شهرستان: میدان انقلاب، ابتدای کارگر جنوبی، کوچه مهدیزاده، پلاک ۹، طبقه همکف
تلفن: ۱۵ ~ ۶۶۹۰۵۳۱۲ تلفکس: ۶۶۹۰۵۳۱۶-۰۲۱
دفتر سفارشات تهران: خ آزادی، خ جمال‌زاده جنوبی، خ دیلمان، پلاک ۱۴ واحد ۲
تهران: ۱۴ ~ ۶۶۵۹۵۵۱۳
نمایندگی تهران: خ انقلاب، نبش ۱۲ فروردین ، پ ۱۴۴۴ ، کتابفروشی الیاس- ۶۶۴۰۵۰۸۴
Email:gostaresh_op@yahoo.com
www.gostaresh-pub.com

فهرست مطالب

عنوان

صفحه

تقدیم به ساحت مقدس یوسف زهرا(عج)

که چشمها برای زیارت صبحش بیدارند.....

مقدمه ویرایش سوم

طراحی الگوریتم یکی از دروس اصلی و پایه در تمامی گرایش‌های مختلف رشته کامپیوتر از جمله سخت‌افزار، نرم‌افزار، هوش مصنوعی، علوم کامپیوتر و IT می‌باشد. درس طراحی الگوریتم در واقع ادامه درس ساختمان‌داده‌ها بوده و ارتباط نزدیکی با آن دارد.

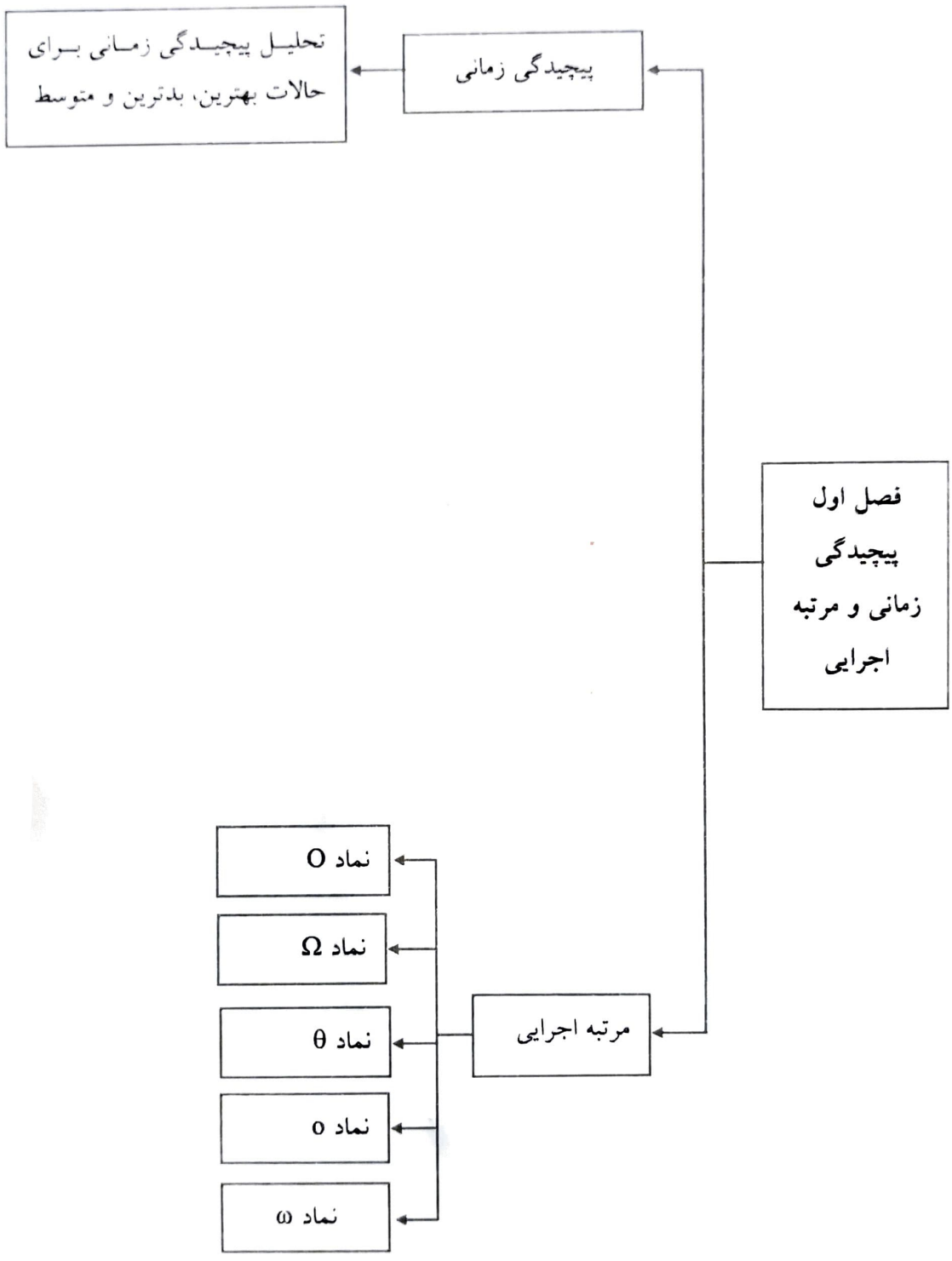
اکثر اساتید ایران از دو مرجع اصلی یعنی کتاب نیپولیتان و هورویتز و برخی نیز از کتاب آقای بهروز قلی‌زاده جهت این درس استفاده می کنند. کتاب حاضر نیز بر مبنای این سه کتاب مرجع نوشته شده است. ٦ فصل اول این کتاب بسیار مهم بوده و می بایست به طور کامل و دقیق تدریس شوند، همان طور که مشاهده خواهید کرد عموم تست های کنکور از همین ٦ فصل است. فصل های ۷ و ۸ از اهمیت کمتری برخوردار بوده و برخی مواقع اساتید به علت کم‌بودن وقت از تدریس آنها خودداری می کنند، البته در کنکورها نیز به ندرت از این فصول سؤالی مطرح می‌گردد. خود بنده در کلاس‌های دانشگاه و نیز کلاس‌های کنکور فصل‌های ۷ و ۸ را تدریس نمی‌کنم.

نکته مهم دیگر آن است که برخی کتاب‌ها و نیز اساتید محترم در این درس دو مبحث «درخت» و «مرتب‌سازی» را نیز شرح می‌دهند. از آنجا که بنده این دو مبحث را به طور کامل در کتاب ساختمان‌داده‌ها شرح داده‌ام از ذکر مجدد آنها در این کتاب خودداری کرده‌ام. ولی در عین‌حال اغلب تست‌های مرتب‌سازی را در فصل تقسیم و غلبه آورده‌ام، البته به نظر بنده و بسیاری اساتید دیگر بهتر است این دو مبحث در ساختمان داده‌ها تدریس گردند. فصل اول کتاب حاضر با موضوع «پیچیدگی اجرا» مشابه فصل دوم کتاب ساختمان داده‌ها (نوشته بنده) است که به علت اهمیت موضوع در هر دو کتاب تکرار شده است. با آن که مبحث گراف جزو سرفصل‌های درس ساختمان داده است ولی از آن جا که در بسیاری از الگوریتم‌های این درس از مفهوم گراف استفاده می‌شود، لازم است یک یادآوری خلاصه از این موضوع مهم بشود. لذا فصل چهارم را به یادآوری گراف اختصاص داده‌ایم. لازم به ذکر است که در اغلب کنکورها نیز سؤالات زیادی از فصل گراف مطرح می‌شود.

هر چند که بهتر است طراحی الگوریتم پس از ساختمان‌داده‌ها تدریس گردد، ولی کتاب حاضر به گونه‌ای نوشته شده است که بتوان این دو درس را به صورت هم‌زمان ارائه کرد.

در آخر از تمامی دانشجویان و اساتید گرامی خواهشمندم اشکالات کتاب و نیز پیشنهادات خود را مستقیماً با شماره تلفن ۰۹۱۲۱۳۸۸٤٤٥ و یا از طریق ایمیل hamid.moghassemi@gmail.com با بنده در میان بگذارند. امید است که این خدمت ناچیز مورد قبول خداوند متعال قرار گرفته و قابل استفاده شما عزیزان باشد.

حمید رضا مقسمی


فصل اول
پیچیدگی
زمانی و مرتبه
اجرایی
پیچیدگی زمانی
تحلیل پیچیدگی زمانی برای
حالات بهترین، بدترین و متوسط
مرتبه اجرایی
نماد O
نماد Ω
نماد θ
نماد o
نماد ω

فصل اول

پیچیدگی زمانی و مرتبه اجرایی

پیچیدگی زمانی (Time Complexity)

در ارزیابی یک الگوریتم دو فاکتور مهمی که باید مورد توجه قرار گیرد یکی حافظه مصرفی و دیگری زمان مصرفی الگوریتم است. یعنی الگوریتمی بهتر است که فضا و زمان کمتری را بخواهد. البته غالباً در الگوریتم‌های این کتاب فاکتور زمان مهمتر از فضا می‌باشد. از آنجا که کامپایل برنامه فقط یکبار صورت می‌گیرد، لذا در مورد زمان، فقط زمان اجرای برنامه (T_{Run}) را درنظر گرفته و از زمان کامپایل صرف‌نظر می‌کنیم.

معمولاً بازدهی الگوریتم‌ها را برحسب زمان تحلیل می‌کنیم. در این تحلیل نمی‌خواهیم تک تک دستورات اجراءشده را شمارش کنیم زیرا تعداد دستورها به نوع زبان برنامه‌نویسی و نحوه نوشتن برنامه بستگی دارد. در عوض به میزانی نیاز داریم که مستقل از کامپیوتر و زبان برنامه‌نویسی باشد. به طور کلی زمان اجرای یک الگوریتم با افزایش اندازهٔ ورودی (n) زیاد می‌شود و زمان اجراء با تعداد دفعاتی که عملیات اصلی انجام می‌شود تناسب دارد. بنابراین بازدهی الگوریتم را با تعیین تعداد دفعاتی که یک عمل اصلی انجام می‌شود، به عنوان تابعی از ورودی تحلیل می‌کنیم.

مثال ۱ : تابع زیر جمع عناصر یک آرایه را در زبان C محاسبه می‌کند.

```
float  sum (float list[ ], int n)
{
  float  s=0 ;       int i;
    for (i = 0;  i<n;  i++)
            s = s + list[i];
      return   s;
  }
```

در این برنامه اندازه ورودی همان n یا تعداد عناصر آرایه است و عمل اصلی s=s+list[i]; می‌باشد که n بار انجام می‌گیرد.

پس از تعیین اندازه ورودی، یک دستور یا گروهی از دستورها را انتخاب می کنیم به طوری که کل کار انجام شده توسط الگوریتم تقریباً متناسب با تعداد دفعاتی باشد که توسط این دستور یا گروه دستورها انجام می‌شوند. این دستور یا گروه دستورها را «عمل اصلی» در الگوریتم می نامند.

به طور کلی تحلیل پیچیدگی زمانی یک الگوریتم عبارت از تعیین تعداد دفعاتی است که عمل اصلی به ازاء هر مقدار از اندازه ورودی انجام می شود. در واقع هیچ قاعده صریحی برای انتخاب عمل اصلی وجود ندارد و این کار معمولاً با تجربه و داوری درست صورت می پذیرد.

معمولاً پیچیدگی زمانی الگوریتم را با T(n) نمایش می‌دهند. البته زمان اجرای الگوریتمی ممکن است وابسته به چند ورودی باشد مانند T(n, m). در مثال فوق اگر عمل اصلی را دستور ;s=s+list[i] فرض کنیم T(n) = n خواهد بود.

مثال ۲ : تعداد کل مراحل برنامه مثال ۱ را محاسبه کنید.

```
float  sum (float  list[ ], int  n)
{
  float  s = 0;
  int     i;
  for (i = 0;  i<n; i++)
        s = s + list[i];
  return  s;
  }
```

0
0
1
0
n+1
n
1
0
2n+3

در مثال فوق زمان اجرای هر عبارت ساده را مساوی 1 واحد زمانی فرض می‌کنیم. عبارت ساده شامل زیر برنامه نمی‌باشد. یک عبارت ساده می‌تواند به اندازه یک دستور ;x = 2 کوچک باشد و یا مشابه دستور زیر طولانی، در هر حال هر دو را 1 واحد زمانی می‌گیریم :

```
x = 5 * y + 6 * a - 5 / w ;
```

توجه کنید } و { و نیز خط اول تعریف تابع و تعریف متغیر دستوراتی نیستند که توسط CPU اجراء شوند پس مرحله اجرائی آنها صفر است. در خط ;float s = 0 چون عدد صفر در s ریخته می‌شود پس یک مرحله می‌باشد. همچنین توجه کنید دستور داخل حلقه n بار انجام می‌شود ولی آزمایش کردن شرط حلقه در خط for به تعداد n + 1 بار صورت می‌گیرد. دستور return نیز توسط CPU باید اجراء شود. همانطور که قبلاً گفتیم اگر عمل اصلی را فقط خط ;s =s+list[i] فرض کنیم آنگاه T(n)=n خواهد بود. پس توجه داشته باشید که گاهی اوقات فقط می‌خواهیم بدانیم یک دستور ویژه چند بار تکرار می‌شود و گاهی اوقات نیز تعداد کل مراحل یا گام‌های برنامه را می‌خواهیم. البته عموماً تنها تعداد اجرای عمل اصلی مدنظر قرار می‌گیرد.

نکته : هنگام محاسبه تعداد دفعاتی که یک دستور درون حلقه‌ها اجراء می‌گردد می‌توان از فرمولهای زیر استفاده کرد :

$\sum_{i=1}^{n} 1 = n$ $\qquad$ $\sum_{i=1}^{n} kf(i) = k\sum_{i=1}^{n} f(i)$ $\qquad$ K عدد ثابتی است.

$\sum_{i=1}^{n} i = 1+2+3+\ldots+n = \frac{n(n+1)}{2}$ $\qquad$ $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

جمع تصاعد عددی $= \frac{n(a_1 + a_n)}{2}$: a_1 جمله اول، a_n جمله آخر و n تعداد جملات است

راه دیگر Trace کردن برنامه است.

مثال ۳ : دستور اصلی ;x: = x+1 در تکه برنامه زیر چند بار اجرا می‌شود؟ تعداد کل گامهای برنامه چقدر است؟

```
for i: = 1 to m do
   for j: = 1 to n do
         x: = x+1;
```

راه‌حل اول : حلقه‌های داده شده مستقل از یکدیگرند بنابراین طبق آنچه در زبانهای برنامه‌نویسی پاسکال و C خوانده‌اید تعداد اجرای دستور درون حلقه‌ها برابر mn می‌باشد.

راه حل دوم : تعداد اجرای دستور اصلی $\sum_{i=1}^{m}\sum_{j=1}^{n}1=\sum_{i=1}^{m}n=n\left(\sum_{i=1}^{m}1\right)=nm=$

حال برای محاسبه تعداد کل گام‌های برنامه ابتدا حلقه بیرونی **i را کنار گذاشته و در نظر نمی‌گیریم**. در این حال دستور ;x: = x+1 به تعداد n بار و عبارت for j به تعداد (n+1) بار اجرا می‌گردد. یعنی تعداد کل (n+1+n). حال خود این ۲ خط درون حلقه i بوده و به تعداد m بار اجرا می‌شوند یعنی m(n+1)+mn از آنجا که عبارت for i نیز m+1 بار اجرا می‌شود، پس :

تعداد کل مراحل برنامه = (m + 1) + m(n + 1) + mn

مثال ٤ : دستور اصلی ;x: = x+1 در تکه برنامه زیر چند بار اجرا می‌شود؟

```
for j: = 1 to n do
   for i: = 1 to j do
         x: = x+1;
```

راه حل اول : حلقه‌های داده شده به یکدیگر وابسته‌اند :

j	تغییرات i	تعداد اجرا شدن دستور اصلی
1	1	1 بار
2	1,2	2 بار
3	1,2,3	3 بار
	.....	
n	1,2,3,...,n	n بار

تعداد اجرا شدن دستور ;x: = x +1 $=1+2+3+...+n=\frac{n(n+1)}{2}$

راه حل دوم : $\sum_{j=1}^{n}\sum_{i=1}^{j}1=\sum_{j=1}^{n}j=\frac{n(n+1)}{2}$

مثال ۵ : تعداد اجرا شدن دستور اصلی x: = x+1 در **تکه برنامه** زیر چیست؟

```
i:=n;
while (i>1) do begin
        x:=x+1;
         i: = i div 2;
end;
```

اگر n = 16 باشد :

i	شرط i>1	تعداد اجرا شدن دستور اصلی
16	درست	۱ بار
8	درست	۱ بار
4	درست	۱ بار
2	درست	۱ بار
1	غلط	-
		جمعاً ٤ بار

حال اگر n را برابر 14 فرض کنیم :

i	شرط i>1	تعداد اجرا شدن دستور اصلی
14	درست	۱ بار
7	درست	۱ بار
3	درست	۱ بار
1	غلط	-
		جمعاً ۳ بار

پس در حالت کلی دستور اصلی به تعداد $\lfloor \log_2^n \rfloor$ بار اجرا می‌شود.

یادآوری : کف یک عدد اعشاری : $\lfloor 3.7 \rfloor = \lfloor 3.2 \rfloor = 3$

سقف یک عدد اعشاری : $\lceil 3.7 \rceil = \lceil 3.2 \rceil = 4$

کف و سقف یک عدد صحیح با خود عدد برابر است : $\lfloor 3 \rfloor = \lceil 3 \rceil = 3$

تذکر : اغلب در کتاب‌های ساختمان داده‌ها و طراحی الگوریتم منظور از Log n یا Lg n عبارت Log_2^n می‌باشد.

تحلیل پیچیدگی زمانی برای حالات بهترین، بدترین و متوسط

برخی مسائل برای همه موارد یک تابع پیچیدگی دارند مثل الگوریتم جمع عناصر یک آرایه :

```
A : Array [1 .. n]  of Integer;
S := 0;
For I := 1  To  n  do
    S := S + A[i] ;
```

در برنامه فوق عمل اصلی S := S + A[i] به تعداد n بار اجرا شده و همواره T(n) = n می‌باشد. ولی در الگوریتمی مثل جستجوی خطی (ترتیبی) تابع پیچیدگی برای حالات مختلف ممکن است متفاوت باشد. فرض

کنید در آرایه n خانه‌ای A می‌خواهیم خانه به خانه از اول تا انتها به دنبال عدد معین x بگردیم. اگر x را در آرایه A پیدا کردیم بگوئیم Yes و در غیر این صورت بگوئیم No :

```
A : Array [1 .. n] of Integer;
For i := 1  to n do
    if (x = A[i]) {
          write ('Yes');
            exit ( ) ; → خروج از برنامه

    }
write ('No');
```

در برنامه فوق عمل اصلی شرط if (x = A[i]) می‌باشد.

در بدترین حالت عدد x ، در خانه آخر قرار دارد و یا اصلاً در آرایه نیست که در این حالت باید n بار عمل اصلی آزمایش کردن، انجام گیرد. برای بدترین حالت به جای نماد T(n) از نماد W(n) استفاده می‌کنیم که W مخفف Worst یعنی بدترین است. پس در برنامه فوق W(n) = n می‌باشد.

در بهترین حالت عدد x در اولین خانه قرار دارد و تنها به یک عمل if مورد نیاز است. بهترین حالت را با B نمایش می‌دهیم که مخفف Best است لذا در برنامه فوق داریم : $B(n)=1$

برای تحلیل برنامه فوق در حالت متوسط که آن را با A (مخفف Average) نشان می‌دهیم باید به صورت زیر عمل کنیم. ابتدا فرض می‌کنیم x در آرایه A وجود داشته باشد. بدیهی است احتمال آنکه x در خانه i ام باشد برابر $\frac{1}{n}$ است و تعداد دفعات آزمایش کردن در این حالت برابر i است. لذا :

$$A(n)=\sum_{i=1}^{n}\frac{1}{n}\times i=\frac{1}{n}\sum_{i=1}^{n}i=\frac{1}{n}\frac{n(n+1)}{2}=\frac{n+1}{2}$$

پس اگر x در آرایه حتماً باشد : $\boxed{A(n)=\frac{n+1}{2}}$

یعنی به طور متوسط نیمی از عناصر آرایه باید جستجو شود.

حال موردی را در نظر می‌گیریم که x ممکن است اصلاً در آرایه نباشد. فرض کنید احتمال وجود x در آرایه برابر p است. پس احتمال آنکه x در یکی از خانه‌های آرایه (مثل خانه i ام) باشد برابر $\frac{p}{n}$ است. احتمال آنکه x در آرایه نباشد برابر 1-p است. اگر x در خانه i ام باشد دستور اصلی i بار اجرا می‌شود و اگر x در آرایه نباشد دستور اصلی n بار اجرا می‌شود، لذا :

$$A(n)=\left(\sum_{i=1}^{n}\frac{p}{n}\times i\right)+(1-p)n=\frac{p}{n}\times\frac{n(n+1)}{2}+n(1-p)=n(1-\frac{p}{2})+\frac{p}{2}$$

پس اگر احتمال وجود x در آرایه P باشد : $\boxed{A(n)=n\left(1-\frac{P}{2}\right)+\frac{P}{2}}$

توجه کنید که اگر $p = 1$ باشد همان حالت قبلی $A(n) = \frac{n+1}{2}$ بدست می‌آید و اگر $p = \frac{1}{2}$ باشد $A(n) = \frac{3n}{4} + \frac{1}{4}$ می‌شود و این بدان معناست که حدود $\frac{3}{4}$ آرایه به طور میانگین باید جستجو شود.

برای الگوریتم‌هایی که فاقد پیچیدی زمانی در هر حالت می‌باشند، اغلب تحلیل‌های بدترین و حالت متوسط را محاسبه می‌کنیم. در برخی الگوریتم‌ها مثل مرتب‌سازی که به طور مکرر برای داده‌های متفاوت ورودی اعمال می‌شوند تحلیل میانگین مناسب‌تر است. ولی مثلاً تحلیل حالت میانگین در سیستم کنترل نیروگاه هسته‌ای مناسب نیست و در این حالت تحلیل بدترین حالت مفیدتر است.

تمرین اول : تست‌های با عنوان «**تست‌های پیچیدگی زمانی**» را حل کنید.

مفهوم مرتبه اجرایی الگوریتم (O)

در قسمت قبلی به طور دقیق محاسبه کردیم که یک دستور اصلی دقیقاً چند بار اجرا می‌شود و یا اینکه تعداد کل مراحل برنامه چند گام است. در عمل، محاسبات دقیق فوق اغلب مشکل بوده و از طرف دیگر این محاسبه دقیق مورد نیاز نیست. در کاربردهای واقعی که سرعت کامپیوترها و نوع کامپایلر می‌تواند سرعت اجرای برنامه‌ها را چندین مرتبه کاهش یا افزایش دهد، بحث بر سر اینکه فلان دستورالعمل 1000 بار اجرا می‌شود یا 999 بار، لازم نیست.

به جای محاسبات دقیق ما به ابزاری نیاز داریم که زمان اجرای الگوریتم‌ها را به صورت حدودی و طبقه‌بندی شده نشان می‌دهد. این بحث تا حدی شبیه بحث هم‌ارزی‌ها در مسائل حد ریاضیات است. مثلاً در ریاضیات خوانده‌اید که $\lim_{n\to\infty} 5n^2 + 4n - 3$ هم‌ارز با عبارت $5n^2$ است یعنی هنگامی که n زیاد می‌شود عبارت 4n-3 در مقابل $5n^2$ قابل صرف‌نظر بوده و می‌توانیم فقط $5n^2$ را در نظر بگیریم.

مرتبه اجرایی یک الگوریتم نیز شبیه هم‌ارزی فوق است. مثلاً الگوریتمی که پیچیدگی زمانی آن $T(n) = 5n - 4$ می‌باشد از مرتبه n است که آن را به صورت $T(n) \in O(n)$ نمایش می‌دهیم. یا مثلاً الگوریتمی که پیچیدگی زمانی آن $6n^2 - 3n + 2$ می‌باشد از مرتبه $O(n^2)$ است و آن را به صورت $6n^2 - 3n + 2 \in O(n^2)$ نشان می‌دهیم.

تعریف دقیق O جلوتر آورده شده است.

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

مثال ٦ :
$$f(n) = \frac{n}{2}(n-1) = \frac{1}{2}n^2 - \frac{n}{2} \in O(n^2)$$

مثال ۷ : مرتبه اجرائی برنامه های زیر را بدست آورید :

الف) $x := x + 1 ; \quad \Rightarrow \quad O(1)$

ب) `for i: = 1 to n do`

`x: = x+1;` $\Rightarrow O(n)$

ج) `for i: = 1 to n do`

`for j: = 1 to n do` $\Rightarrow O(n^2)$

`x: = x + 1 ;`

$O(1)$ زمان محاسبه ثابتی را نشان می‌دهد که تابعی از n نیست. بنابراین برنامه‌ای که حلقه ندارد و دستورات آن محدود است (مثلاً 1000 خط ثابت دارد) از مرتبه $O(1)$ است.

مثال ۸ : تعداد دقیق اجرا شدن دستور write('OK') و مرتبه اجرایی آن را در تکه برنامه زیر بدست آورید:

```
for i:=1 to n do
    for j:=i to n do
        write('OK');
```

حل : حلقه‌های فوق وابسته به یکدیگر بوده و همانطور که قبلاً دیدید تعداد دقیق اجرا شدن دستور write برابر جمع تصاعد عددی از 1 تا n می‌باشد یعنی $\frac{n(n+1)}{2}$ و بنابر قضیه اصلی مرتبه اجرایی آن $O(n^2)$ می‌باشد.

مثال ۹ : مرتبه اجرایی حلقه زیر $O(1)$ می‌باشد چرا که حلقه به تعداد معین و محدودی اجرا می‌شود:

```
for i:=5 to 13 do
        write('OK');
```

نکته ۱ : با توجه به مثال‌های فوق می‌توان گفت ۲ حلقه for تودرتو (چه وابسته به هم باشند و چه مستقل) در حالت معمولی از مرتبه $O(n^2)$ و به همین ترتیب ۳ حلقه for تودرتو (مستقل یا وابسته) از مرتبه $O(n^3)$ می‌باشد البته به شرطی که تمامی حلقه‌ها به نحوی تابعی از n باشند.

مثال ۱۰ : مرتبه اجرائی برنامه زیر چیست؟

فرض کنید n = 32 باشد آنگاه

```
x : = 0;
i : = n ;
while (i > 1) do begin
        x : = x + 1;
        i : = i div 2;
end;
```

i	x
32	1
16	2
8	3
4	4
2	5
1	-

پس برای n = 32 عمل اصلی x: = x +1; ۵ بار انجام می‌شود ($5 = \log_2 32$). پس در حالت کلی مرتبه اجرائی الگوریتم فوق برابر $O(\log n)$ می‌باشد. توجه کنید در درس ساختمان داده عموماً منظور از logn یعنی $\log_2 n$ ، که گاهی آن را به صورت lg n هم نمایش می‌دهند.

نکته ۲ : در حلقه while که به طور طبیعی شمارنده آن از n تا 1 تغییر می‌کند اگر مرتباً شمارنده آن با دستور i:=i div k; بر عدد k تقسیم شود مرتبه اجرایی آن $O(\log_k^n)$ خواهد بود. به همین ترتیب اگر شمارنده با دستور i:=i*k; از 1 تا n تغییر کند باز هم مرتبه اجرایی آن $O(\log_k^n)$ می‌باشد :

```
i: = n;
while (i >1) {
    عمل اصلی ;
    i: = i div k;
}
```

یا

```
i:=1;
while(i < n) {
    عمل اصلی ;
    i:=i*k:
}
```

$\Rightarrow \quad O(\log_k^n)$

پس مفهوم O بدین معناست که زمان اجرای یک الگوریتم حدوداً به چه میزانی است.

رشد توابع

طبق تعریف، رشد تابع $f(n)$ به شرطی از تابع $g(n)$ بیشتر است که داشته باشیم:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\infty$$

مثال ۱۱: رشد $f(n)=\frac{1}{n}$ از $g(n)=\frac{1}{n^2}$ بیشتر است چرا که:

$$\lim_{n\to\infty}\frac{\frac{1}{n}}{\frac{1}{n^2}}=\lim_{n\to\infty}\frac{n^2}{n}=\lim_{n\to\infty}n=\infty$$

و می‌نویسیم $f>g$ یعنی رشد f از g بیشتر است.

به همین ترتیب اگر $\lim_{n\to\infty}\frac{f(n)}{g(n)}=k$ شود که k عدد ثابت غیر صفر باشد آن‌گاه می‌گوییم رشد f و g یکسان است و اگر $\lim_{n\to\infty}\frac{f(n)}{g(n)}=0$ شود می‌گوییم رشد f از g کمتر است یعنی:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\begin{cases}0 & f<g\\ \infty & f>g\\ k\neq 0 & f=g\end{cases}$$

مثال ۱۲: رشد 2^n از n^2 بیشتر است چرا که:

$$\lim_{n\to\infty}\frac{2^n}{n^2}=\infty \qquad \text{یا} \qquad \lim_{n\to\infty}\frac{n^2}{2^n}=0$$

پس از نظر رشد $n^2<2^n$ است.

مثال ۱۳: رشد $f(n)=\log_a n$ با $g(n)=\log_b n$ (که $a,b>1$ ثابت هستند) یکسان است. در دبیرستان خوانده‌اید که:

$$\log_a n=\frac{\log_x n}{\log_x a}$$

پس:
$$\lim_{n\to\infty}\frac{\log_a n}{\log_b n}=\lim_{n\to\infty}\frac{\frac{\text{Ln } n}{\text{Ln } a}}{\frac{\text{Ln } n}{\text{Ln } b}}=\lim_{n\to\infty}\frac{\text{Ln } b}{\text{Ln } a}=\frac{\text{Ln } b}{\text{Ln } a}=k\neq 0$$

چون نتیجه حد یک عدد ثابت غیر صفر شده، پس هم رشد هستند و مثلاً تابع $\log_5 n$ با تابع $\log_2 n$ هم رشد است. بدین خاطر است که در کتاب‌های کامپیوتری هنگام ذکر مرتبه اجرایی یک برنامه لگاریتمی اغلب پایه آن را نمی‌نویسند و مثلاً می‌گویند این برنامه از مرتبه $O(\log n)$ است. در زیر چند تابع مهم از نظر رشد به ترتیب نوشته شده‌اند که لازم است آنها را حفظ کنید:

$$\frac{1}{n^2}<\frac{1}{n}<1<\lg n<\sqrt{n}<n<n\lg n<n^2<n^3<\cdots<2^n<3^n<\cdots<n!<n^n$$

برنامه‌هایی که مرتبه اجرایی آنها یا رشد آنها $O(1)$ است برنامه‌های ثابت نامیده می‌شوند. برنامه‌های $O(\lg n)$ لگاریتمی، $O(n)$ خطی، $O(n^2)$ مرتبه 2، $O(n^3)$ مرتبه 3 و در حالت کلی $O(n^m)$ چندجمله‌ای، $O(2^n)$ نمایی و $O(n!)$ فاکتوریل خوانده می‌شوند.

تذکر مهم: ترتیب رشد توابع برای nهای خیلی بزرگ تعریف می‌شود $(n\to\infty)$ و برای nهایی که کمتر از یک حد خاص هستند ممکن است ترتیب فوق درست نباشد مثلاً ترتیب $n^2<2^n<n!$ برای n=3 صادق نیست:
$3^2\not<2^3\not<3!$

ولی برای nهای $n\geq 5$ صادق است:
$5^2<2^5<5!$

تعریف دقیق O

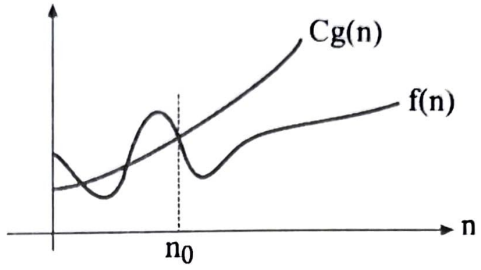
تعریف: اگر $f(n)$ و $g(n)$ دو تابع پیچیدگی باشند آنگاه می‌نویسیم $f(n)\in O(g(n))$ و می‌خوانیم $f(n)$ عضو اُی بزرگ $g(n)$ است هرگاه حداقل یک عدد ثابت $C\in\mathbb{R}^+$ و یک $n_0\in\mathbb{N}$ وجود داشته باشد که:

$$\forall\, n\geq n_0 \;:\; f(n)\leq Cg(n)$$

عبارت فوق بدین معناست که برای nهای به اندازه کافی بزرگ $(n\geq n_0)$، $f(n)$ همواره از $Cg(n)$ کوچکتر یا مساوی است، یعنی در این حالت رشد g بیشتر یا مساوی f است یا به عبارتی دیگر f سریعتر یا هم‌سرعت g می‌باشد (k عدد ثابت غیر صفر و محدود است):

$$f(n)\in O(g(n)) \xleftrightarrow{\text{معادل}} \text{رشد } f\leq \text{رشد } g \xleftrightarrow{\text{معادل}} \lim_{n\to\infty}\frac{g(n)}{f(n)}=\infty \text{ یا } k$$

از نظر نموداری مفهوم O را می‌توان مثلاً به صورت زیر نشان داد:



یعنی n_0ی وجود دارد که از آنجا به بعد Cg(n) همواره بالاتر از f(n) قرار می‌گیرد و در واقع O یک کران بالا را برای توابع مشخص می‌سازد. مثلاً $3n+5\in O(n)$ است ولی در عین حال عبارت $3n+5\in O(n^2)$ یا $3n+5\in O(2^n)$ نیز درست هستند ولی عموماً منظور همان $3n+5\in O(n)$ است. پس توجه کنید چیزی که جلوی O می‌آید می‌تواند بزرگتر یا مساوی رشد تابع سمت چپی باشد. این مفهوم را می‌توان به صورت زیر هم بیان کرد که O(n) شامل همه توابعی است که رشدشان کمتر یا مساوی n است یعنی:

$$O(n)=\{\ldots,\ n+3\ ,\ n^2-7\ ,\ n\log n\ ,\ 2n^2+n\ ,\ n!+3^n,\ldots\}$$

مثال ۱۴: $n^2+3n-6\in O(n^2)$ است و عبارات زیر هم درست می‌باشند:

$$n^2+3n-6\in O(n^3)\quad ,\quad n^2+3n-6\in O(n^2\log n)$$
$$O(n^2)=\{\ldots\ ,\ n^2\ ,\ n^2+3n-6\ ,\ n^2+9n\ ,\ n^3-7n\ ,\ 3^n-6n\ ,\ \ldots\}$$

هر چه پیچیدگی یک الگوریتم بیشتر باشد، سرعت اجرای آن کمتر است.

تذکر: در برخی از کتاب‌ها به جای علامت $\in$ از علامت $=$ استفاده شده است ولی از نظر مفهومی علامت $\in$ دقیق‌تر است. مثلاً: $5n^3-2n\in O(n^3) \longleftrightarrow 5n^3-2n=O(n^3)$

مثال ۱۵: زمان اجرای یک الگوریتم $1000n^2$ و زمان اجرای الگوریتم دیگری $10n^3$ می‌باشد. با نماد O الگوریتم اول از مرتبه $O(n^2)$ و الگوریتم دوم از مرتبه $O(n^3)$است ولی برای nهای کمتر از 100 الگوریتم دوم سریعتر از الگوریتم اول خواهد بود چرا که:

$$1000n^2\le 10n^3\Rightarrow 100\le n$$

لذا هنگام مقایسه دقیق سرعت اجرای الگوریتم‌ها به محدوده‌ی n نیز باید توجه داشته باشیم و همان طور که قبلاً گفتیم عموماً الگوریتم‌ها را برای nهای بزرگ $(n\to\infty)$ با یکدیگر مقایسه می‌کنیم.

تذکر: برنامه‌هایی با پیچیدگی نمایی مثل $O(2^n)$ تنها برای مقادیر کوچک n (اغلب $n\le 40$) سودمند هستند و از نظر عملی برای nهای بزرگ ($n\ge 100$) تنها برنامه‌هایی با پیچیدگی کم (مثل n ، $n\log n$ ، n^2 و n^3) سودمند می‌باشند.

مثال ۱۶: فرض کنید در کامپیوتری در هر ثانیه یک میلیارد (10^9) عملیات پایه می‌تواند اجرا شود و یا به عبارتی دیگر هر عمل پایه‌ای یک نانوثانیه $(10^{-9}s)$ زمان ببرد. جدول زیر زمان اجرای تقریبی الگوریتم‌هایی با پیچیدگی‌ها و اندازه‌های ورودی (n) متفاوت را نشان می‌دهد.

$(\log_n 10 \approx 3 \quad , \quad \log_2 100 \approx 7 \quad , \quad \log_2 1000 \simeq 10)$

اندازه ورودی	تابع پیچیدگی			
n	$\log_2 n$	n	n^2	2^n
10	$3\times10^{-9}s$	$10\times10^{-9}=10^{-8}s$	$10^2\times10^{-9}=10^{-7}s$	$2^{10}\times10^{-9}\approx10^3\times10^{-9}=10^{-6}s$
10^2	$7\times10^{-9}s$	$10^2\times10^{-9}=10^{-7}s$	$10^4\times10^{-9}=10^{-5}s$	بیش از صد میلیارد قرن
10^3	$10\times10^{-9}s$	$10^3\times10^{-9}=10^{-6}s$	$10^6\times10^{-9}=10^{-3}s$	بیش از صدهزار میلیارد قرن

جدول بالا نشان می‌دهد که اگر در آینده، سرعت کامپیوترها چند هزار برابر هم شود، باز هم توابع نمایی و فاکتوریل، توابعی رام‌نشدنی و مهارنشدنی هستند و افزایش سرعت کامپیوترها در عمل تنها بر روی الگوریتم‌های چندجمله‌ای اثر دارد. از جدول بالا مشخص می‌گردد که توابع رام‌شدنی مثل n ، $n\log n$ ، n^2 و n^3 حتی اگر در ضرایب بزرگ ضرب شوند مثل $10^6\times n^3$ باز هم رشد آنها از توابع نمایی و فاکتوریل مثل 0.001×2^n کمتر خواهد بود.

نکته ۱: در حالت کلی برای $a,b>1$ رشد توابع زیر به ترتیب مشخص شده است:

$$n^b < a^n < n! < n^n$$

مثلاً رشد 3^n بیشتر از n^{100} است.

نکته ۲: در حالت کلی رشد $\log n$ از n^ε (که $\varepsilon>0$) کمتر است. مثلاً $\log n < n^{0.01}$ و یا $\log n < \sqrt{n} = n^{0.5}$ چرا که:

$$\lim_{n\to\infty}\frac{\text{Ln } n}{n^\varepsilon} \xrightarrow{\text{هوپیتال}} \lim_{n\to\infty}\frac{\frac{1}{n}}{\varepsilon n^{\varepsilon-1}} = \lim_{n\to\infty}\frac{1}{\varepsilon\times n\times n^{\varepsilon-1}} = \lim_{n\to\infty}\frac{1}{\varepsilon n^\varepsilon} = 0$$

می‌توان اثبات کرد که حتی اگر $\log n$ به توان b برسد باز هم رشد کمتری از n^ε دارد یعنی از نظر رشد تابعی:

$$\log^b n < n^a \quad : \quad a,b>0$$

مثلاً $\log^3 n < n^{0.001}$ ، $\log^7 n < n^2$.

نکته ۳: می‌توان اثبات کرد از نظر رشد توابع:

$$\log n! < n^\alpha < (\log n)! \quad : \quad \text{برای } \alpha>1$$

نکته ٤ : اگر $O(n_1)$ ، t_1 و V_1 به ترتیب مرتبه اجرایی و زمان اجرایی الگوریتمی در کامپیوتری با سرعت V_1 باشند و به همین ترتیب $O(n_2)$ ، t_2 و V_2 به ترتیب مرتبه اجرایی و زمان اجرای الگوریتمی دیگر در کامپیوتری با سرعت V_2 باشند، خواهیم داشت :

$$\frac{t_2}{t_1}=\frac{O(n_2)}{O(n_1)}\times\frac{V_1}{V_2}$$

چرا که زمان اجرا (t) نسبت مستقیم با O و نسبت معکوس با سرعت (V) دارد.

مثال ۱۷: الگوریتمی با مرتبه زمانی $O(n\log n)$ در کامپیوتری در مدت زمان ۱ ثانیه اجرا می‌شود. همان الگوریتم روی کامپیوتر دیگری با سرعت 100 برابر در چه مدت زمانی اجرا خواهد شد؟

حل :

$$\frac{O(n_2)}{O(n_1)}=1=\frac{t_2}{t_1}\times\frac{V_2}{V_1}=\frac{t_2}{1}\times\frac{100}{1}\Rightarrow t_2=\frac{1}{100}=10^{-2}\ \text{sec}$$

مثال ۱۸: برنامه‌ای با اندازه 10 و مرتبه اجرایی $O(n^2)$ روی سیستمی در مدت زمان 1 msec اجرا می‌شود. همان مسأله با اندازه 100 روی همان کامپیوتر در چه مدت زمان اجرا می‌شود؟

حل :

$$\frac{O(n_2)}{O(n_1)}=\frac{(100)^2}{(10)^2}=\frac{t_2\times V_2}{t_1\times V_1}=\frac{t_2}{1}\Rightarrow t_2=10^2=100\text{m sec}$$

نمادهای Ω و θ (امگای بزرگ و تتا)

همان‌طور که قبلاً گفتیم نماد O حد بالایی را برای یک تابع مشخص می‌سازد ولی در مورد مطلوب بودن این حد چیزی را نشان نمی‌دهد. مثلاً دیدید که $3n+3\in O(n)$ ولی در عین حال عبارت $3n+3\in O(n^2)$ نیز درست است. به همین ترتیب عبارات $3n+3\in O(n^3)$ یا $3n+3\in O(2^n)$ هم درست هستند. ولی در عمل منظور ما از مرتبه اجرایی $3n+3$ همان $O(n)$ می‌باشد.

به صورت صوری می‌توان عبارت زیر را برای تعریف O در نظر گرفت :

$f\le g$ یعنی رشد تابع g بزرگتر یا مساوی رشد تابع f است. $\quad f\le g\approx f(n)\in O(g(n))$

عکس تعریف O ، تعریف Ω می باشد که به صورت صوری، معادل عبارت زیر است :

$$f\ge g\approx f(n)\in\Omega(g(n))$$

یعنی Ω حد پایین را برای تابع مشخص می‌سازد.

مثال ۱۹: برای $f(n)=5n^3+2n$ عبارات زیر همگی درست هستند :

$$f(n)=\Omega(n)\ ,\quad f(n)=\Omega(n^2)\ ,\quad f(n)=\Omega(n^3)$$

ولی اغلب منظور، از Ω در مثال فوق همان $f(n)=\Omega(n^3)$ است. بنابراین تعاریف O و Ω چندان دقیق نبوده و به همین دلیل اغلب از مفهوم θ استفاده می‌شود که مطابق تعریف صوری زیر است :

$f=g$ یعنی رشد تابع f با رشد تابع g برابر است.

$$f=g \approx f(n)\in\theta(g(n))$$

مثال ۲۰: برای $f(n)=5n^3-5n$ عبارات زیر درست هستند :

$$f(n)=O(n^3)\ ,f(n)=\Omega(n^3)\ ,f(n)=\theta(n^3)$$
$$f(n)=O(n^5)\ ,\ f(n)=\Omega(n)$$

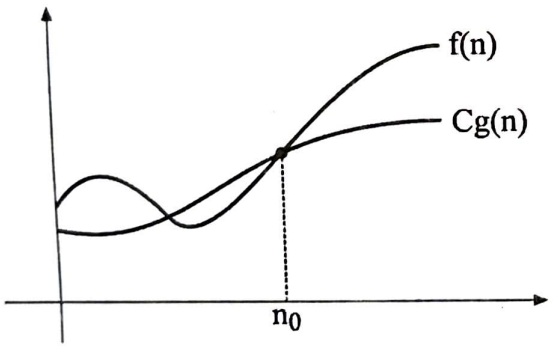
ولی عبارات زیر غلط هستند :

$$f(n)=\theta(n^5)\quad ,\quad f(n)=\theta(n)$$

تذکر : اغلب در تست‌های کنکور منظور از نماد O همان θ می‌باشد.

در واقع مشابه O برای Ω نیز می‌توان مفهوم نموداری زیر را در نظر گرفت:

یعنی حداقل یک ضریب ثابت C و یک n_0 وجود دارد که برای $\forall n\geq n_0$ همواره رابطه $f(n)\geq Cg(n)$ برقرار است.



یعنی رشد g از f کمتر یا مساوی است و آن را به صورت $f(n)\in(g(n))\Rightarrow\Omega(g(n))$ نمایش می‌دهیم. همانطور که قبلاً گفتیم در برخی از کتاب‌ها به جای علامت تعلق $\in$ از علامت = استفاده می‌شود.

مثال ۲۱: $\Omega(n^2)$ شامل همه توابعی است که رشدشان بزرگتر یا مساوی n^2 است یعنی:

$$\Omega(n^2)=\{\ldots,n^2+3,5n^3-7,n^3\log^2 n,2^n,\ldots\}$$
$$n^2+3=\Omega(n^2)\ ,\quad 5n^3+7=\Omega(n^2)\ ,\quad n^3\log^2 n=\Omega(n^2),\ldots$$

پس به کمک مفهوم رشد یا حد می‌توان O، Ω و θ را به صورت زیر در نظر گرفت: (k عدد ثابت غیر صفر است)

$$f=O(g)\Leftrightarrow f\leq g\Leftrightarrow \lim_{n\to\infty}\frac{f}{g}=0 \quad \text{یا} \quad k$$

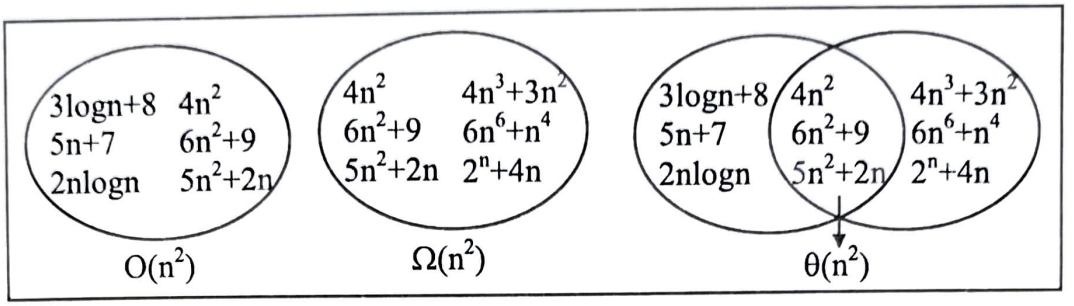
$$f=\Omega(g)\Leftrightarrow f\geq g\Leftrightarrow \lim_{n\to\infty}\frac{f}{g}=\infty \quad \text{یا} \quad k$$

$$f=\theta(g)\Leftrightarrow f=g\Leftrightarrow \lim_{n\to\infty}\frac{f}{g}=k$$

در واقع با توجه به تعاریف فوق داریم :

$$\theta(f(n)) = O(f(n)) \cap \Omega(f(n))$$

مثال ۲۲:



نکته ۱ : $g(n) \in O(f(n))$ اگر و فقط اگر $f(n) \in \Omega(g(n))$

نکته ۲ : $g(n) \in \theta(f(n))$ اگر و فقط اگر $f(n) \in \theta(g(n))$

نکته ۳: $g(n) \in \theta(f(n))$ اگر و فقط اگر $g(n) \in O(f(n))$ و $g(n) \in \Omega(f(n))$

مثال ۲۳: عبارات زیر همگی درست می‌باشند :

۱) $5n^2 - 6n = \theta(n^2)$

۲) $n! = O(n^n)$

۳) $2n^2 2^n + n\log n = \theta(n^2 2^n)$

۴) $\sum_{i=0}^{n} i^2 = \theta(n^3)$

۵) $\sum_{i=0}^{n} i = \theta(n^2)$

۶) $n^{2^n} + 6 \times 2^n = \theta(n^{2^n})$

۷) $n^3 + 10^6 n^2 = \theta(n^3)$

۸) $6n^3 / (\log n + 1) = O(n^3)$

۹) $n^{1.001} + n\log n = \theta(n^{1.001})$

۱۰) $10n^3 + 15n^4 + 100n^2 2^n = O(n^2 2^n)$

۱۱) $33n^3 + 4n^2 = \Omega(n^2)$

۱۲) $33n^3 + 4n^2 = \Omega(n^3)$

۱۳) $6n^2 + 20n \in O(n^3)$

مثال ۲۴: عبارات زیر همگی نادرست هستند :

۱) $10n^2 + 9 = O(n)$

۲) $n^2 \log n = \theta(n^2)$

۳) $n^2 / \log n = \theta(n^2)$

۴) $3^n = O(2^n)$

۵) $n^3 2^n + 6n^2 3^n = O(n^3 2^n)$

۶) $6n^2 + 20n \in \Omega(n^3)$

مثال ۲۵:

a) اگر $f(n) > 1$ باشد آن‌گاه $f(n) = O\left(f^2(n)\right)$ برقرار است.

ولی اگر $f(n) < 1$ باشد رابطه $f(n) = O\left(f^2(n)\right)$ برقرار نیست.

b) رابطه $f(n) + O(f(n)) = \theta(f(n))$ صحیح است.

مثلاً برای $f(n)=n^2$، $n=O(n^2)$ داریم $n^2+n=\theta(n^2)$

c) عبارت اگر $f(n)=O(g(n))$ آن‌گاه $2^{f(n)}=O(2^{g(n)})$ برقرار نیست چرا که مثلاً اگر $f(n)=2n$ و $g(n)=n$ باشد با آن‌که $f(n)=O(g(n))$ است ولی $2^{2n}\neq O(2^n)$ می‌باشد.

d) رابطه $f(n)=\theta\left(f\left(\frac{n}{2}\right)\right)$ غلط است چرا که مثلاً $2^n\neq\theta\left(2^{\frac{n}{2}}\right)$.

نکته ٤ : بین دو عدد حقیقی x و y یکی از حالات $x<y$، $x=y$ و $x>y$ حتماً وجود دارد ولی برای دو تابع $f(n)$ و $g(n)$ ممکن است هیچکدام از حالات $f(n)=O(g(n))$ و $f(n)=\theta(g(n))$ و $f(n)=\Omega(g(n))$ برقرار نباشند، مثل توابع $f(n)=n^{1+\sin n}$ و $g(n)=n$ چرا که $1+\sin n$ بین 0 و 2 نوسان می‌کند.

نکته ٥ : $\boxed{\log_a^n=\theta\left(\log_b^n\right)=\theta(Lnn)}$ چرا که : $\log_a^b$ یک عدد ثابت است:

$$\log_a^n=\frac{\log_b^n}{\log_a^b}=\theta\left(\log_b^n\right)$$

یعنی همه توابع لگاریتمی در یک دسته پیچیدگی قرار دارند و مثلاً $\log_3^n$ با $\log_2^n$ فرقی ندارد.

نکته ٦ : $\log n!=O(n\log n)$ چرا که :

$$\log n!<\log n^n=n\log n$$

به کمک تقریب استرلینگ $n!\approx\sqrt{2\pi n}\left(\frac{n}{e}\right)^n$ برای nهای بزرگ می‌توان اثبات کرد $\lg n!$ با $n\lg n$ رشد یکسانی دارد یعنی:

$$\boxed{\log n!=\theta(n\log n)}$$

نکته ۷ :

$$\boxed{\sum_{i=1}^{n}i^k=\theta\left(n^{k+1}\right)}$$

مثلاً :

$$1+2+3+\ldots+n=\frac{n(n+1)}{2}=\theta\left(n^2\right)$$

$$1^2+2^2+3^2+\ldots+n^2=\frac{n(n+1)(2n+1)}{6}=\theta\left(n^3\right)$$

نکته ۸ :

$$1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}=\theta(\log n)$$

اثبات فرمول مهم فوق به صورت زیر است:

$$1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n-1}=\theta(\log(n-1))$$

n را به n-1 تبدیل می‌کنیم:

با تفریق دو رابطه فوق به دست می‌آید که :

$$\frac{1}{n}=\theta(\log n-\log(n-1))=\theta(\log\frac{n}{n-1})$$

$$\lim_{n\to\infty}\frac{\mathrm{Ln}\frac{n}{n-1}}{\frac{1}{n}}\xrightarrow{\text{هوپیتال}}\lim_{n\to\infty}\frac{\frac{\frac{n-1-n}{(n-1)^2}}{\frac{n}{n-1}}}{-\frac{1}{n^2}}=\lim_{n\to\infty}\frac{\frac{-1}{n(n-1)}}{-\frac{1}{n^2}}=\lim_{n\to\infty}\frac{n^2}{n(n-1)}=1$$

چون نسبت حد فوق ۱ شده است پس هم‌مرتبه هستند.

یادآوری : در ریاضی عمومی به سری $\sum_{n=1}^{\infty}\frac{1}{n}$ ،سری هارمونیک یا همساز می‌گوییم که واگرا است ولی این واگرایی به صورت لگاریتمی است نه خطی یعنی از مرتبه $\theta(\log n)$ است نه از مرتبه $\theta(n)$.

نمادهای o و ω (اُی کوچک و امگای کوچک)

در برخی کتاب‌ها تعریف o کوچک به صورت زیر آمده است.

$f(n)=o(g(n))$ است، اگر و تنها اگر عبارت زیر برقرار باشد :

$$f(n)=o(g(n))\Leftrightarrow\lim_{n\to\infty}\frac{f(n)}{g(n)}=0\Leftrightarrow g>f$$

یعنی رشد g از رشد f بیشتر باشد

مثال ۲۶ : $3n+2=o(n^2)$ است چرا که :

$$\lim_{n\to\infty}\frac{3n+2}{n^2}=0$$

در واقع می‌توان گفت اُی بزرگ O به معنای بزرگتر یا مساوی ولی اُی کوچک o به معنای فقط بزرگتر است :

$$f<g\approx f(n)=o(g(n)) \quad , \quad f\le g\approx f(n)=O(g(n))$$

مثال ۲۷ : مشابه مثال فوق می‌توان نشان داد که :

$$6*2^n+n^2=o(3^n) \quad , \quad 3n+2=o(n\log n)$$

$6*2^n + n^2 \neq o(2^n)$, $3n+2 \neq o(n)$

$6*2^n + n^2 = o(2^n \log n)$

مثال ۲۸ : $5n^2 - 3n + 4 = O(n^2)$ ولی $5n^2 - 3n + 4 \neq o(n^2)$

نکته : همان‌طور که قبلاً گفتیم دسته‌های پیچیدگی معروف به ترتیب از چپ به راست عبارتند از :

$(2 < i < k \ , \ 1 < a < b)$

$\theta(\log n), \theta(n), \theta(n \log n), \theta(n^2), \theta(n^i), \theta(n^k), \theta(a^n), \theta(b^n), \theta(n!)$

اگر تابع g(n) در طرف چپ تابع f(n) باشد آنگاه $g(n) \in o(f(n))$

مثلاً برای $a < b$ داریم $a^n \in o(b^n)$ چرا که :

زیرا $\frac{a}{b} < 1$ می‌باشد : $$\lim_{n\to\infty} \frac{a^n}{b^n} = \lim_{n\to\infty} (\frac{a}{b})^n = 0$$

برعکس o کوچک، می‌توان ω کوچک را به صورت زیر تعریف کرد :

یعنی رشد g از f کمتر است: $$f(n) = \omega(g(n)) \Leftrightarrow \lim_{n\to\infty} \frac{g(n)}{f(n)} = 0 \Leftrightarrow f > g$$

یا :

$$f(n) = \omega(g(n)) \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

مثال ۲۹ : $5n^2 - 3n + 4 = \omega(n)$ چرا که :

$$\lim_{n\to\infty} \frac{n}{5n^2 - 3n + 4} = 0$$

توجه کنید که $5n^2 - 3n + 4 = \Omega(n^2)$ ولی $5n^2 - 3n + 4 \neq \omega(n^2)$

به عبارتی دیگر می‌توان گفت Ω به معنای کوچکتر یا مساوی ولی ω به معنای فقط کوچکتر است :

$$f \geq g \approx f(n) = \Omega(g(n)) \quad , \quad f > g \approx f(n) = \omega(g(n))$$

مقایسه خواص O، Ω، θ ، o و ω

۱ـ خاصیت تقارنی : این خاصیت را فقط θ دارد :

$$f(n) = \theta(g(n)) \Leftrightarrow g(n) = \theta(f(n))$$

۲ـ خاصیت بازتابی :

$$f(n) = \theta(f(n)) \quad , \quad f(n) = \Omega(f(n)) \quad , \quad f(n) = O(f(n))$$

خاصیت بازتابی را o و ω ندارند.

۳ـ خاصیت ترانهاده تقارنی :

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$$

$$f(n) = o(g(n)) \Leftrightarrow g(n) = \omega(f(n))$$

٤ـ خاصیت تعدی :

$$f(n) = O(g(n)) \ , \ g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$$
$$f(n) = \Omega(g(n)) \ , \ g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$$
$$f(n) = \theta(g(n)) \ , \ g(n) = \theta(h(n)) \Rightarrow f(n) = \theta(h(n))$$
$$f(n) = o(g(n)) \ , \ g(n) = o(h(n)) \Rightarrow f(n) = o(h(n))$$
$$f(n) = \omega(g(n)) \ , \ g(n) = \omega(h(n)) \Rightarrow f(n) = \omega(h(n))$$

تذکر : با توجه به خواص گفته شده می‌توان تشابهات مفهومی زیر را در نظر گرفت :

$$f \le g \approx f(n) = O(g(n)) \ , \ f \ge g \approx f(n) = \Omega(g(n))$$
$$f < g \approx f(n) = o(g(n)) \ , \ f > g \approx f(n) = \omega(g(n))$$
$$f = g \approx f(n) = \theta(g(n))$$

تمرین دوم : تست‌های با عنوان «**مرتبه اجرایی**» را حل کنید.

تست‌های فصل اول

تست‌های پیچیدگی زمانی

۱- در برنامه زیر تعداد دفعات تکرار دستورالعمل شماره 3 برابر است با ..... (علوم کامپیوتر ـ دولتی ۸۰)

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

۱) n^2 ۲) $\frac{n(n+1)}{2}$ ۳) $\frac{n^2}{2}$ ۴) $\frac{n(n-1)}{2}$

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

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

۱) 2m + 2n
۲) $mn^2 + m^2n$
۳) m(2n+1)+1
۴) m(2n+1)-1

۳- مقدار محاسبه شده برای TOTAL را تعیین کنید. (علوم کامپیوتر ـ دولتی ۷۹)

```
TOTAL:=0;
For I:=1 To N Do
begin
     K:=N;
     while (K< > l) Do begin
              K:=K DIV 2;
              TOTAL : = TOTAL + 1;
     end;
end;
```

۱) $N(\log_2 N - 1)$ ۲) $N \log_2 N$ ۳) $N(\log_2 N+1)$ ۴) $\log_2 N + 1$

۴ـ در قطعه برنامه زیر، تعداد دفعاتی که جملهٔ نشان داده شده با علامت (*) اجرا می‌شود، برابر کدام است؟ (کارشناسی ارشد صنایع - دولتی ۷۹)

```
FOR I : = 1   TO   M   DO
FOR J := 1   TO   (M-I)   DO
      T := T + 1;                (*)
```

۱) $\frac{M(M+1)}{2}$ ۲) $\frac{M(M+1)}{2}+1$ ۳) $\frac{M(M-1)}{2}$ ٤) $\frac{M(M-1)}{2}+1$

۵ـ فرض کنید **M** یک عدد صحیح بزرگتر از یک است. پس از اجرای قطعه برنامهٔ : (کارشناسی ارشد صنایع - دولتی ۸۰)

```
X := 0;
FOR I := 0 TO M DO
      FOR J := 0 TO I DO
          FOR K:= 1 TO M DO
          X =: X + 1 ;
```

مقدار **X** برابر است با .... .

۱) $\frac{M^2(M+1)}{2}+1$ ۲) $\frac{M^2(M+1)}{2}$

۳) $\frac{M(M+1)(M+2)}{2}+1$ ٤) $\frac{M(M+1)(M+2)}{2}$

٦- در تکه برنامه زیر **Process a** چند بار اجرا می‌شود؟ (علوم کامپیوتر ـ دولتی ۸٤)

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

۱) $\frac{N}{2}$ ۲) $\frac{N^2}{4}$ ۳) $\frac{N+1}{2}$ ٤) $\left\lfloor\frac{N+1}{2}\right\rfloor\times\left\lfloor\frac{N+1}{2}\right\rfloor$

۷- قطعه کد زیر را در نظر بگیرید . تعداد دفعاتی که عبارت **x:=x + 1** در قطعه کد زیر اجراء می‌شود چیست؟ (مهندسی کامپیوتر - آزاد ۸۰)

```
for    i: = 1    to    n    do
     for    j:= i     to    n      do
begin
        k: = j   ;      r: =1;
        while (r < k)  do
        begin
                x : = x + 1 ;
                r : = r + 1 ;
         end;
end;
```

۱) $\frac{n^4-9n^2-8n}{6}$ ۲) $\frac{n^3-n}{3}$ ۳) $\frac{n^3-9n^2-10n}{6}$ ٤) $\frac{n^3-11n^2+9n}{6}$

۸- برای محاسبه عبارت زیر، حداقل چند عمل ضرب می‌توان استفاده کرد؟ (مهندسی کامپیوترـ دولتی ۸۱)

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

۱) n ۲) n^2 ۳) $n+\frac{n(n+1)}{2}$ ٤) $n+\frac{n(n-1)}{2}$

۹- تعداد پرانتزگذاری یک عبارت جبری با n عملوند (operand) که مقدار آن تغییر نکند، چند است؟ (علوم کامپیوتر ـ دولتی ۸۵)

۱) $n!$ ۲) $C_n = \frac{1}{n+1}\binom{2n}{n}$

۳) $C_{n-1} = \frac{1}{2}\binom{2(n-1)}{n-1}$ ۴) مشخص نمی‌باشد.

۱۰- با فرض این که تعداد عناصر موجود در آرایه n باشد و بخواهیم عنصر x را در آرایه جستجو کنیم و عنصر x در آرایه وجود داشته باشد، در این صورت پیچیدگی زمانی حالت میانی جستجو کدام است؟ (تخصصی هوش مصنوعی ـ آزاد ۸۶)

۱) $\frac{n}{2}$ ۲) $\frac{n-1}{2}$ ۳) $\log_2 n$ ۴) $\frac{n+1}{2}$

۱۱- آرایه‌ای با تعداد عناصر n مفروض است. می‌خواهیم عنصر x را در آرایه جستجو نماییم. اگر عنصر x با احتمال $\frac{1}{2}$ در آرایه موجود باشد، در این صورت به طور میانگین چه تعداد از عناصر آرایه بایستی مورد جستجو قرار گیرد؟ (مهندسی IT ـ آزاد ۸۶)

۱) $\frac{1}{2}$ ۲) $\frac{3}{4}$ ۳) $\frac{2}{3}$ ۴) $\frac{3}{5}$

۱۲- پیچیدگی زمانی دو الگوریتم A و B به ترتیب برابر $A(n)=\frac{n^2}{4}$ و $B(n)=25n$ می‌باشد. حداکثر به ازاء کدام n استفاده از پیچیدگی زمانی $A(n)$ کاراتر خواهد بود؟ (مهندسی IT ـ آزاد ۸۷)

۱) $n <= 50$ ۲) $n <= 150$ ۳) $n <= 100$ ۴) $n <= 150$

۱۳- در کد زیر دستور x++ چند بار تکرار می‌شود؟ فرض کنید n >= 3 است. (مهندسی IT ـ دولتی ۸۷)

```
for (i=3;  i<= n; i = i *2)
x ++;
```

۱) $\left\lceil \log \frac{n}{3} \right\rceil$ ۲) $\lceil (\log n)/3 \rceil$ ۳) $\left\lfloor \log \frac{n}{3} + 1 \right\rfloor$ ۴) $\lfloor (\log n + 1)/3 \rfloor$

۱۴- با اجرای قطعه کد زیر چند بار کلمه Test نمایش داده می‌شود؟ (مهندسی کامپیوتر ـ آزاد ۸۹)

```
for (m = 1;   m <= 4 ; m ++)
   for (n = 1;  n <= 4;  n++)
      for (k = m;  k <= n;  k ++)
         cout << "Test \n";
```

۱) 25 ۲) 16 ۳) 20 ۴) 26

۱۵- دنباله‌ای از 2^n عمل بر روی داده ساختاری انجام می‌شود. هزینه‌ی عمل i ام برابر i است اگر i توانی از 2 باشد، وگرنه برابر 1 است. میانگین هزینه یک عمل دلخواه (یعنی مجموع هزینه‌ها تقسیم بر تعدادشان) به کدام گزینه نزدیک‌تر است؟ (مهندسی کامپیوتر ـ دولتی ۸۹)

۱) 2 ۲) n ۳) 3 ٤) 2n

۱۶- رویه زیر را در نظر بگیرید : (مهندسی کامپیوتر - دولتی ۸۳)

```
Function power (x,k)
    y ← x
    i ← k
    a ← 1
    while i > 0
        do if i is odd
                then a ← a×y
            y ← y × y
            i ← ⌊i/2⌋
    return a
```

این الگوریتم قرار است مقدار x^k را محاسبه کند. مقدار مستقل از حلقه (Loop invariant) این الگوریتم چیست؟ یعنی چه عبارتی همواره در ابتدای حلقه درست است؟

۱) $y^i a = x^k$ ۲) $y^i a^i = x^k$ ۳) $a(ay)^i = x^k$ ٤) $y(ay)^i = x^k$

۱۷- با توجه به قطعه برنامه زیر کدام گزینه صحیح است؟ n را توانی از 2 فرض کنید.

(علوم کامپیوتر ـ دولتی ۸٦)

```
y = m ; i = n; x = 1;
while (i ≠0) {
    if ⌊i/2⌋ ≠ i then x = x * y;
    y = y * y;
    i = ⌊i/2⌋ ;
}
```

۱) $x = m^{2n-1}$ ۲) $y = m^n$ ۳) $m = y^n$ ٤) $m = x^n$

۱۸- شمار فراوانی کد زیر در بدترین حالت چیست؟ (مهندسی فناوری اطلاعات ـ آزاد ۸۵)

```
i :=1   ;   j :=n ;
repeat
  k := (i + j) / 2
  if a[k] < =x    then    i :=k+1
  else j :=k-1
until   i>j
```

۱) $\lceil \log(n+1) \rceil$ ۲) $2+3\lceil \log(n+1) \rceil$ ۳) $\log n$ ۴) $2+4\lceil \log(n+1) \rceil$

۱۹- دستور x++ در قطعه برنامه زیر چند بار اجرا می‌گردد؟

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

۱) $\lceil \log_2 n \rceil$ ۲) $\lfloor \log_2 n \rfloor$ ۳) $\left\lfloor \frac{\log(n+1)}{2} \right\rfloor$ ۴) $n-1$

تست‌های مرتبه اجرایی

۲۰- کدامیک از مجموعه توابع زیر بر حسب افزایش مرتبه (order) از چپ به راست مرتب هستند؟ (علوم کامپیوتر ـ دولتی ۷۹)

۱) $(1.005)^n, n^{1000}, n!$ ۲) $n^{1000}, (1.005)^n, n!$ ۳) $(1.005)^n, n!, n^{1000}$ ۴) $n^{1000}, n!, (1.005)^n$

۲۱- مرتبه بزرگی (order of magnitude) قطعه کد زیر چیست؟(مهندسی کامپیوتر ـ آزاد ۸۰)

```
k:=0;
for i:=1 to n do begin
     for j:=1 to m do
          k:=k+1;
     j:=1;
     while j<n  do begin
           k:=k+1;
           j:=2 *j
    end;
end;
```

۱) $\theta(n^2)$ ۲) $\theta(nm)$ ۳) $\theta(nm+n^2)$ ۴) $\theta(nm+nlogn)$

۲۲- پیچیدگی زمانی قطعه برنامه زیر چیست؟ (مهندسی کامپیوتر ـ آزاد ۷۷)

```
for  i:=1  to n do
   for j:=1 to i do
      for k:=1 to n do
          x:=x+1;
```

۱) $O(n^2)$ ۲) $O(n^3)$ ۳) $O(2^n)$ ۴) $O(n^2 \log^2 n)$

۲۳- درجه الگوریتم زیر چیست؟(مهندسی کامپیوتر - آزاد ۸۲)

```
For  i ← 0 to n do
{
    j ← i;
    while j ≠ 0 do
          j ← j div 2
    }
```

۱) n^2 ۲) $nlogn$ ۳) $n + logn$ ۴) n

۲۴- کدام عبارت صحیح است؟ (علوم کامپیوتر ـ دولتی ۸۲)

۱) $(n+1)(n^2-2n+1) \in O(2^n)$ ۲) $(n+1)(n^2-2n+1) \in \theta(n)$

۳) $(n+1)(n^2-2n+1) \in \Omega(n^4)$ ۴) $(n+1)(n^2-2n+1) \in O(n^2 \log n)$

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

الف : $n! = O(n^n)$ ب : $\frac{n^2}{\log n} = \theta(n^2)$

ج : $n^2 \log n = \theta(n^2)$ د : $n^{2^n} + 6(2^n) = \theta(n^{2^n})$

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

۲۶- کدام یک از عبارات زیر غلط است؟ (مهندسی کامپیوتر - دولتی ۸۱)

۱) $10^n + n^{20} \notin \theta(n^n)$ ۲) $\log_2^n \in \theta(\log_{10}^n)$

۳) $(\log_2^n)! \in \Omega(n!)$ ۴) $4n^3 + 5n^2 + 7n \in \Omega(\log_2^n)$

۲۷- 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++;
```

۱) $O(n^3)$ ۲) $O(n)$

۳) $O(n\log n)$ ۴) $O(n^2)$

۲۸- کدامیک از گزینه‌های زیر غلط است؟ (مهندسی کامپیوتر - دولتی ۷۹) (مهندسی کامپیوتر ـ آزاد ۷۹)

۱) if $f(n) = O(g(n))$ and $f(n) = \theta(g(n))$ then $f(n) = \Omega(g(n))$

۲) if $f(n) = \Omega(g(n))$ and $f(n) = \theta(g(n))$ then $f(n) = O(g(n))$

۳) if $f(n) = \Omega(g(n))$ and $f(n) = O(g(n))$ then $f(n) = \theta(g(n))$

۴) if $f(n) = O(g(n))$ and $g(n) = \Omega(f(n))$ then $f(n) = \theta(g(n))$

۲۹- فرض کنید f و g دو تابع دلخواه به شکل $f, g : N \to R^+$. بعلاوه فرض کنید : $\lim_{n\to\infty} \frac{f(n)}{g(n)} = +\infty$.

کدام یک از گزینه‌های زیر درست است؟ (مهندسی کامپیوتر- دولتی ۸۲)

۱) $f(n) \in O(g(n)), g(n) \notin \Omega(f(n))$ ۲) $g(n) \in O(f(n)), f(n) \in O(g(n))$

۳) $f(n) \in \Omega(g(n)), f(n) \notin \theta(g(n))$ ۴) $f(n) \in \theta(g(n)), g(n) \notin \Omega(f(n))$

۳۰- فرض کنید $f(n) = O(g(n))$، کدام یک از گزینه‌های زیر برای هر تابع با مقادیر مثبت مانند $g(n)$ و $f(n)$ برقرار است؟ (مهندسی کامپیوتر - دولتی ۸۳)

۱) $g(n) = \Omega(f(n))$

۲) $2^{f(n)} = O(2^{g(n)})$

۳) $2^{f(n)} = \Omega(2^{g(n)})$

۴) $\log(g(n)) = O(\log(f(n)))$ با فرض: $\log(g(n)) \geq 1$

۳۱- مرتبه زمانی شبه‌کد زیر چیست؟ (مهندسی IT ـ دولتی ۸٤)

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

۱) n ۲) n^2 ۳) $\log n$ ٤) $n \log n$

۳۲- پیچیدگی زمانی اجرای حلقه زیر چه می‌باشد؟ (علوم کامپیوتر ـ دولتی ۸٤)

while $(n > 0)$ $\{n = \frac{n}{10}\}$

۱) $O(1)$ ۲) $O(\frac{n}{10})$ ۳) $O(n)$ ٤) $O(\log n)$

۳۳- کدام گزینه صحیح می‌باشد؟ (علوم کامپیوتر ـ دولتی ۸٤)

۱) $0 < \varepsilon < 0.1 \quad n^3 \log n = O(n^{3+\varepsilon})$ ۲) $\sqrt{n} = O(\log n)$

۳) $0 < \varepsilon < 0.1 \quad n^{1+\varepsilon} = O(n \log n)$ ٤) $n^2 = O(\frac{n^2}{\log n})$

۳٤- گزینه درست را انتخاب کنید. (علوم کامپیوتر ـ دولتی ۸۰)

۱) $f(n) \in O(g(n)), h(n) \in \Omega(g(n)) \Rightarrow f(n) \in \theta(h(n))$

۲) $f(n) \in \theta(g(n)), g(n) \in O(h(n)) \Rightarrow h(n) \in O(f(n))$

۳) $f(n) \in \Omega(g(n)), h(n) \in O(g(n)) \Rightarrow f(n) \in \theta(g(n))$

٤) $f(n) \in \Omega(g(n)), g(n) \in \theta(h(n)) \Rightarrow f(n) \in \Omega(h(n))$

۳۵- کدام یک از عبارات زیر صحیح است؟ (علوم کامپیوتر ـ دولتی ۸۱)

۱) $\sum_{i=0}^{n} i^2 = O(n^3)$ ۲) $3^n = O(2^n)$

۳) $n^2 \log n = O(n^2)$ ٤) $n^2 / \log n = \Theta(n^2)$

۳٦- کدامیک از روابط ذیل درست است؟ (مهندسی کامپیوتر ـ آزاد ۸۱)

۱) $O(\log n) < O(\sqrt{n})$ ۲) $O(\log n) > O(\sqrt{n})$

۳) $O(n!) < O(a^n)$ ٤) $O(\sqrt{n^3}) < O(n)$

۳۷- میزان زمان لازم برای اجرای قطعه برنامه ز یر چگونه برآورد می‌شود؟ (مهندسی IT ـ دولتی ۸۳)

```
for  i ← 1 to n
    for j ← n   to  i
        for  k ← 1 to n^2
            sun ← sum + A
```

۱) $O(n^3)$ ۲) $\theta(n^3)$ ۳) $\Omega(n^3)$ ٤) $\theta(n^4)$

۳۸- کدام یک از عبارات زیر درست‌اند : (مهندسی کامپیوتر ـ دولتی ۸۵)

$e^{c\sqrt{n}} = O(e^{\sqrt{n}}).I \quad C \geq 1$

$n^2 = O(n \log n).II$

$n = O(n \log n).III$

۱) فقط II ۲) فقط III ۳) I و II ۴) I و III

۳۹- پس از اجرای قطعه کد زیر، مقدار نهایی x چه مقداری خواهد بود؟ (مهندسی IT ـ دولتی ۸۵)

```
x = 0 ;
for (i = 1 ; i <= n ; i ++) {
      for (j = 1 ; j <= n ; j ++)
            x ++ ;
      j = 1 ;
      while (j < n) {
            x ++ ;       j = j * 2 ;
      }
}
```

۱) $n.\lceil \log_2 n \rceil$ ۲) $n^2 + \lceil \log_2 n \rceil$

۳) $n^2 + n\lceil \log_2 n \rceil$ ۴) $n(1 + \lfloor \log_2 n \rfloor)$

۴۰- در رشد توابع زیر کدام ترتیب صحیح می‌باشد؟ (علوم کامپیوتر ـ دولتی ۸۵)

۱) $O(n \log n), O(1+\varepsilon)^n, O\left(\frac{n^2}{\log n}\right)$ ۲) $O(1+\varepsilon)^n, O(n \log n), O\left(\frac{n^2}{\log n}\right)$

۳) $O\left(\frac{n^2}{\log n}\right), O(n \log n), O(1+\varepsilon^n)$ ۴) $O(n \log n), O\left(\frac{n^2}{\log n}\right), O(1+\varepsilon)^n$

۴۱- کدام گزینه صحیح است؟ (علوم کامپیوتر ـ دولتی ۸۲)

۱) $n^a = O((\log n)^b)$ اگر $a > b$ ۲) $n^a \neq O((\log n)^b)$ اگر $a > b$

۳) $n^a = O((\log n)^b) \quad \forall b, a > 0$ ۴) $n^a \neq O((\log n)^b) \quad \forall b, a > 0$

۴۲- توابع $f(n) = 4^{\log n}$ ، $g(n) = (\log n)^{\log n}$ و $h(n) = \log^2 n$ را در نظر می‌گیریم. کدام یک از گزاره‌های زیر صحیح است؟ (مهندسی کامپیوتر ـ دولتی ۸۵ و دولتی ۸۹)

۱) $f(n) \in O(g(n)), f(n) \in \Omega(h(n))$ ۲) $g(n) \in \Omega(h(n)), h(n) \in \Omega(f(n))$

۳) $f(n) \in (h(n)), g(n) \in \Omega(f(n))$ ۴) $h(n) \in O(g(n)), f(n) \in \theta(g(n))$

۴۳- مرتبه زمانی الگوریتم زیر چیست؟ (علوم کامپیوتر- دولتی ۸۵ و مشابه IT ـ دولتی ۸۷ و آزاد ۹۱)

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

۱) $\theta(n)$ ۲) $\theta(n^2)$ ۳) $\theta(n^3)$ ۴) $\theta(n \log n)$

۴۴- مرتبه زمانی قطعه کد زیر چیست؟ (مهندسی کامپیوتر ـ آزاد ۸۵)

```
i := 2
while  i <= n  do
begin
    i := i^2
    x := x+1
end
```

۱) $\theta(\log n)$ ۲) $\theta(\log(\log n))$ ۳) $\theta(n)$ ۴) $\theta(n \log n)$

۴۵- کدام یک از تساوی‌های زیر درست است؟ (مهندسی کامپیوتر ـ آزاد ۸۵)

۱) $n^2 \log n = \theta(n^2)$ ۲) $6n^3 /(\log n + 1) = O(n^3)$

۳) $n^2 / \log n = \theta(n^2)$ ۴) $n^3 2^n + 6n^2 3^n = O(n^3 2^n)$

۴۶- مرتبه زمانی قطعه کد زیر چیست؟ (مهندسی IT ـ آزاد ۸۵)

```
j := n
while j ≥ 1  do
begin
    for  i := 1  to  n  do
         x := x + 1
    j := ⌊j / 2⌋
end
```

۱) $\theta(n \log n)$ ۲) $\theta(\log n)$ ۳) $\theta(n)$ ۴) $\theta(n^2)$

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

۱) $n^{\frac{1}{10}} \in \Omega(\log n)$ ۲) $\log(n!) \in O(\log n)$

۳) $8n^2 + 3n - 4 \in O(n \log n)$ ۴) $1^2 + 2^n + ... + n^n \in O(n^n)$

۴۸- با توجه به تعاریف علامات θ، O و Ω کدام یک از گزینه‌های زیر غلط است : (مهندسی کامپیوتر آزاد ۸۰ و مشابه آزاد ۷۸)

۱) $O(\sum_{i=1}^{m} f_i(n)) = O(\underset{i=1}{\overset{m}{Max}}\{f_i\})$

۲) if $g(n) \le C$ for $n \ge n_0$ then $g(n) \cdot f(n) = \theta(f(n))$

۳) $O(f(n) + g(n)) = O(f(n)) + O(g(n))$

۴) if $f(n) = O(g(n))$ and $g(n) = \Omega(f(n))$ then $f(n) = \theta(g(n))$

۴۹- مرتبه زمانی شبه کد زیر چیست؟ (مهندسی IT ـ دولتی ۸۶)

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

۱) n ۲) $n \wedge 2$ ۳) $\log n$ ۴) $n \log n$

۵۰- کامپیوتری در واحد زمان مسأله‌ای به اندازه ۱۶ را که الگوریتم آن از مرتبه زمانی $n2^n$ است حل می‌کند. اگر سرعت کامپیوتر ۱۳۱۰۷۲ برابر گردد این کامپیوتر همان مسأله را با چه اندازه‌ای در واحد زمان حل خواهد کرد؟ (مهندسی کامپیوتر ـ دولتی ۸۶)

۱) $16 + 17 + \log 17$ ۲) $16 \times 17 \times \log 17$

۳) 32 ۴) $16 + \log 131072$

۵۱- کدام یک از موارد زیر نادرست است؟ (مهندسی کامپیوتر ـ آزاد ۸۶)

۱) $6*2^n + n^2 = \theta(n^2)$ ۲) $6*2^n + n^2 = O(2^n)$

۳) $6*2^n + n^2 = \Omega(2^n)$ ۴) $6*2^n + n^2 = \Omega(n^2)$

۵۲- یک نماد θ برای دفعات اجرای دستور $x := x + 1$ در قطعه کد زیر کدام است؟ (مهندسی IT ـ آزاد ۸۶)

```
i := n
  while i ≥ 1  do
    begin
      for j := 1 to n  do
          x := x + 1
       i : = ⌊i/3⌋
    end
```

۱) $\theta(n \log_2 n)$ ۲) $\theta(n)$ ۳) $\theta(n \log_3 n)$ ۴) $\theta(\log_3 \log_2 n)$

۵۳- زمان‌های اجرای چهار الگوریتم مختلف به صورت زیر مشخص شده است. کدام یک دارای پیچیدگی زمانی بیشتری است؟ (مهندسی IT ـ آزاد ۸۶)

۱) $O(n^3)$ ۲) $O(n^2)$ ۳) $O(2^n)$ ۴) $O(n \log n)$

۵۴- تعداد دستورات اجرایی یک الگوریتم با تابع $f(n) = \sum_{i=1}^{n} 2i^3$ شمارش شده است. پیچیدگی زمانی این الگوریتم چیست؟ (علوم کامپیوتر ـ دولتی ۸۷)

۱) $O(n^3)$ ۲) $O(n^4)$ ۳) $O(n^3 \log n)$ ۴) $O(n^3 + \log n)$

۵۵- با فرض آن که $n = m$ باشد، پیچیدگی زمانی الگوریتم زیر کدام است؟ (مهندسی IT ـ آزاد ۸۷)

```
for   i := 1  to  n  do
    for  j := 1   to m  do
         for  k := 1  to  j   do
              a := a +3 ;
```

۱) $O\left(\frac{m(m+1)}{2}\right)$ ۲) $O\left(\frac{n+1}{2}\right)$ ۳) $O(n^3)$ ۴) $O(n^2)$

۵۶- مرتبه زمانی شبه کد زیر چیست؟ (مهندسی IT ـ دولتی ۸۸)

```
for (i=1; i ≤ n; i = i * 2)
  for (j = 1; j ≤ n; j = j * 2)
     for (k=1; k ≤ j; k ++)
          x++;
```

۱) $\theta(n)$ ۲) $\theta(n^2)$ ۳) $\theta(n\log n)$ ۴) $\theta\left(n(\log n)^2\right)$

۵۷- یک نماد θ برای زمان بدترین حالت در الگوریتم زیر چیست؟ (مهندسی کامپیوتر ـ آزاد ۸۸)

```
Procedure  iskey(s, n, key)
  for i := 1  to  n-1  do
    for  j := i + 1 to n   do
       if  s_i + s_j = key  then
          return(1)
       else
          return(0)
end iskey
```

۱) $\theta(n)$ ۲) $\theta(1)$ ۳) $\theta(n^2)$ ۴) $\theta(\log n)$

۵۸- کدام یک از گزینه‌های زیر صحیح نمی‌باشد؟ (مهندسی IT ـ آزاد ۸۹)

۱) $\frac{n^2}{\log n} = \theta(n^2)$ ۲) $\left(1+\frac{3}{\log n}\right)n^3 = O(n^3)$

۳) $10n^3 + 3n^2 = \Omega(n^2)$ ۴) $10n^2 2^n + n\log n = \theta(n^2 2^n)$

۵۹- پیچیدگی زمانی الگوریتم خاصی به‌صورت چندجمله‌ای زیر معین شده است، در این‌صورت زمان اجرای آن کدام است؟ (فرض کنید **n** بسیار بزرگ) $P(n) = 2^n + 100n^3 + n\log_2^n$ (مهندسی IT ـ آزاد ۸۹)

۱) $O(100n^3)$ ۲) $O(n\log_2^n)$ ۳) $O(2^n + n^3)$ ۴) $O(2^n)$

۶۰- پیچیدگی زمانی چند جمله‌ای زیر کدام است؟ (مهندسی کامپیوتر ـ آزاد ۸۹)

$$P(n) = 5n^2 + 100n + 200000$$

(فرض کنید **n** بسیار بزرگ است)

۱) $O(5n^2)$ ۲) $O(n^2)$ ۳) $O(100n)$ ۴) $O(200000)$

۶۱- اگر $f(n) = (\lceil \log_2 n \rceil)!$ ، $g(n) = \log n!$ ، $h(n) = \sqrt[3]{n^4 + 3n}$ آن‌گاه کدام گزینه صحیح است؟ (مهندسی کامپیوتر ـ آزاد ۸۹)

۱) $h(n) = o(g(n)), f(n) = o(h(n))$ ۲) $h(n) = o(f(n)), g(n) = o(h(n))$

۳) $g(n) = o(f(n)), h(n) = o(g(n))$ ۴) $g(n) = o(h(n)), f(n) = o(g(n))$

۶۲- پیچیدگی زمانی چند جمله‌ای زیر کدام است؟ (مهندسی کامپیوتر ـ آزاد ۸۹)

$P(n) = n! + 2^n + 1000n^{10}$

(فرض کنید n بسیار بزرگ است)

۱) $O(2^n)$ ۲) $O(1000n^{10})$ ۳) $O(n!)$ ۴) $O(n! + 2^n)$

۶۳- کدام یک از گزاره‌های زیر غلط است؟ (مهندسی IT ـ دولتی ۸۹)

۱) $f(n) + O(f(n)) = \Theta(f(n))$ ۲) $g(n) = \Omega(f(n)) \Rightarrow g(n) = \Omega(O(f(n)))$

۳) $g(n) = \Omega(f(n)) \Rightarrow f(n) = O(g(n))$ ۴) $\Omega(f(n)) + O(f(n)) = \Theta(f(n))$

۶۴- کدام یک ترتیب صحیح از کوچکترین نرخ رشد تا بزرگترین نرخ رشد است؟ (مهندسی ـ دولتی ۸۴)

۱) $x \operatorname{Log} x \ , 16x^2 \ , (3x)^{3.2} \ , x^{3.2} \operatorname{Log}\sqrt{x}$

۲) $x \operatorname{Log} x \ , 16x^2 \ , x^{3.2} \operatorname{Log}\sqrt{x} \ , (3x)^{3.2}$

۳) $16x^2 \ , x \operatorname{Log} x \ , (3x)^{3.2} \ , x^{3.2} \operatorname{Log}\sqrt{x}$

۴) $16x^2 \ , x \operatorname{Log} x \ , x^{3.2} \operatorname{Log}\sqrt{x} \ , (3x)^{3.2}$

۶۵- گزینه صحیح را انتخاب کنید. (علوم کامپیوتر ـ دولتی ۷۹)

۱) اگر $f_1(n), f_2(n), \ldots, f_n(n) \in \theta(g(n))$ آنگاه $\sum_{i=1}^{n} f_i(n) \in \theta(g(n))$

۲) اگر $f(n) \notin O(g(n))$ آنگاه $g(n) \in O(f(n))$

۳) اگر $f(n) \in O(g(n))$ و $g(n) \in \theta(h(n))$ آنگاه $h(n) \in \Omega(f(n))$

۴) اگر $f(n) \in O(n)$ آنگاه $2^{f(n)} \in O(2^n)$

۶۶- کدام یک از گزینه‌های زیر صحیح است؟ (IT ـ دولتی ۸۴)

۱) $n^2 |\sin n| \in \Omega(n)$ ۲) $n^2 \sin n \in O(n)$

۳) $n \in \theta(n^2 \sin n)$ ۴) $n \in \Omega(n^2 \sin n)$

۶۷- در یک الگوریتم، چند مرحله اول از پیچیدگی $O(n)$، چند مرحله بعدی از پیچیدگی $O(n^4)$ و چند مرحله آخر از پیچیدگی $O(n^2)$ است. پیچیدگی کل الگوریتم چقدر است؟ (علوم کامپیوتر ـ آزاد ۸۶)

۱) $O(n)$ ۲) $O(n^3)$ ۳) $O(n^4)$ ۴) $O(n^7)$

۶۸- یک آرایه از اعداد صحیح به صورت A[1..m] مفروض است، به طوری که $\sum_{i=1}^{m} A[i] = S$ می‌باشد. در این صورت درجه اجرای الگوریتم زیر کدام یک از گزینه‌ها است؟(مهندسی آزاد ۸۱ و مشابه IT آزاد ۸۶)

```
T := 0 ;
for i :=1   to  m  do
   for j :=1 to  A[i]  do
     T :=T+1 ;
```

۱) $O(s^2)$ ۲) $O(m+s)$ ۳) $O(m^2)$ ۴) $O(ms)$

۶۹- برای $k>1$ و $\varepsilon<\frac{1}{2}$ کدام گزینه صحیح است؟ **(علوم کامپیوتر ـ دولتی ۸۸)**

۱) $n^{\varepsilon}=O(\sqrt{n})=O(\lg n!)=O((\lg n)^k)$

۲) $n^{\varepsilon}=O(\sqrt{n})=O((\lg n)^k)=O(\lg n!)$

۳) $(\lg n)^k=O(n^{\varepsilon})=O(\lg n!)=O(\sqrt{n})$

۴) $(\lg n)^k=O(n^{\varepsilon})=O(\sqrt{n})=O(\lg n!)$

۷۰- با فرض مثبت بودن توابع f و g ، تعداد گزاره‌های درست از میان گزاره‌های زیر چند است؟

(مهندسی IT ـ دولتی ۸۸)

- **گزاره ۱:** $f(n)=O\left((f(n))^2\right)$
- **گزاره ۲:** $f(n)+o(f(n))=\theta(f(n))$
- **گزاره ۳:** $f(n)=O(g(n))\Rightarrow 2^{f(n)}=O(2^{g(n)})$
- **گزاره ۴:** $f(n)=\theta(f(n/2))$

۱) یک گزاره صحیح است. ۲) دو گزاره صحیح است.

۳) سه گزاره صحیح است. ۴) گزاره‌ای صحیح نیست.

۷۱- کدام گزینه صحیح است؟ **(علوم کامپیوتر ـ دولتی ۸۹)**

۱) $(0<x<1)\ \frac{n}{\lg n}=O(n^{1-x})\ ,\ 3^n=\Omega(n2^n)$

۲) $n(\log_3 n)^5=\Omega(n^{1.2})\ ,\ \sqrt{n}=O((\lg n)^5)$

۳) $\frac{n}{\lg n}=\Omega(n^{1-x})\ 0<x<1\ ,\ 3^n=O(n2^n)$

۴) $n(\log_3 n)^5=O(n^{1.2})\ ,\ \sqrt{n}=\Omega((\lg n)^5)$

۷۲- کدام یک از گزاره‌های زیر غلط است؟ **(مهندسی IT ـ دولتی ۹۰)**

۱) $f(n)=O(f(n))=\Theta(f(n))$

۲) $g(n)=\Omega(f(n))\Rightarrow g(n)=\Omega(O(f(n)))$

۳) $f(n)+g(n)=O(\max\{f(n),g(n)\})$

۴) $g(n)\neq O(f(n))\Rightarrow g(n)=\Omega(f(n))$

۷۳- در صورتی که $n>100$ باشد، کدام گزینه دارای بیشترین پیچیدگی محاسباتی است؟

(مهندسی کامپیوتر ـ آزاد ۹۰)

۱) $O(n^2+100n)$ ۲) $O(2^n)$ ۳) $O(n\log_2^n)$ ۴) $O(10n^3+80n^2)$

۷۴- زمان اجرای الگوریتمی برابر با $\sum_{i=1}^{n} i^p$ است. مرتبه زمانی آن کدام یک از موارد زیر است؟

(مهندسی کامپیوتر ـ آزاد ۹۰)

۱) $\theta\left(n^p\right)$ ۲) $\theta\left(n^{p-1}\right)$ ۳) $\theta\left(n^{p+1}\right)$ ۴) $\theta\left(n^p \log n\right)$

۷۵- هزینه‌ی زمانی تکه برنامه‌ی زیر کدام است؟

(مهندسی IT ـ دولتی ۹۱)

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

۱) $O(\lg n)$ ۲) $O(\lg^2 n)$ ۳) $O(n)$ ۴) $O(n^2)$

۷۶- فرض کنید که میانگین زمان اجرای یک الگوریتم تصادفی **A** بر روی ورودی‌های به اندازه‌ی **n** برابر $\theta(n^2)$ است. چند تا از گزاره‌های زیر درست هستند؟

(مهندسی نرم‌افزار ـ دولتی ۹۲)

ممکن است داده‌ی ورودی‌ای باشد که **A** آن را در زمان $\Omega(n^3)$ حل کند.

ممکن است داده‌ی ورودی‌ای باشد که **A** آن را در زمان $\theta(n)$ حل کند.

ممکن است داده‌ی ورودی‌ای باشد که **A** آن را در زمان **O(1)** حل کند.

۱) 0 ۲) 1 ۳) 2 ۴) 3

پاسخ تست‌های فصل اول

پاسخ پیچیدگی زمانی

۱- (۲)

تغییرات i	تغییرات K
1,2,3,...,n	0
1,2,3,...,n-1	1
1,2,3,...,n-2	2
⋮	⋮
1	n-1

$\Rightarrow$ تعداد اجراء خط 3 $= 1 + 2 + 3 + 4 + ... + n = \frac{n(n+1)}{2}$

راه حل دوم:

$$\sum_{k=0}^{n-1}\sum_{i=1}^{n-k}1 = \sum_{k=0}^{n-1}(n-k) = \sum_{k=0}^{n-1}n - \sum_{k=0}^{n-1}k = n\sum_{k=0}^{n-1}1 - \frac{n(n-1)}{2} = n^2 - \frac{n(n-1)}{2} = \frac{n(n+1)}{2}$$

۲- (۳)

```
for (j = 0; j < m; j++)                    m + 1
{
    s[n - 1][j] = 0;                       m × (1
    for (i =0; i < n - 1; i++)                n
        s[n-1][j]+ = s[i][j];                 n - 1)
 }
```

تعداد کل مراحل اجرا $= m + 1 + m (1 + n + n - 1) = m + 1 + 2 mn = m (2n + 1) + 1$

۳- (۲) مسأله را با مثال حل می‌کنیم. فرض کنید N=4 باشد :

I	K	شرط K<>1	TOTAL
1	4	T	1
	2	T	2
	1	F	
2	4	T	3
	2	T	4
	1	F	
3	4	T	5
	2	T	6
	1	F	
4	4	T	7
	2	T	8
	1	F	

تنها گزینه‌ای که به ازای N=4 جواب 8 می‌دهد گزینهٔ ۲ می‌باشد. البته برای آزمایش ساده‌تر این است که به ازای N=1 جواب Total = 0 را بررسی کنیم.

تذکر: در حلقه درونی while متغیر k که دارای مقدار اولیه N است مرتب تقسیم بر ۲ می‌شود پس تعداد اجرای این حلقه فاکتور $\log_2 N$ را دارد. از طرف دیگر حلقه FOR بیرونی نیز N بار اجرا می‌شود. پس دستور TOTAL := TOTAL + 1; به تعداد حدود $N\text{Log}_2N$ بار اجراء می‌شود و بدین ترتیب گزینهٔ ٤ حتماً غلط است.

٤ـ (۳) مثلاً M را برابر ۵ بگیرید :

I	J
1	1,2,3,4
2	1,2,3
3	1,2
4	1
5	--

1 + 3 + 3 + 4 ⟵ جمع اعداد 1 تا M - 1 $= \frac{M(M-1)}{2}$

۵ـ (٤) حلقه داخلی K مستقل از دو حلقه دیگر بوده و به تعداد M بار اجراء می‌شود، پس ابتدا حلقه K را کنار گذاشته و فقط به دو حلقه بیرونی I و J توجه می‌کنیم (مثلاً M را برابر ٤ بگیرید)

I	J	تعداد اجراء
0	0	1
1	0,1	2
2	0,1,2	3
3	0,1,2,3	4
4	0,1,2,3,4	5

پس برای حالتی کلی M تعداد اجراء برابر است با :

$1+2+3+...+M+1 = \frac{(M+1)(M+2)}{2}$ (جمع تصاعد عددی)

از طرف دیگر چون حلقه داخلی K نیز M بار اجراء می‌شود پس در حالت کلی دستور X:=X + 1 به تعداد $\frac{M(M+1)(M+2)}{2}$ بار اجراء می‌شود.

٦- (٤) توجه کنید i حلقه داخلی ربطی به i حلقه خارجی ندارد. پس حلقه را به صورت زیر درنظر بگیرید:

```
for (i=0; i < N-i ; ++i)
    for (j=0 ;  j < N-j ; ++j)
```

یعنی حلقه‌ها مستقل از یکدیگرند. حال فرض کنید مثلاً N برای 4 باشد :

i	شرط i < 4 - i	تعداد اجرا
0	درست (0 < 4)	1
1	درست (1 < 3)	1
2	غلط (2 < 2)	-

حال اگر N برابر 5 باشد :

i	شرط i < 5-i	تعداد اجرا
0	درست (0 < 5)	1
1	درست (1 < 4)	1
2	غلط (2 < 3)	1
3	غلط (3 < 2)	-

پس تعداد اجرای هر حلقه برابر (N+1) div 2 می‌باشد. چون دو حلقه مستقل هستند پس تعداد اجرای آنها درهم ضرب می‌شود.

$$\left\lfloor\frac{(N+1)}{2}\right\rfloor\times\left\lfloor\frac{(N+1)}{2}\right\rfloor$$

۷- (۲) ابتدا فرض کنید n = 3 باشد، ببینیم دستور x: = x +1 درون سه حلقه تو در تو چند بار اجراء می‌شود. فرض کنید مقدار اولیه x برابر صفر باشد پس برای n =3 دستور مورد نظر ۸ بار اجراء می‌گردد. گزینه‌های ۱ و ۳ و ٤ به ازاء n = 3 منفی می‌شوند که اشتباه است. گزینه ۲ تقریباً همان ۸ می‌شود. از آنجا که برنامه از سه حلقهٔ تودر تو تشکیل شده پس مرتبه اجرائی آن $O(n^3)$ می‌باشد پس حتما گزینه ۱ غلط است چون مرتبه آن $O(n^4)$ است.

i	j	k	r	x
1	1	1	1	0
	2	2	1	1
			2	-
	3	3	1	2
			2	3
			3	-
2	2	2	1	4
			2	-
	3	3	1	5
			2	6
			3	-
3	3	3	1	7
			2	8
			3	-

راه دوم :

$$\sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{r=1}^{j-1}1=\sum_{i=1}^{n}\sum_{j=i}^{n}(j-1)=\sum_{i=1}^{n}(i)-1+(i+1)-1+(i+2)-1+\cdots(n)-1$$

$$=\sum_{i=1}^{n}\frac{(n-i+1)(i+n)}{2}-(n-i+1)=\sum_{i=1}^{n}\frac{n^2-i^2+i+n-2n+2i-2}{2}$$

$$= \sum_{i=1}^{n} \frac{n^2 - n - i^2 + 3i - 2}{2} = \frac{1}{2}\left[n^3 - n^2 - \frac{n(n+1)(2n+1)}{6} + \frac{3(n)(n+1)}{2} - 2n\right]$$

$$= \frac{1}{2}\left[\frac{6n^3 - 6n^2 - 2n^3 - n^2 - 2n^2 - n + 9n^2 + 9n - 12n}{6}\right] = \frac{1}{12}\left[4n^3 - 4n\right] = \frac{n^3 - n}{3}$$

۸- (۱) بنابر روش هرنر (Horner) مثلاً چندجمله‌ای $a_3x^3 + a_2x^2 + a_1x^1 + a_0$ را می‌توان به صورت روبرو محاسبه کرد : $x(x(a_3x + a_2) + a_1) + a_0$

که به ۳ ضرب نیاز دارد. پس در حالت کلی چند جمله‌ای درجه n به n ضرب نیاز خواهد داشت.

۹- (٤) تعداد این پرانتز گذاری بستگی به نوع عملگرها هم دارد. مثلاً داریم :

$((a+b)+c)$ معادل است $(a+(b+c))$

$((a*b)+c)$ معادل نیست $(a*(b+c))$

۱۰- (٤) در بهترین حالت یک مقایسه و در بدترین حالت n مقایسه و در حالت متوسط $\frac{n+1}{2}$ مقایسه لازم است.

۱۱- (۲) اگر احتمال وجود x در آرایه برابر p باشد، احتمال آنکه x در یکی از خانه‌های آرایه (مثل خانه i ام) باشد برابر $\frac{p}{n}$ است. احتمال آنکه x در آرایه نباشد برابر p - 1 است. اگر x در خانه i ام باشد دستور اصلی i بار اجرا می‌شود و اگر x در آرایه نباشد دستور اصلی n بار اجرا می‌شود، لذا :

$$A(n) = \left(\sum_{i=1}^{n} \frac{p}{n} \times i\right) + (1-p)n = \frac{p}{n} \times \frac{n(n+1)}{2} + n(1-p) = n\left(1 - \frac{p}{2}\right) + \frac{p}{2}$$

اگر p = 1 باشد $A(n) = \frac{n+1}{2}$ و اگر $p = \frac{1}{2}$ باشد $A(n) = \frac{3n}{4} + \frac{1}{4}$ می‌شود و این بدان معناست که حدود $\frac{3}{4}$ آرایه به طور میانگین باید جستجو شود.

۱۲- (۳)

$$\frac{n^2}{4} \le 25n \Rightarrow n \le 100$$

۱۳- (۳) حلقه را یکبار به ازای n = 18 و یک بار به ازای n = 24 اجرا کنید :

n	i	شرط i<=24	تعداد اجرا
24	3	بله	1
	6	بله	1
	12	بله	1
	24	بله	1
	48	خیر	

٤ بار = کل اجرا

n	i	شرط i<=18	تعداد اجرا
18	3	بله	1
	6	بله	1
	12	بله	1
	24	خیر	

۳ بار = کل اجرا

پس فرمولی را می‌خواهیم که به ازای $n = 18$ خروجی آن 3 و به ازای $n = 24$ خروجی آن 4 شود که این ویژگی را فرمول $\left\lfloor \log\frac{n}{3}+1 \right\rfloor$ دارد. گزینه ۱ غلط است. چرا که به ازای $n = 6$ حلقه باید 2 بار اجرا شود درحالی که در گزینه ۱ به ازای $n = 6$ فرمول $\left\lfloor \log\frac{n}{3} \right\rfloor$ برابر 1 می‌شود.

تذکر: به ازای n که توانی از 2 باشد حلقه $for(i = 2 ; i <= n ; i = i*2)$ به تعداد $\log_2 n$ بار اجرا می‌شود، حال که i از عدد 3 شروع شده یک واحد از این تکرار کم می‌شود همچنین توجه کنید که:

$$\left\lfloor \log_2\frac{n}{3}+1 \right\rfloor = \lfloor \log_2 n - \log_2 3 + 1 \rfloor = \log_2 n - 1$$

۱۴- (۳)

حلقه بیرونی m	حلقه درونی n	تغییرات k	مقدار اجرا
$m = 1$	$n = 1$ $n = 2$ $n = 3$ $n = 4$	از 1 تا 1 از 1 تا 2 از 1 تا 3 از 1 تا 4	1 2 3 4
$m = 2$	$n = 2$ $n = 3$ $n = 4$	از 2 تا 2 از 2 تا 3 از 2 تا 4	1 2 3
$m = 3$	$n = 3$ $n = 4$	از 3 تا 3 از 3 تا 4	1 2
$m = 4$	$n = 4$	از 4 تا 4	1

تعداد کل اجرا $(1+2+3+4)+(1+2+3)+(1+2)+1 = 20$

این برنامه پشت کامپیوتر آزمایش شد و بیست بار عبارت Test را نمایش داد.

۱۵- (۳) مثلاً برای $n = 3$ که $2^n = 8$ می‌شود :

$$\text{میانگین هزینه عمل} = \frac{(1+2+4+8)+(1+1+1+1)}{8} = \frac{(1+2+4+8)+(8-4)}{8}$$

برای $n = 4$ که $2^n = 16$ می‌شود :

$$\text{میانگین هزینه عمل} = \frac{(1+2+4+8+16)+(16-5)}{16}$$

برای $n=5$ که $2^n=32$ می‌شود :

$$\text{میانگین هزینه عمل} = \frac{(1+2+4+16+32)+(32-6)}{32}$$

و در حالت کلی برای n داریم :

$$\text{میانگین هزینه عمل} = \frac{\left(2^n-1+2^n\right)+\left(2^n-n-1\right)}{2^n} = \frac{3\times 2^n-n-2}{2^n}$$

که برای nهای بزرگ داریم :

$$\approx \frac{3\times 2^n}{2^n} = 3$$

۱۶- (۱) k را برابر 3 در نظر گرفته و الگوریتم را Trace می‌کنیم :

y	i	a	شرط
x	3	1	درست
x^2	1	x	درست
x^4	0	x^3	غلط

برای هر سه سطر می‌بینیم که رابطه $y^i a = x^3$ برقرار است. پس جواب گزینه (۱) می‌باشد.

۱۷- (۱) شرط $\text{if } \left\lfloor \frac{i}{2} \right\rfloor \neq i$ به شرطی درست است که $i \neq 0$ باشد، لذا برنامه به صورت زیر ساده‌تر می‌شود که با مقادیر $n=8$ و $n=16$ آن را آزمایش می‌کنیم:

```
y = m;    i = n;    x = 1;
while (i≠0)  {
    x = x * y;
    y = y * y;
    i = ⌊i/2⌋;
}
```

n	شرط $i \neq 0$	x	y	i
8	بله	m	m^2	4
	بله	m^3	m^4	2
	بله	m^7	m^8	1
	بله	m^{15}	m^{16}	0
	خیر			

n	شرط $i \neq 0$	x	y	i
16	بله	m	m^2	8
	بله	m^3	m^4	4
	بله	m^7	m^8	2
	بله	m^{15}	m^{16}	1
	بله	m^{31}	m^{32}	0
	خیر			

پس $y = m^{2n}$ و $x = m^{2n-1}$ می‌شود.

۱۸- (٤) به ازای n=1 آزمایش می‌کنیم:

تعداد	دستور
۱	$i := 1$
۱	$j := n = 1$
۱	$k := \frac{(i+j)}{2} = \frac{1+1}{2} = 1$
۱	$a[k] <= x$
۱	$i := k+1 = 2$ یا $j := k-1 = 0$
۱	$i > j$ شرط درست شده و حلقه تمام می‌شود.
٦	جمع خطوط اجرا شده

توجه کنید حلقه repeat-until تا وقتی که شرط جلوی until غلط است اجرا می‌شود.

۱۹- (۲) چون شمارنده تقسیم بر ۲ می‌شود پس در جواب $\log_2 n$ داریم که تا اینجا گزینه ٤ حتماً غلط است.

با $n = 14$ آزمایش می‌کنیم :

تعداد اجرای x++	شرط j>1	j
۱ بار	بله	14
۱ بار	بله	7
۱ بار	بله	3
	خیر	1

$n = 14 \Rightarrow$ جواب = 3

این حالت تنها در گزینه ۲ صدق می‌کند که :

$$\lfloor \log_2 n \rfloor = \lfloor \log_2 14 \rfloor = 3$$

پاسخ تست‌های مرتبه اجرایی

۲۰- (۲) در حالت کلی داریم :

$$a , b > 1 \Rightarrow n^b < a^n < n!$$

۲۱- (٤) حلقه for j به تعداد m بار و حلقه while j به تعداد log n بار انجام می‌گیرد و هر دوی اینها در حلقه for i به تعداد n بار اجراء می‌شوند. مرتبه دو حلقه زیر هم، با یکدیگر جمع می‌شوند. مرتبه دو حلقه درون یکدیگر، با همدیگر ضرب می‌شوند:

$$n (m + \log n) = nm + n \log n$$

۲۲- (۲) حلقه درونی n بار اجراء می‌شود تعداد اجرای دو حلقه بیرونی به صورت زیر است .

$$1+2+3+...+n = \frac{n(1+n)}{2} \Rightarrow$$

$$x := x+1 \quad \text{تعداد اجراء دستور} = n \times \frac{n(1+n)}{2} \Rightarrow O(n^3)$$

پس تعداد کل اجرا برابر ٤+٥+٦ است. در حالت کلی تعداد اجرا برابر اعداد n، n-1 ، n-2 تا .... تقریباً n/2 است. توجه کنید در حلقه بیرونی هر بار یک و احد به i اضافه شده و یک واحد از n کم می‌شود.

$$n + (n-1) + (n-2) + \ldots + \frac{n}{2} = \frac{(n+\frac{n}{2})(\frac{n}{2})}{2} = O(n^2)$$

۳۲- (٤) چون n هر بار تقسیم بر ۱۰ می‌شود پس مرتبه اجرایی $O(\log_{10} n)$ است. از طرفی دیگر $O(\log_a n) = O(\log_b n)$ لذا مرتبه اجرایی برنامه داده شده $O(\log n)$ بوده و مبنای لگاریتم آن مهم نیست.

۳۳- (۱) توابع n^ε که $\varepsilon > 0$، رشدشان از تابع $\log n$ بیشتر است. لذا گزینهٔ ۱ درست است و گزینه‌های ۲ و۳ و ٤ برعکس نوشته شده‌اند.

۳٤- (٤) چون $g(n) \in \theta(h(n))$ پس در عبارت $f(n) \in \Omega(g(n))$ می‌توان به جای $g(n)$ عبارت هم‌مرتبه آن یعنی $h(n)$ را قرار داد. پس گزینه ٤ درست است. مثلاً گزینه ۱ نادرست است، هنگامی که $g(n)$ حد بالای $f(n)$ باشد و $g(n)$ حد پائین $h(n)$ باشد آنگاه بدیهی است که $h(n)$ حد بالای $f(n)$ خواهد بود یعنی در گزینه ۱ سمت راست $\Rightarrow$ باید عبارت $f(n) \in O(h(n))$ باشد و الزاماً رابطه $f(n) \in \theta(h(n))$ درست نیست، یعنی:

$$h \geq g \ , \ g \geq f \Rightarrow f = g$$

۳٥- (۱)

۳٦- (۱) جهت نامساوی در گزینه‌های ۲و۳و٤ باید برعکس شود تا درست گردند. رشد n^ε، $\varepsilon > 0$ از رشد $\log n$ بیشتر است.

۳۷- (٤) حلقه اول و دوم هر کدام $\theta(n)$ و حلقه سوم از مرتبه $\theta(n^2)$ است، پس جواب $\theta(n^4)$ است.

۳۸- (۲) در عبارت III عبارت $n \log n$ از n بزرگتر است پس درست است ولی در عبارت II عبارت $n \log n$ از n^2 کوچکتر است لذا غلط است. اگر عبارت I را به صورت $e^{C\sqrt{n}} = (e^{\sqrt{n}})^C$ بنویسیم مشخص است که برای $C \geq 1$، $(e^{\sqrt{n}})^C$ نمی‌تواند از مرتبه $O(e^{\sqrt{n}})$ باشد.

۳۹- (۳)

```
for (   )   {
        for (   )        O(n)
            x + + ;
        while ( ) {      O(log n)
        }
}
```

$$O(n) \Rightarrow O(n.\ (n + \log n)) = O(n^2 + n\log n)$$

فقط گزینه ۳ از مرتبه فوق است.

۴۰- (٤) می‌دانیم رشد a^n که $a > 1$ می‌باشد، از تمامی توابع دیگرداده شده سریع‌تر است، پس گزینه ۱ و ۲ حتماً غلط هستند. رشد $\frac{n^2}{\log n}$ سریع‌تر از رشد $n \log n$ است چرا که :

$$\lim_{n\to\infty} \frac{\frac{n^2}{\log n}}{n\log n} = \lim_{n\to\infty} \frac{n}{\log^2 n} \xrightarrow{\text{هوپیتال}} \lim_{n\to\infty} \frac{1}{2\times\log n\times\frac{1}{n}} = \lim_{n\to\infty} \frac{n}{2\log n} = \infty$$

۴۱- (٤) گزینه ٤ درست است چرا که مرتبه توابع n^a هیچ‌گاه کمتر از توابع لگاریتمی مثل $(\log n)^b$ نیست. در واقع رشد توابع n^a از $(\log n)^b$ به ازای $a , b > 0$ همواره بیشتر است.

۴۲- (۱)

$$\left.\begin{aligned} f(n) &= 4^{\log n} = n^{\log 4} = n^2 \\ g(n) &= (\log n)^{\log n} = n^{\log\log n} \\ h(n) &= \log^2 n \end{aligned}\right\} \longrightarrow \begin{aligned} &h(n) \in O(f(n)) \in O(g(n)) \\ &h \le f \le g \end{aligned}$$

$$h(n) \in O(f(n)) \Rightarrow f(n) \in \Omega(h(n))$$

پس گزینه ۱ درست است.

۴۳- (٤) در حالت کلی حفظ کنید که اگر وابستگی دوحلقه‌ی تودرتو در مقدار اولیه یا نهایی آنها باشد از مرتبه $\theta(n^2)$ و اگر در تغییر اندیس باشد از مرتبه $\theta(n\log n)$ است یعنی:

```
for (i = 1 , i ≤ n  ;  i + +)
    for (j = 1  ;  j ≤ i  ;  j + +)
        [ ]   ⟶ θ(n²)
```

ولی

```
for (i = 1 ; i ≤ n ; i++)
    for (j = 1 ; j ≤ n ; j = j+i)
        [ ]  ⟶ θ(n log n)
```

اثبات به صورت زیر است. مثلاً برای $n = 8$ داریم :

n	i	j	تعداد بار اجرا
8	1	1,2,3,4,5,6,7,8	$8 = \left\lceil \frac{8}{1} \right\rceil$
	2	1,3,5,7	$4 = \left\lceil \frac{8}{2} \right\rceil$
	3	1,4,7	$3 = \left\lceil \frac{8}{3} \right\rceil$
	4	1,5	$2 = \left\lceil \frac{8}{4} \right\rceil$
	5	1,6	$2 = \left\lceil \frac{8}{5} \right\rceil$
	6	1,7	$2 = \left\lceil \frac{8}{6} \right\rceil$
	7	1,8	$2 = \left\lceil \frac{8}{7} \right\rceil$
	8	1	$1 = \left\lceil \frac{8}{8} \right\rceil$

پس تعداد دقیق اجرای دستور ; $x = x + 1$ برابر $T(n) = \sum_{i=1}^{n} \frac{n}{i}$ می‌باشد. توجه کنید که دو حلقه تودرتو همیشه از مرتبه $O(n^2)$ است ولی ممکن است در شرایطی خاص از این حداکثر $O(n^2)$ کمتر باشد.

$$T(n) = \sum_{i=1}^{n} \frac{n}{i} = n \sum_{i=1}^{n} \frac{1}{i} = n(1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}) = \theta(n \log n)$$

چرا که می‌دانیم:

$$1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \theta(\log n)$$

۴۴- (۲) فرض کنید $n = 2^{16} = 2^8 \times 2^8 = 256 \times 256 = 65536$

تعداد بار اجرا	شرط i <= n	i
1	درست	$2^1 = 2$
1	درست	$2^2 = 4$
1	درست	$2^4 = 16$
1	درست	$2^8 = 256$
1	درست	$2^{16} = 65536$
	غلط	2^{32}

پس به ازای $n = 2^{16}$ به تعداد ۵ بار و به ازای 2^8 به تعداد ۴ بار اجرا می‌شود پس تعداد اجرا برابر $\log(\log_2 n)+1$ می‌باشد که مرتبه آن به صورت $\theta(\log(\log_2 n))$ می‌شود.

اثبات دقیق: حلقه به ازای $i = 2^{(2^0)}, 2^{(2^1)}, 2^{(2^2)}, \ldots, 2^{(2^k)}$ اجرا می‌شود یعنی به تعداد $k+1$ بار به گونه‌ای که $n = (2)^{(2^k)}$ است، پس:

$$2^k \log_2 2 = \log_2 n \Rightarrow 2^k = \log_2 n \Rightarrow k \log_2 2 = \log_2 \log_2 n \Rightarrow k = \theta(\log \log n)$$

اگر تغییر i به صورت $i = i^3$ بود آنگاه حلقه به ازای $i = 2^{3^0}, 2^{3^1}, 2^{3^2}, \ldots, 2^{3^k}$ اجرا می‌شود به گونه‌ای که $n = (2)^{(3^k)}$ است و

$$3^k \log_2 2 = \log_2 n \Rightarrow 3^k = \log_2 n \Rightarrow k \log_3 3 = \log_3 \log_2 n \Rightarrow k = \theta(\log_3 \log_2 n)$$

۴۵- (۲) گزینه ۱ نادرست است چرا که در واقع عبارت زیر درست است :

$$n^2 \log n = \theta(n^2 \log n) = O(n^3)$$

گزینه ۳ نادرست است و عبارت زیر درست می‌باشد :

$$n^2 / \log n = \theta(n^2 / \log n) = O(n^2)$$

گزینه ۴ نادرست است و عبارت زیر به جای آن درست است :

$$n^3 2^n + 6n^2 3^n = O(n^2 3^n)$$

گزینه ۲ درست است.

۴۶- (۱) حلقه for داخلی از مرتبه $\theta(n)$ و while بیرونی از مرتبه $\theta(\log n)$ و در نتیجه چون دو حلقه داخل یکدیگر هستند مرتبه کلی $\theta(n \log n)$ می‌باشد.

۴۷- (۱) از طرفین گزینهٔ ۱، log می‌گیریم $\log n$ و $n^{\frac{1}{10}}$

$$\log \log n \Rightarrow \frac{1}{10} \log n \quad , \quad \log(n^{\frac{1}{10}})$$

بدیهی است که $\log n \in \Omega(\log\log n)$ پس گزینه ۱ صحیح است.

راه‌حل دوم :

$$\lim_{n\to\infty}\frac{\log n}{n^{\frac{1}{10}}} \xrightarrow{\text{هوپیتال}} \lim_{n\to\infty}\frac{\frac{1}{n}}{\frac{1}{10}\times n^{-\frac{9}{10}}} \Rightarrow \lim_{n\to\infty}\frac{1}{n\times n^{-\frac{9}{10}}} = \lim_{n\to\infty}\frac{1}{n^{\frac{1}{10}}} = \lim_{n\to\infty}\frac{1}{\sqrt[10]{n}} = 0$$

$$\Rightarrow n^{\frac{1}{10}} \in \Omega(\log n)$$

در حالت کلی رشد $\log n$ از n^{ε} $(\varepsilon > 0)$ کمتر است.

گزینه ۳ غلط است چرا که درست آن به صورت روبه‌روست : $8n^2 + 3n - 4 \in O(n^2)$

گزینه ۲ هم غلط بوده و درست آن به صورت زیر است :

$$\log(n!) < \log n^n = n\log n \Rightarrow \log(n!) \in O(n\log n)$$

گزینه ٤ هم باید به صورت زیر باشد :

$$1^n + 2^n + \ldots + n^n < n\times n^n \in O(n^{n+1})$$

٤٨- (٤) توجه کنید که داریم :

$$f(n) = O(g(n)) \Rightarrow g(n) = \Omega(f(n))$$

$$\text{if } f(n) = O(g(n)) \text{ and } f(n)\ \Omega\ (g(n)) \Rightarrow f(n) = \theta(g(n))$$

در گزینهٔ ۲ در واقع $g(n)$ یک عدد ثابت فرض شده است.

٤٩- (١) ظاهراً مسأله دو حلقه تودرتو است و مرتبه آن $O(n^2)$ است ولی در بدنه حلقه مقدار n تغییر می‌کند:

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

در حلقه داخلی j هر بار که j اضافه می‌شود یک واحد از n کم می‌شود، پس حلقه‌ی داخلی بار اول، $\frac{n}{2}$ بار اجرا می‌شود و n تقریباً $\frac{n}{2}$ می‌شود. در بار دوم اجرای i مقدار نهایی در واقع $\frac{n}{2}$ است و لذا حلقه‌ی درونی j این بار $\frac{n}{4}$ بار اجرا می‌شود و به همین ترتیب داریم:

$$\text{تعداد اجرای } x{+}{+} \simeq \frac{n}{2}+\frac{n}{4}+\frac{n}{8}+\cdots = n\left(\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16}+\cdots\right) = n\left(\frac{\frac{1}{2}}{1-\frac{1}{2}}\right) = O(n)$$

۵۰- (۳) می‌دانیم که :

$$\frac{t_2}{t_1}=\frac{O(n_2)}{O(n_1)}\times\frac{V_1}{V_2}$$

چون در هر دو حالت $t_2=t_1=1$ پس داریم :

$$\frac{O(n_2)}{O(n_1)}=\frac{v_2}{v_1}\Rightarrow\frac{n\times 2^n}{16\times 2^{16}}=131072$$

با ضرب مرتب $2^{10}=1024$ در عدد 2 متوجه می‌شویم که : $2^{17}=131072$. پس :

$$n\times 2^n=16\times 2^{16}\times 2^{17}=2^{37}$$

$$n\times 2^n=2^{37}\Rightarrow n=32$$

۵۱- (۱) درست شده گزینه ۱ عبارت روبه‌رو است : $6*2^n+n^2=\theta(2^n)$

۵۲- (۳) for درونی از مرتبه $\theta(n)$ و حلقه while بیرونی از مرتبه $\theta(\log_3^n)$ است و چون این دو حلقه درون یکدیگر و مستقل از همدیگرند، تعداد اجرای آنها در همدیگر ضرب می‌شوند : $\theta(n\log_3^n)$. البته در اصل گزینهٔ ۱ و ۳ معادل هستند ولی گزینهٔ ۳ دقیق‌تر است.

۵۳- (۳)

۵۴- (۲) ضریب ۲ از $\sum$ خارج شده و در محاسبه O اثری ندارد. در حالت کلی داریم :

$$\sum_{i=1}^{n} i\in O(n^2)\ ,\ \sum_{i=1}^{n} i^2\in O(n^3),\ \sum_{i=1}^{n} i^3\in O(n^4)$$

و الی آخر.

۵۵- (۳) حلقه خارجی i از دو حلقه دیگر کاملاً مستقل است پس در ابتدا آن را کنار می‌گذاریم. دو حلقه داخلی وابسته استاندارد هستند که به تعداد $\frac{m(m+1)}{2}$ بار اجرا می‌شوند. پس تعداد دقیق اجرا شدن دستور a := a + 3; برابر است با $\frac{nm(m+1)}{2}$ که اگر n=m باشد تعداد دقیق برابر $\frac{n^2(n+1)}{2}$ می‌شود که برابر $O(n^3)$ است.

۵۶- (۳) حلقه i از دو حلقه دیگر مستقل بوده و به تعداد log n بار اجرا می‌شود. لذا فقط به دو حلقه وابسته j و k توجه می‌کنیم :

```
for (j = 1 ; j ≤ n ; j = j * 2)
    for (k = 1 ; k ≤ j ; k ++)
```

فرض کنید $n = 16$ باشد، برای این مقدار مرتبه اجرایی دو حلقه وابسته فوق را به‌دست می‌آوریم :

n	j	تغییرات k	تعداد بار اجرا
16	1	از 1 تا 1	1
	2	از 1 تا 2	2
	4	از 1 تا 4	4
	8	از 1 تا 8	8
	16	از 1 تا 16	16

تعداد کل اجرا $= 1 + 2 + 4 + 8 + 16 = 15 + 16 = 2 \times 16 - 1$

تعداد کل اجرا در حالت کلی $= 2n - 1 = \theta(n)$

پس در حالت کلی مرتبه اجرایی 3 حلقه مذکور برابر $\theta(n \log n)$ می‌باشد.

۵۷- (۲) دو حلقه تودرتو، چه وابسته به هم باشند و چه مستقل از هم از مرتبه $\theta(n^2)$ هستند. این تست دو حلقه i و j وابسته به هم دارد. ولی به این نکته ظریف توجه **کنید که return باعث اتمام تابع می‌شود** و در همان بار اول اجرای حلقه‌ها چون جلوی if و else، هر دو **return** وجود دارد، پس حلقه‌ها فقط یک بار اجرا می‌شوند.

۵۸- (۱) درست گزینه ۱ به صورت $\frac{n^2}{\log n} = \theta\left(\frac{n^2}{\log n}\right)$ است و گزینه‌های ۲و۳و۴ درست هستند. توجه کنید که O معادل بزرگ‌تر یا مساوی، Ω معادل کوچک‌تر یا مساوی و θ معادل مساوی است.

۵۹- (٤) از آن‌جا که $O(n \log n) < O(n^3) < O(2^n)$ پس جمع این سه جمله در نهایت معادل $O(2^n)$ می‌شود.

۶۰- (۲) طبق قضیه زیر :

$$a_m n^m + a_{m-1} n^{m-1} + \ldots + a_0 = O(n^m)$$

۶۱- (۲) اثبات می‌شود که رشد تابع $(\log n)!$ از چند جمله‌ای بیش‌تر است یعنی برای $\alpha > 1$: $n^\alpha < (\log n)!$ از طرف دیگر $\log n! < n^\alpha$ است چرا که $\log n! = \theta(n \log n)$ و $n \log n < n^\alpha$ است. لذا در کل داریم : $\boxed{\log n! < n^\alpha < (\log n)!}$:

و در نتیجه : $g < h < f$

$$g(n) = O(f(n)) \,,\, g(n) = O(h(n)) \,,\, h(n) = O(f(n))$$

۶۲- (۳) با توجه به : $O(n!) > O(2^n) > O(n^a)$. جواب گزینه ۳ یعنی $O(n!)$ می‌شود.

۶۳- (٤) درستی گزینه ۳ بدیهی ست چرا که مفهوم Ω و O عکس یکدیگر است.

گزینه ٤ غلط است چرا که مثلاً اگر $f(n)=n^2$ باشد :

$$\Omega(n^2)+O(n^2)=n^3+n\neq\theta(n^2)$$

گزینه ۱ درست است چرا که مثلاً اگر $f(n)=n^2$ باشد :

$$f(n)+O(f(n))=n^2+O(n^2)=n^2+n=\theta(f(n))$$

گزینه ۲ درست است چرا که مثلاً اگر $f(n)=n^2$ و $g(n)=n^3$ باشند :

$$g(n)=\Omega(f(n)) \;:\; n^3=\Omega(n^2)\Rightarrow n^3=\Omega(O(n^2))=\Omega(n)$$

٦٤- (۱)

$$x\log x<x^2<x^{3.2}<x^{3.2}\log\sqrt{x}$$

٦٥- (۳)

مثال نقض گزینهٔ ۱:

$$f_1(n)=f_2(n)=\cdots=f_n(n)=n\in\theta(n)$$

$$\sum_{i=1}^{n}f_i(n)=\sum_{i=1}^{n}n=n^2\in\theta(n^2)$$

مثال نقض گزینهٔ ۲:

$$f(n)=n\quad,\quad g(n)=n^2\left|\sin\frac{n\pi}{2}\right|$$

$$f(n)\notin O(g(n))\qquad,\qquad g(n)\notin O(f(n))$$

مثال نقض گزینهٔ ٤:

$$f(n)=2n\in O(n)\quad,\quad 2^{2n}\notin O(2^n)$$

٦٦- (۱) گزینه‌های ۲ و ۳ و ٤ غلط هستند.

٦٧- (۳) $O(n+n^4+n^2)\Rightarrow O(n^4)$

٦٨- (۲)

$$\sum_{i=1}^{m}\sum_{j=1}^{A[i]}1=\sum_{i=1}^{m}A[i]=s$$

از آنجایی که ممکن است خود m از s بیشتر باشد جواب O(m+s) است.

٦٩- (٤) برای $\varepsilon<\frac{1}{2}$ و $k>1$ داریم:

$$(\log n)^k<n^\varepsilon<\sqrt{n}<\log n!$$

۷۰- (۱) فقط گزینهٔ (۲) صحیح است، مثلاً $f(n)=n^2$: $n^2+O(n^2)=n^2+n=\theta(n^2)$

گزینهٔ (۱) وقتی $f(n)<1$ درست نیست.

گزینهٔ (۳) درست نیست مثلاً $f(n)=2n$ ، $g(n)=n$ فرض کنید.

گزینهٔ (٤) درست نیست مثلاً $f(n)=2^n$ فرض کنید.

۷۱- (٤) می‌دانیم که $\log^b n = O(n^a)$ برای $a,b>0$

لذا گزینهٔ (٤) درست است $n(\log n)^5 = O(n^{1.2}) \Rightarrow (\log n)^5 = O(n^{0.2})$

$\sqrt{m} = n^{0.5} = \Omega(\log^5 n)$

گزینهٔ (۱) غلط است چرا که: $\frac{n}{\log n} \neq O(n^{1-x})$

گزینهٔ (۲) غلط است چرا که: $\sqrt{n} \neq O(\log^5 n)$

گزینهٔ (۳) غلط است چرا که: $3^n \neq O(n^{2^n})$

۷۲- (٤) گزینهٔ (٤) غلط است چرا که مثلاً برای $f(n)=n \;,\; g(n)=n^2\left|\sin\frac{n\pi}{2}\right|$ داریم $g(n)\neq O(f(n))$ ولی نمی‌توان نتیجه گرفت که $g(n)=\Omega(f(n))$ است. چرا که مرتباً به ازای مقادیر مختلف n تابع g بالا و پایین f قرار می‌گیرد.

مثال درستی گزینهٔ ۱: $f(n)=n^3 \Rightarrow n^3 = O(n^3) = \theta(n^3)$

مثال درستی گزینهٔ ۲: $f(n)=n^3 \;,\; g(n)=n^4 : g=\Omega(f) \Rightarrow n^4 = \Omega\left(O(n^3)\right) = \Omega(n^2)$

مثال درستی گزینهٔ ۳: $f(n)+g(n)=n^3+n^4 = O\left(\max\{n^3\;,\;n^4\}\right) = O(n^4)$

۷۳- (۲)

$n\log n < n^2 < n^3 < 2^n$

$O\left(10n^3+80n^2\right)=O\left(n^3\right) \quad , \quad O\left(n^2+100n\right)=O\left(n^2\right)$

۷٤- (۳) مثلاً برای $P=1$ داریم :

$$\sum_{i=1}^{n} i = 1+2+3+\ldots+m = \frac{n(n+1)}{2} = \theta\left(n^2\right)$$

یا برای $P=2$ داریم :

$$\sum_{i=1}^{n} i^2 = 1^2+2^2+3^2+\ldots+n^2 = \frac{n(n+1)(2n+1)}{6} = \theta\left(n^3\right)$$

۷۵- (۲) راه حل اول: اگر برنامه فقط حلقه (while (i>1 را می‌داشت جواب $O(\log n)$ می‌شود ولی چون دو حلقه‌ی تو در تو دارد پس از $O(\log n)$ بیشتر است. اگر به جای $j=i;$ فرض کنیم دستور $j=n;$ را می‌داشتیم پیچیدگی بیشتر می‌شد و چون در گزینه‌ها O را داریم این فرض ما غلط نیست در این صورت حلقه‌ی درونی $O(\log_3 n)$ و حلقه‌ی بیرونی $O(\log_2 n)$ بار اجرا می‌شود و داریم:

$O(\log_2 n \times \log_3 n) = O(\log^2 n)$

(چون مبنا در لگاریتم مهم نیست). از طرف دیگر می‌دانیم رشد a^n بزرگتر $\log^b n$ (به ازای $a,b>0$) می‌باشد لذا گزینهٔ ۲ دقیق‌تر از گزینه‌های ۳ و ٤ است.

راه دوم (دقیق‌تر): مقادیر i عبارتند از:

$$\frac{n}{2}, \frac{n}{4}=\frac{n}{n^2} \quad , \quad \frac{n}{8}=\frac{n}{2^3}, \cdots, \frac{n}{2^k}$$

که $k \approx \log_2 n$ است و به ازای مقادیر مختلف i حلقه while j به تعداد $\log_3 i$ بار اجرا می‌شود پس هزینه کل برنامه برابر است با:

$$\log_3 \frac{n}{2}+\log_3 \frac{n}{4}+\log_3 \frac{n}{8}+\cdots+\log_3 \frac{n}{2^k}=\log_3 n-\log_3 2+\log_3 n-\log_3 4+\cdots+\log_3 n-\log_3 2^k$$

$$=k\ \log_3 n-\left(\log_3 2^1\times 2^2\times 2^3\times\cdots\times 2^k\right)=k\log_3 n-\log_3 2^{\frac{k(k+1)}{2}}$$

$$=\log_2 n\times\log_3 n-\frac{k(k+1)}{2}\times\log_3 2\approx\log^2 n-\frac{\log_2^2 n}{2}=\theta(\log^2 n)$$

۷٦- (٤) زمان اجرای میانگین $\theta(n^2)$ است پس ممکن است زمان اجرا در بدترین حالت بیش از n^2 و در بهترین حالت کمتر از n^2 باشد. پس ممکن است ورودی وجود داشته باشد که زمان آن از n^3 بیشتر یا مساوی شود یا از n کمتر یا مساوی شود. پس هر سه جمله درست هستند. مثلاً در مرتب‌سازی حبابی که $\theta(n^2)$ است ممکن است آرایه از قبل مرتب باشد و زمان اجرا $\theta(n)$ شود.