# فصل اول

# کارایی. تحلیل و مرتبه الگوریتم‌ها - بازگشتی

## الگوریتم

الگوریتم دنباله‌ای است از دستورات که با اجرای آنها، هدف خاصی برآورده می‌شود. در هر الگوریتم، موارد زیر قابل بیان است:

۱) ورودی: الگوریتم می‌تواند ورودی داشته یا نداشته باشد.

۲) خروجی: الگوریتم باید حداقل یک خروجی داشته باشد.

۳) قطعیت: هر دستور باید واضح و خالی از هر نوع ابهامی باشد.

۴) محدودیت: الگوریتم باید پس از طی مراحل محدودی خاتمه یابد. مثلاً محاسبه و چاپ اعداد اول محدود نیست.

۵) کارایی: هر دستور باید انجام‌پذیر باشد، یعنی بتوان آن را به صورت دستی و با قلم و کاغذ اجرا کرد. مثلاً $\sqrt{-1}$ قابل اجرا نیست.

لازم به ذکر است که تفاوت برنامه (Program) با الگوریتم آن است که برنامه لزوماً محدود نیست (مورد ۴)؛ مثلاً سیستم عامل برنامه‌ای است که دائماً در یک سیکل انتظار است تا برنامه بعدی را اجرا کند (بجز در مواقعی که سیستم دچار خرابی شود).

برای ارزیابی کارایی یک الگوریتم، دو مسئله دارای اهمیت است:

۱) زمان اجرای الگوریتم ۲) حافظه مورد نیاز الگوریتم

از بین این دو عامل، زمان اجرا از اهمیت بیشتری برخوردار است. زمان اجرا به عوامل متعددی بستگی دارد از جمله:

۱) **سخت افزار**: مثلاً پردازنده Pen III از Pen II سریعتر است.

۲) **کامپایلر**: مثلاً برنامه‌های نوشته شده به زبان C معمولاً از مشابه پاسکال خود سریعتر هستند.

۳) **اندازه ورودی**: مثلاً در مرتب‌سازی یک آرایه، هر چه تعداد عناصر آرایه بیشتر باشد، زمان اجرا بیشتر است.

۴) **ترکیب یا ترتیب ورودی**: در بررسی الگوریتم‌ها معمولاً ۳ حالت وجود دارد: بهترین حالت (Best case)، حالت متوسط (Average case) و بدترین حالت (worst case). مثلاً برای مرتب‌سازی یک آرایه، اگر آرایه از قبل مرتب باشد معمولاً (نه همیشه) بهترین حالت است.

۵- **پیچیدگی الگوریتم**: ممکن است برای یک مسئله الگوریتم‌های مختلف با زمان‌های مختلف وجود داشته باشد. مثلاً برای مرتب‌سازی، الگوریتم‌های مختلفی وجود دارد که پیچیدگی‌شان، یکسان نیست.

اگر n اندازه ورودی باشد، زمان اجرای الگوریتم برحسب n را معمولاً با $T(n)$ نشان می‌دهیم. اگر الگوریتم بیش از یک ورودی داشته باشد که اندازه‌هایشان مثلاً n و m باشد، زمان را با $T(n , m)$ یا $T(m , n)$ نشان می‌دهیم.

**مثال.** اگر زمان اجرای خط شماره i برابر $t_i$ باشد، زمان اجرای تابع sum را محاسبه کنید:

```
float sum (float List [ ] , int n)
{
۱)  float  s = ۰; int  i;
۲)  for (i = ۰ ; i < n ; i + +)
۳)  s = s + List[i];
۴)  return s;
}
```

**پاسخ:** این تابع به زبان C نوشته شده است. خروجی این تابع حاصل جمع عناصر آرایه $List[۰..n-۱]$ است. تعریف تابع و تعریف متغیر، زمان صرف نمی کنند، البته اگر به متغیر مقدار اولیه دهیم، باید آن را حساب کنیم. برای سایر خطوط، کافیست تعداد اجرای هر خط را در زمان آن خط ضرب کنید و مقادیر بدست آمده را جمع کنید:

| خط | تعداد | زمان | زمان کل |
|---|---|---|---|
| ۱ | ۱ | $t_۱$ | $۱ \times t_۱$ |
| ۲ | n+۱ | $t_۲$ | $(n+۱)\, t_۲$ |
| ۳ | n | $t_۳$ | $n\, t_۳$ |
| ۴ | ۱ | $t_۴$ | $t_۴$ |

بنابراین:

$$T(n) = t_۱ + (n+۱)\, t_۲ + n\, t_۳ + t_۴$$

اگر فرض کنیم زمان هر خط i برابر واحد باشد $(t_i = ۱)$ آنگاه:

$$T(n) = ۱ + n + ۱ + n + ۱ = ۲n + ۳$$

بنابراین زمان اجرای تابع sum برابر $T(n) = ۲n + ۳$ است که این تعداد اجرای همه خطوط نیز هست. پس از این به بعد، برای محاسبه زمان اجرای یک الگوریتم می‌توان تعداد اجرای همه خطوط را محاسبه کرد (گام شماری).

**مثال:** در تکه برنامه‌های زیر، تعداد اجرای همه خطوط محاسبه شده است:

۱) `for (i = ۱ ; i <= n ; i++)` // $n+۱$
`x+ = ۱;` // $n$
$answer = n+۱+n = ۲n+۱$

۲) `for (i = a; i <= b ; i++)` // $b-a+۲$
`x+ = ۱;` // $b-a+۱$
$answer = b-a+۲+b-a+۱ = ۲(b-a)+۳$

**نکته:**

$a \le i \le b \Rightarrow$ تعداد اجرای جمله داخل حلقه $= b-a+۱$
تعداد اجرای خود حلقه $= b-a+۲$

$a \le i < b \Rightarrow$ تعداد اجرای جمله داخل حلقه $= b-a$
تعداد اجرای خود حلقه $= b-a+۱$

۳) `for (i = a ; i < b; i++)` // $b-a+۱$
`x+ = ۱ ;` // $b-a$
$answer = b-a+۱+b-a = ۲(b-a)+۱$

۴) `for (i = ۱ ; i <= n ; i++)` // $n+۱$
`for (j = ۱; j <= n ; j++)` // $n(n+۱)$
`x+ = ۱;` // $n^۲$
$answer = n+۱+n^۲+n+n^۲ = ۲n^۲+۲n+۱$

۵) `for(i = ۱; i <= n; i++)` // $n+۱$
`for(j = ۱; j <= n; j++)` // $n(n+۱)$
`for(k = ۱; k <= n; k++)` // $n^۲(n+۱)$
`x+ = ۱;` // $n^۳$
$answer = n+۱+n^۲+n+n^۳+n^۲+n^۳$
$= ۲n^۳+۲n^۲+۲n+۱$

**توجه:** در بسیاری از موارد، محاسبه تعداد اجرای همه خطوط به صورت دقیق مورد نظر نیست بلکه فقط درجه یا مرتبه آن خواسته می‌شود. مثلاً در تکه برنامه شماره ۵ فوق، درجه یا مرتبه جواب، $n^۳$ است که گفته می‌شود. زمان اجرای این تکه برنامه از مرتبه $O(n^۳)$ یا $\theta(n^۳)$[۱] است. بنابراین زمان تکه برنامه

۱ - مفهوم $O$ و $\theta$ و سایر نمادها، در ادامه همین فصل بررسی شده است.

۴، $\theta(n^۲)$ و زمان تگه برنامه ۱، $\theta(n)$ است. برای محاسبه مرتبه، لازم نیست تعداد اجرای همه خطوط محاسبه شوند، بلکه تعداد اجرای جمله اصلی (جمله داخل حلقه)، مرتبه را مشخص می‌کند.

**مثال:** در تگه برنامه زیر، ابتدا تعداد اجرای جمله اصلی را بیابید، سپس مرتبه آن را بیان کنید.

```
for (i = ۱; i <= n ; i++)
    for (j = ۱ ; j <= i ; j++)
        x+ = ۱;
```

**پاسخ:** به ازای i=۱، متغیر j از ۱ تا ۱ تغییر می‌کند و جمله x+ = ۱ یک بار اجرا می‌شود.

به ازای i=۲، متغیر j از ۱ تا ۲ تغییر می‌کند و جمله x+ = ۱ دو بار اجرا می‌شود.

به ازای i=n، متغیر j از ۱ تا n تغییر می‌کند و جمله x+ = ۱ n بار اجرا می‌شود.

بنابراین تعداد اجرای جمله x+ = ۱ برابر است با:

$$۱+۲+\ldots+n=\frac{n(n+۱)}{۲}=\theta(n^۲)$$

**توجه:** برای محاسبه تعداد اجرای جمله اصلی در حلقه‌های for که اندیس‌هایشان ۱ واحد افزایش می‌یابد می‌توان از $\sum$ استفاده کرد. به این صورت که به ازای هر حلقه $\sum$ قرار دهید و جمله داخل حلقه را ۱ بگذارید:

$$\sum_{i=۱}^{n}\sum_{j=۱}^{i}۱=\sum_{i=۱}^{n}i=\frac{n(n+۱)}{۲}$$

**توجه:** برخی خواص $\sum$ عبارتند از:

۱) $\sum_{i=m}^{n} f(i)=f(m)+f(m+۱)+\ldots+f(n)$

۲) $\sum_{i=m}^{n} af(i)=a\sum_{i=m}^{n} f(i)$ ، a به i وابسته نیست

۳) $\sum_{i=m}^{n} a=(n-m+۱)a$ ، a به i وابسته نیست

۴) $\sum_{i=m}^{n} f(i)=\sum_{i=m+k}^{n+k} f(i-k)$

۵) $\sum_{i=۱}^{n} i=۱+۲+\ldots+n=\frac{n(n+۱)}{۲}=\theta(n^۲)$

۶) $\sum_{i=۱}^{n} i^۲=۱^۲+۲^۲+\ldots+n^۲=\frac{n(n+۱)(۲n+۱)}{۶}=\theta(n^۳)$

۷) $\sum_{i=۱}^{n} i^۳=۱^۳+۲^۳+\ldots+n^۳=\left(\frac{n(n+۱)}{۲}\right)^۲=\theta(n^۴)$

۸) $\sum_{i=1}^{n} i^k = 1^k + 2^k + \ldots + n^k \sim \frac{1}{k+1} n^{k+1} = \theta(n^{k+1})$ و منظور از $\sim$ هم‌ارزی است

۹) $\sum_{k=1}^{\infty} a_k = \lim_{n\to\infty} \sum_{k=1}^{n} a_k$

۱۰) $\sum_{k=1}^{n} \theta(f(k)) = \theta\left(\sum_{k=1}^{n} f(k)\right)$

**مثال:** تعداد اجرای جمله اصلی را محاسبه کنید و مرتبه را بیان کنید:

```
for i ← 1 to n
    for j ← 1 to i
        for k ← 1 to j
            x ← x + 1
```

**پاسخ:**

$$\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{j} 1 = \sum_{i=1}^{n}\sum_{j=1}^{i} j = \sum_{i=1}^{n} \frac{i(i+1)}{2} = \frac{1}{2}\left(\sum_{i=1}^{n} i^2 + \sum_{i=1}^{n} i\right)$$

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

**نکته:** *تعداد اجرای جمله* $x \leftarrow x+1$ *در* $k$ *حلقه وابسته زیر برابر* $\binom{k+n-1}{k}$ *است که از مرتبه* $\theta(n^k)$ *است.*

```
for i1 ← 1 to n
for i2 ← 1 to i1
  ⋮
for ik ← 1 to ik-1
    x ← x + 1;
```

**مثال:** مرتبه زمانی تکه برنامه‌های زیر را بیابید:

۱)
```
for (i = 1 ; i <= n ; i++)
  {
    for (j = 1 ; j <= n ; j++)
      x = x + 1;
    n = n - 1;
  }
```

**پاسخ:** (فرض کنید n زوج است) به ازای $i = 1$، جمله $x = x + 1$، $n$ بار اجرا می‌شود و از $n$ یک واحد کم می‌شود.

به ازای $i=2$، جمله $x = x + 1$، $n-1$ بار اجرا می‌شود و از $n$ مجدداً یک واحد کم می‌شود.

به ازای i=۲، جمله x = x + ۱، n-۲ بار اجرا می‌شود و از n یک واحد دیگر کم می‌شود.

⋮

به ازای $i = \frac{n}{۲}$، جمله x = x + ۱، $n = \frac{n}{۲} + ۱$ بار اجرا می‌شود و از n یک واحد دیگر کم می‌شود.

به ازای $i = \frac{n}{۲} + ۱$ شرط i <= n دیگر درست نیست!

$$(\text{تعداد اجرای } x = x + ۱) = n + n - ۱ + n - ۲ + \ldots + \left(n - \frac{n}{۲} + ۱\right) = \frac{۳n^۲ + ۲n}{۸} = \theta(n^۲)$$

```
۲)  for (i = ۱ ; i <= n ; i + +)
      for (j = ۱ ; j <= n ; j + +)
       {
         x = x + ۱;
         n = n − ۱;
       }
```

**پاسخ:** به ازای i=۱، جمله x = x + ۱، $\frac{n}{۲}$ بار اجرا می‌شود.

به ازای i=۲، جمله x=x+۱، $\frac{n}{۴}$ بار اجرا می‌شود.

به همین ترتیب:

$$(\text{تعداد اجرای جمله } x = x + ۱) = \frac{n}{۲} + \frac{n}{۴} + \frac{n}{۸} + .. = n\left(\frac{۱}{۲} + \frac{۱}{۴} + \frac{۱}{۸} + ..\right) = n \times \frac{\frac{۱}{۲}}{۱ - \frac{۱}{۲}} = \theta(n)$$

```
۳)  for (i = ۱ ; i <= n ; i + +)
      for (j = ۱ ; j <= n ; j + = i)
        x = x + ۱;
```

**پاسخ:**

$$(\text{تعداد اجرای جمله } x = x + ۱) = n + \frac{n}{۲} + \frac{n}{۳} + \frac{n}{۴} + \ldots + \frac{n}{n} = n\left(۱ + \frac{۱}{۲} + \frac{۱}{۳} + \ldots + \frac{۱}{n}\right) = \theta(n \lg n)$$

**نکته:** ثابت می‌شود $۱ + \frac{۱}{۲} + \ldots + \frac{۱}{n} = \text{Ln}n + O(۱)$ *(سری هارمونیک). و همچنین در این بخش بررسی خواهیم کرد که لگاریتم‌ها با هر پایه‌ای، مرتبه‌شان یکسان است:* $\ln n = \theta(\lg n)$

**توجه:** لگاریتم در پایه ۲ را معمولاً به صورت lg می‌نویسیم: $\log_۲^n = \lg n$

**نکته:** *مجموع n جمله اول یک تصاعد هندسی که جمله اول آن a و قدر نسبت آن q است، برابر است با:*

$$a + aq + aq^۲ + \ldots + aq^{n-۱} = a \times \frac{q^n - ۱}{q - ۱}$$

بنابراین اگر $n \to \infty$ و $|q| < ۱$ آنگاه می‌توان نوشت: $a + aq + aq^۲ + \ldots = \frac{a}{۱-q}$

**نتیجه:** $\sum_{k=۰}^{\infty} kx^k = \frac{x}{(۱-x)^۲}$ (مشتق گرفته در x ضرب کنید) $\Rightarrow |x| < ۱ \Rightarrow \sum_{k=۰}^{\infty} x^k = \frac{۱}{۱-x}$

**نکته:** *مجموع n جمله اول تصاعد حسابی با جمله اول a و جمله nام b و قدر نسبت d برابر است با:*

$$a + (a+d) + (a+۲d) + \ldots + \underbrace{(a+(n-۱)d)}_{b} = \frac{n}{۲}(a+b)$$

**توجه:** در بسیاری از الگوریتم‌ها، تعداد اجرای جمله اصلی، متفاوت است و به شرایط بستگی دارد. در این صورت بهترین حالت، حالت متوسط و بدترین حالت برای الگوریتم مطرح می‌شود. مثلاً برای جستجوی خطی عنصر x در آرایه غیرمرتب A[۱..n]، اگر x در A[۱] باشد فقط یک مقایسه انجام می‌شود. اگر x در A[n] باشد یا اصلاً در A نباشد n مقایسه انجام می‌شود. پس می‌توان گفت متوسط $\frac{n+۱}{۲}$ مقایسه انجام می‌شود:

$B(n) = ۱$ (Best Case)

$W(n) = n$ (Worst Case)

$A(n) = \frac{n+۱}{۲} = \theta(n)$ (Average Case)

البته، باید تحلیل دقیق تری برای یافتن A(n) انجام شود. برای این منظور فرض کنید، عنصر x حتماً در آرایه A وجود دارد و احتمال حضورش در همه خانه‌های A، یکسان است. احتمال وجود x در هر خانه مثل i $(P_i)$ برابر $\frac{۱}{n}$ است پس اگر $C_i$ تعداد مقایسه‌ها هنگامی که x در خانه i هست باشد آنگاه:

$$A(n) = \sum_{i=۱}^{n} C_i P_i = ۱ \times \frac{۱}{n} + ۲ \times \frac{۱}{n} + \ldots + n \times \frac{۱}{n} = \frac{۱}{n}\left(\frac{n(n+۱)}{۲}\right)$$

$$\Rightarrow A(n) = \frac{n+۱}{۲}$$

البته اگر فرض کنیم با احتمال P عنصر x در آرایه A هست و با احتمال P-۱ در آرایه نیست، آنگاه احتمال وجود x در هر خانه مثل i برابر $P_i = \frac{P}{n}$ است و:

$$A(n) = ۱ \times \frac{P}{n} + ۲ \times \frac{P}{n} + \ldots + n \times \frac{P}{n} + n \times (۱-P) = \frac{P}{n}\frac{n(n+۱)}{۲} + n(۱-P)$$

مثلاً اگر $P = \frac{۱}{۲}$ باشد آنگاه:

$$A(n) = \frac{۳n+۱}{۴}$$

**مثال:** در الگوریتم زیر (مرتب‌سازی درجی) فرض کنید هر بار اجرای خط iام زمان $C_i$ صرف کند و $C_i$ ثابت است. همچنین فرض کنید به ازای هر n و ... و ۳ و ۲=j تعداد اجرای حلقه while در خط ۵ برابر $t_j$ باشد. زمان اجرا را بیابید:

| INSERTION-SORT (A) | Cost | Times |
|---|---|---|
| ۱) for j ← ۲ to n do | $C_1$ | n |
| { | | |
| ۲) key ← A[j] | $C_2$ | n-۱ |
| ۳) // Insert A[j] into the sorted Sequence A[۱..j-۱] | ۰ | n-۱ |
| ۴) i ← j-۱ | $C_4$ | n-۱ |
| ۵) While (i > ۰) and (A[i] > key) do | $C_5$ | $\sum_{j=2}^{n} t_j$ |
| { | | |
| ۶) A[i+۱] ← A[i] | $C_6$ | $\sum_{j=2}^{n} (t_j - 1)$ |
| ۷) i ← i-۱ | $C_7$ | $\sum_{j=2}^{n} (t_j - 1)$ |
| } | | |
| ۸) A[i+۱] ← key | $C_8$ | n-۱ |
| } | | |

برای محاسبه زمان اجرای T(n) کافیست $C_i$ ها را در تعداد بار اجرا ضرب کنیم و مقادیر بدست آمده را جمع کنیم:

$$T(n) = C_1 n + C_2(n-1) + C_4(n-1) + C_5 \sum_{j=2}^{n} t_j + C_6 \sum_{j=2}^{n} (t_j - 1) + C_7 \sum_{j=2}^{n} (t_j - 1) + C_8(n-1)$$

بهترین حالت در الگوریتم فوق وقتی است که آرایه از قبل مرتب باشد که در این صورت در حلقه while شرط A[i] > key همیشه غلط است و بنابراین $t_j = 1$ ، $j = 2,3,\ldots,n$ می‌باشد پس:

$$B(n) = C_1 n + C_2(n-1) + C_4(n-1) + C_5(n-1) + C_8(n-1) =$$

$$= (C_1 + C_2 + C_4 + C_5 + C_8)n - (C_2 + C_4 + C_5 + C_8) = an + b = \theta(n)$$

اگر آرایه معکوس مرتب باشد، بدترین حالت است. در این صورت باید A[j] با همه عناصر قبل خودش یعنی با A[۱..j-۱] مقایسه شود پس $t_j = j$ ، $j = 2,3,\ldots,n$ بنابراین:

$$W(n) = C_1 n + C_2(n-1) + C_4(n-1) + C_5\left(\frac{n(n+1)}{2} - 1\right) + C_6\left(\frac{n(n-1)}{2}\right)$$

$$+C_۷\left(\frac{n(n-۱)}{۲}\right)+C_۸(n-۱)$$

$$=\left(\frac{C_۵}{۲}+\frac{C_۶}{۲}+\frac{C_۷}{۲}\right)n^۲+\left(C_۱+C_۲+C_۴+\frac{C_۵}{۲}-\frac{C_۶}{۲}-\frac{C_۷}{۲}+C_۸\right)n$$

$$-(C_۲+C_۴+C_۵+C_۸)=an^۲+bn+c=\theta(n^۲)$$

در حالت متوسط می‌توان فرض کرد که $A[j]$ از نصف عناصر $A[۱..j-۱]$ کوچکتر است بنابراین $t_j=\frac{j}{۲}$ که اگر محاسبه کنید $A(n)=an^۲+bn+c=\theta(n^۲)$. یعنی حالت متوسط تابعی درجه ۲ از a می‌باشد (همانند بدترین حالت).

**توجه:** برای بررسی درستی یک الگوریتم می‌توان از Loop invariant (Li) استفاده کرد. باید ۳ مورد را بررسی کنیم:

initialization: نشان دهیم قبل از اولین اجرای حلقه، نتیجه صحیح است. مثلاً برای مرتب‌سازی درجی، قبل از اولین اجرای حلقه یعنی $j=۲$، آرایه فقط $A[۱]$ است که مرتب است.

Maintenance: اگر قبل از اجرای فعلی حلقه نتیجه درست است، قبل از اجرای بعدی حلقه نیز صحیح بماند. در مرتب‌سازی درجی، وقتی $j=k$ باشد $A[۱..k]$ مرتب است و اگر $j=k+۱$، $A[۱..k+۱]$ مرتب است.

termination: وقتی حلقه پایان یافت بتوان نشان داد نتیجه دلخواه بدست آمده است. در مرتب‌سازی درجی، حلقه for وقتی تمام می‌شود که $j=n+۱$ (مثل زبان C، فرض کرده‌ایم حلقه که تمام شد شمارنده $n+۱$ است)

این مراحل شبیه استقرای ریاضی هستند بجز آخرین مرحله، چون در استقرای معمولی، مقادیر نامحدود هستند و termination ندارد.

## رشد توابع

گوییم رشد تابع $f(n)$ از تابع $g(n)$ بیشتر است اگر $n\to\infty$ آنگاه $f(n)$ زودتر به $\infty$ میل کند.

**مثال:** توابع n و lgn و $n^۲$ و $۲^n$ و $n^۳$ و nlgn را به ترتیب رشد، مرتب کنید.

**پاسخ:** به ازای مقادیر ۳۲ و ۱۶ و ۸ و ۴ و ۲ و ۱ =n، مقادیر توابع فوق را محاسبه می‌کنیم:

| | ۱ | ۲ | ۴ | ۸ | ۱۶ | ۳۲ |
|---|---|---|---|---|---|---|
| lgn | ۰ | ۱ | ۲ | ۳ | ۴ | ۵ |
| n | ۱ | ۲ | ۴ | ۸ | ۱۶ | ۳۲ |
| nlgn | ۰ | ۲ | ۸ | ۲۴ | ۶۴ | ۱۶۰ |
| $n^۲$ | ۱ | ۴ | ۱۶ | ۶۴ | ۲۵۶ | ۱۰۲۴ |
| $n^۳$ | ۱ | ۸ | ۶۴ | ۵۱۲ | ۴۰۹۶ | ۳۲۷۶۸ |
| $۲^n$ | ۲ | ۴ | ۱۶ | ۲۵۶ | ۶۵۵۳۶ | ۴۲۹۴۹۶۷۲۹۵ |

همانطور که ملاحظه شد، رشد این توابع به شکل زیر است:

$$\lg n < n < n \lg n < n^2 < n^3 < 2^n$$

**نکته:** *ترتیب زیر قابل اثبات است:*

$$1 < \lg n < n < n \lg n < n^2 < n^3 < n^4 < \ldots < 2^n < 3^n < 4^n < \ldots < n! < n^n$$

۱ یعنی بدون رشد، یعنی به n وابسته نیست. مثلاً حلقه‌ای که همیشه ۱۰۰۰ بار اجرا می‌شود رشدش ۱ است:

```
for (i = 1 to 1000)
x = x + 1 ;  //      1000 بار = θ(1)
```

## چند خاصیت ریاضی

صحیح $n \geq 0 , a, b > 0$

۱) $\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor = n$

۲) $\left\lceil \frac{\left\lceil \frac{n}{a} \right\rceil}{b} \right\rceil = \left\lceil \frac{n}{ab} \right\rceil$

۳) $\left\lfloor \frac{\left\lfloor \frac{n}{a} \right\rfloor}{b} \right\rfloor = \left\lfloor \frac{n}{ab} \right\rfloor$

۴) $\left\lceil \frac{a}{b} \right\rceil \leq \frac{(a + (b-1))}{b}$

۵) $\left\lfloor \frac{a}{b} \right\rfloor \geq \frac{(a - (b-1))}{b}$

۶) $a \bmod n = a - \left\lfloor \frac{a}{n} \right\rfloor n$

۷) $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$

۸) $n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \theta\left(\frac{1}{n}\right)\right)$ تقریب استرلینگ

۹) $n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\alpha_n} \quad , \quad \frac{1}{12n+1} < \alpha_n < \frac{1}{12n} \quad , \quad n \geq 1$

۱۰) $a^{\log_a^b} = b$

۱۱) $a^{\log_c^b} = b^{\log_c^a}$

۱۲) $\log_b^a = \frac{\log_c^a}{\log_c^b}$

**نکته:** *برای مقایسه رشد توابع می‌توان از حد استفاده کرد:*

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 \Leftrightarrow & \text{رشد } f \text{ از } g \text{ کمتر است.} \\ \infty \Leftrightarrow & \text{رشد } f \text{ از } g \text{ بیشتر است.} \\ k \neq 0 \Leftrightarrow & f \text{ و } g \text{ رشد یکسان دارند.} \end{cases}$$

**مثال:** رشد $f(n) = 2^n$ را با $g(n) = n^4$ مقایسه کنید.

**پاسخ:**

$$\lim_{n\to\infty} \frac{2^n}{n^4} = \infty \quad \text{(بررسی کنید)}$$

پس رشد $2^n$ از $n^4$ بیشتر است.

**مثال:** رشد $f(n) = \log_3^n$ را با $f(n) = \log_4^n$ مقایسه کنید

**پاسخ:**

$$\lim_{n\to\infty} \frac{\log_3^n}{\log_4^n} = \lim_{n\to\infty} \frac{\frac{\ln n}{\ln 3}}{\frac{\ln n}{\ln 4}} = \log_3^4 \neq 0 \quad \Rightarrow \quad \text{هم رشد هستند}$$

نتیجه: $\log_a^n$ با $\log_b^n$ هم رشد است ($a, b > 1$ ثابت هستند)

**تعریف:** تابع $\lg^* n$ تابعی است با رشد بسیار ناچیز و به این صورت تعریف می‌شود:

$$\lg^* n = \min\left\{ i : \underbrace{\lg \lg \ldots \lg}_{i \text{ بار}} n \leq 1 \right\}$$

یعنی $\lg^* n$ برابر کوچکترین $i$ است که اگر $i$ بار از $n$ لگاریتم بگیریم، حاصل کوچکتر یا مساوی یک شود.

به عنوان مثال:

$\lg^* 2 = 1$ $(\lg 2 = 1)$

$\lg^* 4 = 2$ $(\lg \lg 4 = 1)$

$\lg^* 16 = 3$ $(\lg \lg \lg 16 = 1)$

$\lg^* 65536 = 4$

$\lg^* 2^{65536} = 5$

می‌توان نشان داد $\lg^*(\lg n) = (\lg^* n) - ۱$.

**مثال:** $\lg^* ۱۰۰۰$ را محاسبه کنید.

**پاسخ:** $\lg^* ۱۰۰۰ = ۴$ زیرا $\lg\lg\lg\lg ۱۰۰۰ < ۱$ یعنی چهار بار که از ۱۰۰۰ لگاریتم بگیریم، حاصل کوچکتر از یک می‌شود.

**نکته:** *ثابت می‌شود که* $\lg n!$ *با* $n\lg n$ *رشد یکسان دارد.*

تمرین: تساوی‌های زیر را اثبات کنید:

$$(\lg n)^{\lg n} = n^{\lg\lg n} \quad , \quad ۴^{\lg n} = n^۲ \quad , \quad ۲^{\lg n} = n \quad , \quad n^{\frac{۱}{\lg n}} = ۲$$

$$۲^{\sqrt{۲\lg n}} = n^{\sqrt{\frac{۲}{\lg n}}} \quad , \quad (\sqrt{۲})^{\lg n} = \sqrt{n}$$

**مثال:** ترتیب رشد برخی توابع، به شکل زیر است، همگی قابل اثبات هستند:

$$n^{\frac{۱}{\lg n}} < \lg(\lg^* n) < \lg^* n \sim \lg^*(\lg n) < ۲^{\lg^* n} < \ln\ln n$$

$$< \sqrt{\lg n} < \ln n < \lg^۲ n < ۲^{\sqrt{۲\lg n}} < \sqrt{۲}^{\lg n} = \sqrt{n} < ۲^{\lg n} = n$$

$$< n\lg n \sim \lg n! < ۴^{\lg n} = n^۲ < n^۳ < (\lg n)!$$

$$< (\lg n)^{\lg n} = n^{\lg\lg n} < \left(\frac{۳}{۲}\right)^n < ۲^n < n\times ۲^n < e^n < n! < (n+۱)! < ۲^{۲^n} < ۳^{۲^{n+۱}}$$

منظور از علامت $\sim$ در این مثال، یعنی هم رشد هستند.

**مثال:** نشان دهید $f(n) = \begin{cases} ۲^{۲^{n+۲}} & , \text{even } n \\ ۰ & , \text{odd } n \end{cases}$، $O$ و $\Omega$ هیچ یک از توابع مثال قبل نیست. (تعریف $O$ و $\Omega$ در ادامه آمده است)

## نمادهای مجانبی (Asymptotic notation)

وقتی اندازه ورودی ($n$) بزرگ باشد، بیان زمان دقیق اجرا برحسب $n$ ضروری نیست زیرا جمله با بزرگترین درجه در زمان موثر است. مثلاً اگر زمان اجرای الگوریتمی $۲n^۳ + n^۲ + n$ باشد به ازای $n$های بزرگ فقط $n^۳$ مهم است. می‌توان با نمادهای مجانبی این تفسیر را بهتر نشان داد.

در بیان نمادها فرض می‌شود که دامنه اعداد طبیعی $N = \{۰,۱,۲,...\}$ است البته در برخی مواقع شاید بهتر باشد که دامنه را اعداد حقیقی فرض کنیم.

### نماد θ

برای تابع g(n)، θ(g(n)) یک مجموعه توابع است که به شکل زیر تعریف می‌شود:

$$\theta(g(n))=\{f(n):\exists C_1,C_2,n_0>0\ ,\ \forall\ n\geq n_0:0\leq C_1g(n)\leq f(n)\leq C_2g(n)\}$$

یعنی تابع f(n) عضو $\theta(g(n))$ است ($f(n)\in\theta(g(n))$) یا می‌نویسیم $f(n)=\theta(g(n))$ هرگاه اعداد ثابت مثبت $C_1$ و $C_2$ وجود داشته باشند که برای nهای به اندازه کافی بزرگ f(n) بین $C_1g(n)$ و $C_2g(n)$ ساندویچ شود.

به عنوان مثال $\frac{1}{2}n^2-3n=\theta(n^2)$ زیرا:

$$C_1n^2\leq\frac{1}{2}n^2-3n\leq C_2n^2\ \Rightarrow\ C_1\leq\frac{1}{2}-\frac{3}{n}\leq C_2$$

اگر $C_1=\frac{1}{14}$ و $C_2=\frac{1}{2}$ انتخاب شود نامساوی فوق به ازای همه nهای بزرگتر یا مساوی $n_0=7$ برقرار است. دقت کنید که برای $C_1$ و $C_2$ و $n_0$ انتخاب‌های دیگری نیز وجود دارد که مهم نیست، مهم این است که $C_1$ و $C_2$ و $n_0$ وجود دارند.

و یا می‌توان نشان داد که $6n^3\neq\theta(n^2)$ زیرا نامساوی $6n^3\leq C_2n^2$ به ازای nهای بزرگ، صحیح نیست زیرا $C_2$ ثابت است.

بنابراین یک چند جمله‌ای درجه P مثل $an^P+bn^{P-1}+\ldots$ که P ثابت است، از مرتبه $\theta(n^P)$ است. و به طور کلی θ دقیقاً مرتبه رشد تابع را مشخص می‌کند:

$$6n^5-n+1\in\theta(n^5)\ ,\ 2^n+n^{20}+n\in\theta(2^n)\ ,\ \lg n!\in\theta(n\lg n)\ ,\ \ldots$$

### نماد O

نماد θ یک تابع را از بالا و پایین به صورت مجانبی (حدی) محدود می‌کرد.

نماد O یک کران بالای حدی مشخص می‌کند:

$$O(g(n))=\{f(n):\exists C,n_0>0\ ,\ \forall n\geq n_0:\ \ 0\leq f(n)\leq cg(n)\}$$

بنابراین O یک کران بالا برای تابع مشخص می‌کند. ما می‌نویسیم $f(n)=O(g(n))$ یا $f(n)\in O(g(n))$ اگر f(n) عضوی از $O(g(n))$ باشد.

به راحتی می‌توان نشان داد که $3n^2+n+1\in O(n^2)$ و $3n+1\in O(n^2)$ و ... . به طور کلی $O(n^2)$ شامل همه توابعی است که رشدشان کمتر یا مساوی $n^2$ است:

$$O(n^2)=\{\ldots,n^2,n,n\lg n,2n^2+n\ ,\ \lg n,20,\ldots\}$$

### نماد Ω

نماد Ω یک کران پایین حدی برای تابع مشخص می‌کند:

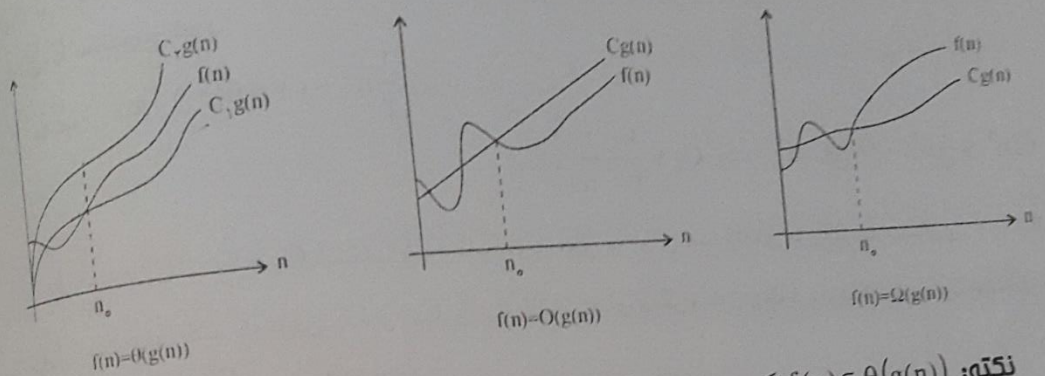
$$\Omega(g(n))=\{f(n):\exists c,n_0>0\ ,\ \forall n\geq n_0:0\leq cg(n)\leq f(n)\}$$

می توان نشان داد که $2n^2+3n+1 \in \Omega(n^2)$ و $2n^3+3n^2+n \in \Omega(n^2)$ و ... .

به طور کلی $\Omega(n^2)$ شامل همه توابعی است که رشدشان بیشتر یا مساوی $n^2$ است:

$$\Omega(n^2)=\{\dots, n^2+1,\ 2n^2-1,\ 2^n,\ n^2\lg n,\ \dots\}$$

شکل زیر نمادهای $O$ و $\theta$ و $\Omega$ را نشان می‌دهد:




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

**نماد o (little – oh)**

نماد اُکوچک، کران بالایی مشخص می‌کند که اکید است مثلاً $2n+1\in O(n)$ ولی $2n+1\notin o(n)$ و $2n+1\in o(n^2)$ و ... .

$$o(g(n))=\{f(n):\forall c>0\ ,\ \exists n_0>0\ ,\ \forall n\geq n_0 : 0\leq f(n)<cg(n)\}$$

یعنی نامساوی $f(n)<cg(n)$ باید به ازای همه اعداد مثبت $c$ برقرار باشد بنابراین $2n+1\notin o(n)$ زیرا $2n+1<c.n$ به ازای مثلاً $c=1$ برای $n$های بزرگ درست نیست. بنابراین مثلاً $o(n^2)$ شامل همه توابعی است که رشدشان از $n^2$ کمتر است:

$$o(n^2)=\{\dots, n, 2n+1,\ \lg n,\ n\lg n,\ \dots\}$$

**نماد ω**

کران پایین اکید است. مثلاً $2n^2+1\in\Omega(n^2)$ ولی $2n^2+1\notin\omega(n^2)$ در عوض $2n^2+1\in\omega(n)$ و ... .

$$\omega(g(n))=\{f(n):\forall c>0\ ,\ \exists n_0>0\ ,\ \forall n\geq n_0 : 0\leq cg(n)<f(n)\}$$

پس مثلاً $\omega(n^2)$ شامل همه توابعی است که رشدشان از $n^2$ بیشتر است:

$$\omega(n^2)=\{\dots, n^3,\ 2n^4+n^2+1,\ n^2\lg n,\ \dots\}$$

## مقایسه نمادها

### تعدی (transitivity)

$$f(n)=\theta(g(n)) \text{ and } g(n)=\theta(h(n)) \text{ imply } f(n)=\theta(h(n))$$

بجای $\theta$ می‌توان سایر نمادها را نیز قرار داد.

### بازتابی (reflexivity)

$\theta$ و $O$ و $\Omega$ بازتاب هستند:

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

### تقارنی (symmetry)

فقط $\theta$ متقارن است:

$$f(n)=\theta(g(n)) \text{ if and only if } g(n)=\theta(f(n))$$

### ترانهاده تقارنی (transpose symmetry)

$$f(n)=O(g(n)) \text{ if and only if } g(n)=\Omega(f(n))$$
$$f(n)=o(g(n)) \text{ if and only if } g(n)=\omega(f(n))$$

**نکته:** *همانطور که قبلاً گفته شد با استفاده از حد می‌توان رشد توابع را تشخیص داد:*

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\begin{cases}\infty \Leftrightarrow f(n)\in\omega(g(n)) \Rightarrow f(n)\in\Omega(g(n)), \text{but } f(n)\notin\theta(g(n))\\ 0 \Leftrightarrow f(n)\in o(g(n)) \Rightarrow f(n)\in O(g(n)), \text{but } f(n)\notin\theta(g(n))\\ k\neq 0 \Leftrightarrow f(n)\in\theta(g(n)) \Rightarrow f(n)\in O(g(n)), \text{and } f(n)\in\Omega(g(n))\end{cases}$$

**نکته:** *اگر $a$ و $b$ اعداد حقیقی باشند و معادل درجه $f(n)$ و $g(n)$ باشند می‌توان گفت:*

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

**توجه مهم:** اگرچه برای هر دو عدد حقیقی a و b می‌توان گفت یا $a<b$ یا $a>b$ یا $a=b$ ولی برای هر دو تابع $f(n)$ و $g(n)$ ممکن است نه $f(n)=O(g(n))$ و نه $f(n)=\Omega(g(n))$ مثلاً توابع $n$ و $n^{1+\sin n}$ با نمادهای مجانبی قابل مقایسه نیستند چون $1+\sin n$ بین ۰ و ۲ نوسان می‌کند.

**نکته:** *زمان اجرای یک الگوریتم $\theta(g(n))$ است اگر و فقط اگر بدترین حالت زمان $O(g(n))$ و بهترین حالت $\Omega(g(n))$ باشد.*

**مثال:** درستی یا نادرستی گزاره‌های زیر را بررسی کنید.

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

۲) اگر $f(n)\in o(g(n))$ آنگاه $2^{f(n)}\in o(2^{g(n)})$

۳) اگر $f(n)\in O(g(n))$ آنگاه $\lg f(n)\in O(\lg g(n))$

۴) $f(n)+g(n)\in \theta(\min\{f(n),g(n)\})$

۵) $f(n)\in O(f^2(n))$

۶) $f(n)\in \theta\left(f(\frac{n}{2})\right)$

۷) $f(n)+O(f(n))\in \theta(f(n))$

۸) $\max(f(n),g(n))=\theta(f(n)+g(n))$

۹) $O(f(n))-\Omega(f(n)),o(f(n))$ دو مجموعه مساوی هستند.

۱۰) $f(n)=|n^2\sin n|$، $f(n)\in O(n)$ یا $f(n)\in \Omega(n)$.

۱۱) به ازای هر دو تابع $f(n)$ و $g(n)$، یا $f(n)=O(g(n))$ یا $f(n)=\Omega(g(n))$.

۱۲) اگر $\lg f(n)\in O(\lg n)$ آنگاه $f(n)$ محدود به چند جمله‌ای است و برعکس.

۱۳) $a,b>0$ ،$\lg^b n=o(n^a)$

۱۴) $o(g(n))\cap\omega(g(n))$ مجموعه تهی است.

۱۵) علامت $\tilde{\Omega}$ را بخوانید «اُمگا بی نهایت»: $f(n)\in\tilde{\Omega}(g(n))$ اگر به ازای حداقل یک $C>0$، نامساوی $f(n)\le cg(n)\le 0$ به ازای بی شمار $n$ صحیح، درست باشد. آیا می‌توان ادعا کرد $f(n)\in O(g(n))$ یا $f(n)\in\tilde{\Omega}(g(n))$ حداقل یکی صحیح است.

**پاسخ:**

۱) نادرست. مثلاً $f(n)=2n$ و $g(n)=n$

۲) درست

۳) درست.

$$f(n)=O(g(n))\Rightarrow f(n)\le Cg(n)\Rightarrow Lg(f(n))\le Lgc+Lg(g(n))$$

برای آنکه نشان دهیم این عبارت کمتر از $b.Lg\,g(n)$ است (برای مقدار ثابتی مثل b)، فرض می‌کنیم $Lgc + Lg(g(n)) = b.Lg(g(n))$ حال طرفین را به $Lg(g(n))$ تقسیم می‌کنیم:

$$b = \frac{Lg(c) + Lg(g(n))}{Lg(g(n))} = \frac{Lg(c)}{Lg(g(n))} + ۱ \le Lgc + ۱$$

۴) نادرست

۵) اگر برای nهای بزرگ $f(n) < ۱$ آنگاه نادرست و اگر $f(n) > ۱$ آنگاه درست.

۶- نادرست

۷- درست

۸- درست. باید نشان دهیم اعداد مثبت $n_۰$ و $C_۱$ و $C_۲$ وجود دارند که

$$۰ \le C_۱(f(n) + g(n)) \le \max(f(n), g(n)) \le C_۲(f(n) + g(n))$$

برای همه $n \ge n_۰$. کافیست $C_۲ = ۱$ و $C_۱ = \frac{۱}{۲}$ انتخاب شوند.

۹- نادرست. به عنوان مثال $g(n) = \begin{cases} n, & n \text{ زوج} \\ ۱, & n \text{ فرد} \end{cases}$، می‌توان نشان داد $g(n) \in O(n) - \Omega(n)$ ولی $g(n) \notin o(n)$

۱۰- اگر n طبیعی فرض شود $|n^۲ \sin n| \in \Omega(n)$ ولی اگر n حقیقی فرض شود نادرست است.

۱۱- نادرست

۱۲- درست

۱۳- درست

۱۴- درست

۱۵- درست

**مثال:** بیان کنید چرا جمله «زمان اجرای الگوریتم A حداقل $O(n^۲)$ است»، بی معنی است.

**پاسخ:** فرض کنید $T(n)$ زمان اجرای الگوریتم A باشد. $T(n) \ge O(n^۲)$ یعنی $T(n) \ge f(n)$ که $f(n)$ عضو $O(n^۲)$ است. از آنجایی که تمام توابع با رشد کمتر مساوی $n^۲$، عضو $O(n^۲)$ هستند مثل $g(n) = ۰$، پس $T(n) \ge O(n^۲)$ یعنی مثلاً $T(n) \ge ۰$ و این بدیهی است زیرا زمان اجرا همیشه بیشتر مساوی صفر است.

**مثال:** آیا $\lceil \lg n \rceil !$ محدود به چند جمله‌ای است؟

**پاسخ:** فرض کنید $f(n) = \lceil \lg n \rceil !$. اگر قرار باشد $f(n)$ محدود به چند جمله‌ای باشد باید $\lg f(n) = O(\lg n)$:

$$\lg f(n) = \lg \lceil \lg n \rceil ! = \theta\left(\lceil \lg n \rceil . \lg \lceil \lg n \rceil\right) = \theta(\lg n . \lg \lg n) \Rightarrow \lg f(n) \ne O(\lg n)$$

پس $\lg\lceil \lg n\rceil!$ محدود به چند جمله‌ای نیست. (دقت کنید که طبق $\lg n! = \theta(n \lg n)$ پس $\lg\lceil \lg n\rceil! = \theta(\lceil \lg n\rceil . \lg\lceil \lg n\rceil)$).

**مثال:** آیا $f(n) = \lceil \lg \lg n\rceil!$ محدود به چند جمله‌ای است؟

**پاسخ:**

$$\lg f(n) = \lg\lceil \lg \lg n\rceil! = \theta\left(\lceil \lg \lg n\rceil . \lg\lceil \lg \lg n\rceil\right) = \theta(\lg \lg n . \lg \lg \lg n) = o\left((\lg \lg n)^2\right).$$

$$= o(\lg^2 \lg n) = o(\lg n)$$

پس $\lceil \lg \lg n\rceil!$ محدود به چند جمله‌ای است.

**توجه:** می‌دانیم $\left(\lg^b n = o(n^a)\right)$ یعنی $\lg^b n < n^a$، قرار دهید $a = 1$ و $b = 2$ و بجای n قرار دهید lgn:

$$\lg^2 \lg n = o(\lg n)$$

**توجه:** می‌توان مفهوم مرتبه‌ها را به ۲ پارامتر مستقل m و n تعمیم داد:

$$O\big(g(n,m)\big) = \{f(n,m) : \exists c, n_0, m_0 > 0, \forall n \geq n_0, m \geq m_0, 0 \leq f(n,m) \leq cg(n,m)\}$$

$$\Omega\big(g(n,m)\big) = \{f(n,m) : \exists c, n_0, m_0 > 0, \forall n \geq n_0, m \geq m_0, 0 \leq cg(n,m) \leq f(n,m)\}$$

$$\theta\big(g(n,m)\big) = \{f(n,m) : \exists c_1, c_2, n_0, m_0 > 0, \forall n \geq n_0, m \geq m_0, 0 \leq c_1 g(n,m) \leq f(n,m) \leq c_2 g(n,m)\}$$

## بازگشتی

یک رابطه یا معادله بازگشتی، رابطه یا معادله‌ای است که هر جمله را برحسب یک یا چند جمله قبلی بیان کند. مثلاً $F_n = F_{n-1} + F_{n-2}$ یک معادله بازگشتی است که جمله nام را برحسب دو جمله قبلی بیان می‌کند. این رابطه را مرتبه ۲ گویند و نیاز به دو مقدار اولیه دارد مثلاً $F_0 = 0$ و $F_1 = 1$. با این ۲ مقدار اولیه می‌توان همه جملات را یافت:

$$F_2 = F_1 + F_0 = 1 + 0 = 1$$

$$F_3 = F_2 + F_1 = 1 + 1 = 2$$

پس جملات این دنباله عبارت است از:

$$0, 1, 1, 2, 3, 5, 8, \ldots$$

همانطور که می‌دانیم، این دنباله فیبوناتچی است.

شکل کلی رابطه بازگشتی خطی مرتبه k با ضرایب ثابت به شکل زیر است:

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k} + f(n)$$

که نیاز به k شرط اولیه دارد مثلاً: $a_0$ و $a_1$ و ... و $a_{k-1}$. مقادیر $c_1, c_2, \ldots, c_k$ ثابت هستند و $c_k \neq 0$.

اگر $f(n) = 0$، رابطه را همگن گویند و اگر $f(n) \neq 0$، ناهمگن گویند.

اگر $f(n) = ۰$، برای حل، معادله مشخصه[۱] تشکیل می‌دهیم:

$$x^k - c_۱x^{k-۱} - c_۲x^{k-۲} - \ldots - c_k = ۰$$

اگر $k$ ریشه حقیقی متمایز $(x_k, \ldots, x_۲, x_۱)$ وجود داشته باشد آنگاه، جواب این رابطه بازگشتی عبارت است از:

$$a_n = \alpha_۱(x_۱)^n + \alpha_۲(x_۲)^n + \ldots + \alpha_k(x_k)^n$$

برای یافتن $\alpha_۱$ و $\alpha_۲$ و ... $\alpha_k$، شرایط اولیه را در رابطه فوق قرار دهید.

اگر $x_۱ = x_۲$ مضاعف و سایر ریشه‌ها حقیقی متمایز باشند آنگاه:

$$a_n = (\alpha_۱ + \alpha_۲.n)(x_۱)^n + \alpha_۳(x_۳)^n + .. + \alpha_k(x_k)^n$$

اگر $x_۱ = x_۲ = x_۳$ ریشه مرتبه ۳ و سایر ریشه‌ها حقیقی متمایز باشند آنگاه:

$$a_n = (\alpha_۱ + \alpha_۲n + \alpha_۳n^۲)(x_۱)^n + \ldots + \alpha_k(x_k)^n$$

و به همین ترتیب.

**مثال:** مطلوبست حل رابطه بازگشتی زیر:

$$\begin{cases} a_n = ۳a_{n-۱} - ۲a_{n-۲} \\ a_۰ = ۱, a_۱ = ۰ \end{cases}$$

**پاسخ:**

مشخصه: $x^۲ - ۳x + ۲ = ۰ \Rightarrow (x-۱)(x-۲) = ۰$

$$\Rightarrow a_n = \alpha_۱(۱)^n + \alpha_۲(۲)^n$$

$$\Rightarrow \begin{cases} a_۰ = \alpha_۱ + \alpha_۲ = ۱ \\ a_۱ = \alpha_۱ + ۲\alpha_۲ = ۰ \end{cases} \Rightarrow \alpha_۱ = ۲ , \alpha_۲ = -۱ \Rightarrow a_n = ۲ - ۲^n$$

اگر رابطه بازگشتی ناهمگن باشد و

$$f(n) = P_۱(n) \times b_۱^n + P_۲(n) \times b_۲^n + \ldots + P_m(n) \times b_m^n$$

یعنی:

$$a_n = c_۱a_{n-۱} + \ldots + c_ka_{n-k} + p_۱(n).b_۱^n + P_۲(n).b_۲^n + \ldots p_m(n).b_m^n$$

و درجه چند جمله‌ای $P_i(n)$ برابر $d_i$ باشد آنگاه معادله مشخصه عبارت است از:

$$\left(x^k - c_۱x^{k-۱} - c_۲x^{k-۲} \ldots - c_k\right)(x-b_۱)^{d_۱+۱}(x-b_۲)^{d_۲+۱}\ldots(x-b_m)^{d_m+۱} = ۰$$

**مثال:**

$$a_n = ۳a_{n-۱} - ۲a_{n-۲} + n \times ۴^n + ۳$$

---

۱- برای مطالعه بیشتر به کتاب ساختمان گسسته پوران پژوهش رجوع کنید.

با توجه به اینکه $f(n) = (n)\times 4^n + (3)\times 1^n$ پس $b_1 = 4$ و $d_1 = 1$ و $b_2 = 1$ و $d_2 = 0$ بنابراین معادله مشخصه:

$$\left(x^2 - 3x + 2\right)(x-4)^2(x-1)^1 = 0 \Rightarrow (x-2)(x-1)^2(x-4)^2 = 0$$

$$\Rightarrow a_n = \alpha_1(2)^n + (\alpha_2 + \alpha_3 n)(1)^n + (\alpha_4 + \alpha_5 n)(4)^n = \theta(n\,4^n)$$

**مثال:**

$$a_n = 3a_{n-1} - 3a_{n-2} + a_{n-3} + n + 2^n$$

**پاسخ:**

$$\left(x^3 - 3x^2 + 3x - 1\right)(x-1)^2(x-2)^1 = 0$$

$$(x-1)^5(x-2) = 0 \Rightarrow a_n = \left(\alpha_1 + \alpha_2 n + \alpha_3 n^2 + \alpha_4 n^3 + \alpha_5 n^4\right)(1)^n + \alpha_6(2)^n$$

همانطور که دیدیم، یکی از روش‌های حل روابط بازگشتی، تشکیل معادله مشخصه است. این روش برای روابط بازگشتی خطی با ضرایب ثابت قابل استفاده است.

روش‌های دیگری مثل تغییر متغیر، جایگذاری، استقراء، قضیه Master و درخت بازگشت نیز مطرح هستند که در ادامه بررسی می‌کنیم.

**مثال:**

$$T(n) = 2T\left(\frac{n}{2}\right) + n \quad , \quad T(1) = 1$$

**پاسخ:** تغییر متغیر $n = 2^m$ را انجام می‌دهیم:

$$T(2^m) = 2T(2^{m-1}) + 2^m \xrightarrow{F(m)=T(2^m)} F(m) = 2F(m-1) + 2^m \Rightarrow (x-2)(x-2) = 0$$

$$\Rightarrow F(m) = (\alpha_1 + \alpha_2 m)(2)^m \Rightarrow T(n) = (\alpha_1 + \alpha_2 . \lg n).n$$

در این معادله، دو مجهول $\alpha_1$ و $\alpha_2$ وجود دارد و با توجه به شرط اولیه $T(1) = 1$ از معادله صورت سوال، $T(2)$ را محاسبه می‌کنیم و سپس مجهولات $\alpha_1$ و $\alpha_2$ محاسبه می‌شوند:

$$T(2) = 2T(1) + 2 = 4$$

بنابراین:

$$\begin{cases} T(1) = (\alpha_1 + 0) = 1 \\ T(2) = (\alpha_1 + \alpha_2)(2) = 4 \end{cases}$$

$$\Rightarrow \alpha_1 = 1 \, , \, \alpha_2 = 1 \Rightarrow T(n) = (1 + \lg n)n$$

**مثال:**

$$T(n) = \frac{3}{2}T\left(\frac{n}{2}\right) - \frac{1}{2}T\left(\frac{n}{4}\right) - \frac{1}{n} \, , \, T(1) = 1 \, , \, T(2) = \frac{3}{2}$$

**پاسخ:**

$$n = 2^m \Rightarrow T(2^m) = \frac{3}{2}T(2^{m-1}) - \frac{1}{2}T(2^{m-2}) - \frac{1}{2^m}$$

$$\Rightarrow F(m) = \frac{3}{2}F(m-1) - \frac{1}{2}F(m-2) - \left(\frac{1}{2}\right)^m$$

$$\Rightarrow \left(x^2 - \frac{3}{2}x + \frac{1}{2}\right)\left(x - \frac{1}{2}\right) = 0 \Rightarrow \left(x - \frac{1}{2}\right)^2 (x-1) = 0$$

$$\Rightarrow F(m) = (\alpha_1 + \alpha_2 m)\left(\frac{1}{2}\right)^m + \alpha_3 (1)^m \Rightarrow T(n) = (\alpha_1 + \alpha_2 \lg n)\frac{1}{n} + \alpha_3$$

با توجه به شرایط اولیه:

$$T(n) = \frac{\lg n}{n} + 1$$

**مثال:**

$$a_n + n a_{n-1} = 2n! \ , \ a_0 = 1$$

**پاسخ:** به $n!$ تقسیم می‌کنیم:

$$\frac{a_n}{n!} + \frac{a_{n-1}}{(n-1)!} = 2$$

قرار می‌دهیم $b_n = \frac{a_n}{n!}$:

$$b_n + b_{n-1} = 2 \Rightarrow (x+1)(x-1) = 0 \Rightarrow b_n = \alpha_1 (-1)^n + \alpha_2 (1)^n \Rightarrow a_n = n!\left(\alpha_1 (-1)^n + \alpha_2\right)$$

با توجه به $a_0 = 1$ و $a_1 = 1$:

$$\begin{cases} a_0 = \alpha_1 + \alpha_2 = 1 \\ a_1 = -\alpha_1 + \alpha_2 = 1 \end{cases} \Rightarrow \alpha_2 = 1 \ , \ \alpha_1 = 0 \Rightarrow a_n = n!$$

**مثال:**

$$T(n) = \frac{1 + T(n-1)}{T(n-2)} \ , \ T(0) = a \ , \ T(1) = b$$

**پاسخ:**

$$T(0) = a \ , \ T(1) = b \ , \ T(2) = \frac{1+b}{a} \ , \ T(3) = \frac{1+a+b}{ab}$$

$$T(4) = \frac{1+a}{b} \ , \ T(5) = a \ , \ T(6) = b \ , \ T(7) = \frac{1+b}{a} \ldots$$

$$\Rightarrow T(n) = T(n \bmod 5) \ , \ n > 4$$

**مثال:**

$$t(n)=\frac{۱}{۴-t(n-۱)}\ ,\ t(۱)=۱$$

**پاسخ:**

$$t(۲)=\frac{۱}{۳}\ ,\ t(۳)=\frac{۳}{۱۱}\ ,\ t(۴)=\frac{۱۱}{۴۱}\ldots$$

$$t(n)=\frac{p_n}{q_n}\ ,\ t_{n+۱}=\frac{p_{n+۱}}{q_{n+۱}}=\frac{۱}{۴-\frac{p_n}{q_n}}=\frac{q_n}{۴q_n-p_n}$$

با توجه به اینکه کسرها، تحویل ناپذیر هستند:

$$p_{n+۱}=q_n\ ,\ q_{n+۱}=۴q_n-p_n$$

$$\Rightarrow q_{n+۱}=۴q_n-q_{n-۱}\Rightarrow x^۲-۴x+۱=۰\Rightarrow x=۲\pm\sqrt{۳}$$

$$\Rightarrow q_n=\alpha_۱(۲+\sqrt{۳})^n+\alpha_۲(۲-\sqrt{۳})^n\ ,\ q_۲=۳\ ,\ q_۳=۱۱$$

با حل این، $q_n$ بدست می‌آید و سپس $p_n$ و سپس $t(n)$

## الگوریتم‌های بازگشتی

الگوریتم بازگشتی، الگوریتمی است که خودش را فراخوانی کند. مثلاً تابع زیر یک تابع بازگشتی است:

$$f(n)=\begin{cases}n*f(n-۱) & ,\ n>۱\\ ۱ & ,\ n=۰\ or\ n=۱\end{cases}$$

این تابع $n!$ را محاسبه می‌کند. اگر $T(n)$ تعداد فراخوانی‌های این تابع به ازای $n$ باشد می‌توان نوشت:

$T(n)=\begin{cases}۱+T(n-۱) & ,\ n>۱\\ ۱ & ,\ n=۰\ or\ n=۱\end{cases}$ پس $T(n)=n=\theta(n)$ یعنی $n$ بار فراخوانی انجام می‌شود

یعنی زمان اجرای این تابع از مرتبه $\theta(n)$ است.

در ادامه تعدادی تابع بازگشتی معروف آمده است.

۱- مجموع ۱ تا n:

$$f(n)=۱+۲+\ldots+n-۱+n=\frac{n(n+۱)}{۲}$$

$$f(n)=\begin{cases}۱ & ,\ n=۱\\ f(n-۱)+n & ,\ n>۱\end{cases}$$

اگر $T(n)$ تعداد فراخوانی این تابع باشد آنگاه:

$$T(n)=\begin{cases}۱ & ,\ n=۱\\ T(n-۱)+۱ & ,\ n>۱\end{cases}$$

پس $T(n)=\theta(n)$

۲- توان

$$f(m,n) = m^n = m * m^{n-1}$$

$$f(m,n) = \begin{cases} m * f(m,n-1) & , \ n > 1 \\ m & , \ n = 1 \end{cases}$$

تعداد فراخوانی‌های این تابع نیز از مرتبه $\theta(n)$ است.

البته می‌توان $m^n$ را با تابع بازگشتی زیر نیز محاسبه کرد که مرتبه آن $\theta(\lg n)$ است:

$$m^n = \left( m^{\frac{n}{2}} \right)^2 \quad n \neq 0 \text{ زوج و}$$

$$m^n = \left( m^{\left\lfloor \frac{n}{2} \right\rfloor} \right)^2 * m \quad n \text{ فرد و}$$

$$\Rightarrow f(m,n) = \begin{cases} \left( f(m, \frac{n}{2}) \right)^2 & , \ n \neq 0 \text{ is even} \\ \left( f(m, \left\lfloor \frac{n}{2} \right\rfloor) \right)^2 * m & , \ n \text{ is odd} \\ 1 & , \ n = 0 \end{cases}$$

۳- خارج قسمت m به n $\left( \left\lfloor \frac{m}{n} \right\rfloor \right)$

$$f(m,n) = \begin{cases} 0 & , \ m < n \\ f(m-n,n) + 1 & , \ m \geq n \end{cases}$$

۴- باقی مانده m به n $(m \bmod n)$:

$$f(m,n) = \begin{cases} m & , \ m < n \\ f(m-n,n) & , \ m \geq n \end{cases}$$

۵- ب.م.م m و n [1]: (بزرگترین مقسوم علیه مشترک m و n)

$$f(m,n) = \begin{cases} f(n,m) & , \ m < n \\ m & , \ n = 0 \\ f(n, m \bmod n) & , \ \text{else} \end{cases}$$

1- Greatest Common Divisor

۶- لگاریتم n در پایه ۲ ($\lfloor \lg n \rfloor$):

$$f(n)=\begin{cases} f\left(\left\lfloor \frac{n}{۲} \right\rfloor\right)+۱ & , \quad n>۱ \\ ۰ & , \quad n=۱ \end{cases}$$

۷- ترکیب $\binom{m}{n}$:

$$f(m,n)=\begin{cases} ۱ & , \quad m=n \quad \text{or} \quad n=۰ \\ f(m-۱,n)+f(m-۱,n-۱) & , \text{ else and } m>n \end{cases}$$

۸- آکرمان

$$f(m,n)=\begin{cases} n+۱ & , \quad m=۰ \\ f(m-۱,۱) & , \quad m\neq ۰,\ n=۰ \\ f(m-۱\ ,\ f(m,n-۱)) & , \quad m,n>۰ \end{cases}$$

در این تابع همیشه $f(۱,n)=n+۲$ است. این تابع را غیربازگشتی نیز می‌توان نوشت. ولی به هر حال رشد این تابع بسیار شدید است.

۹- جایگشت: تابع perm تمام جایگشت‌های n عنصر که در آرایه a[۰...n-۱] ذخیره شده اند را چاپ می‌کند. n عنصر دارای !n جایگشت هستند.

ایده بازگشتی الگوریتم این است که مثلاً جایگشت‌های ۴ عنصر (a , b, c , d) را می‌توان به شکل زیر ساخت:

۱) a به همراه جایگشت‌های (b , c , d)

۲) b به همراه جایگشت‌های (a , c , d)

۳) c به همراه جایگشت‌های (a , b , d)

۴) d به همراه جایگشت‌های (a , b , c)

```
void perm (char * a , int k, int n)
 {
   if (k == n - ۱)
      for (int i=0; i<n ; i++)
          cout << a[i];
   else
      for (i=k ; i<n ; i++)
        {
          Swap (a[i] , a[k]);
          perm (a , k + ۱, n) ; // a[k + ۱]...a[n - ۱]      تمام جایگشت‌های
          Swap (a[i] , a[k]) ; //                          ترتیب اولیه حاصل شود
        }
 }
```

این تابع باید به صورت perm(a , ۰ , n) فراخوانی شود.

**محاسبه زمان اجرا:** وقتی $k = n - ۱$ باشد، زمان $\theta(n)$ است (if).

وقتی $k < n - ۱$ باشد، else اجرا می‌شود و حلقه for آن به تعداد n-k بار اجرا می‌شود و هر بار زمان آن $\theta\left(t_{perm}(k+۱, n-۱)\right)$ است. بنابراین:

$$t_{perm}(k, n-۱) = \theta\left((n-k).t_{perm}(k+۱, n-۱)\right) \ , \ k < n-۱$$

$$\Rightarrow \quad t_{perm}(۰, n-۱) = \theta\left(n(n!)\right) \ , \ n \geq ۱$$

۱۰- هورنر: برای ارزیابی یک چند جمله‌ای درجه n، در یک نقطه با روش عادی، عمل $\frac{n(n+۱)}{۲}$ ضرب لازم است ولی با روش هورنر تعداد ضرب n است:

$$A(x) = a_n x^n + a_{n-۱} x^{n-۱} + \ldots + a_۱ x + a_۰$$

می خواهیم $A(x)$ را در $x_۰$ محاسبه کنیم یعنی $A(x_۰)$ را بیابیم. با روش عادی:

$$A(x_۰) = a_n x_۰^n + a_{n-۱} x_۰^{n-۱} + \ldots + a_۱ x_۰ + a_۰$$

$$\text{تعداد ضرب} = n + n - ۱ + \ldots + ۱ + ۰ = \frac{n(n+۱)}{۲}$$

دقت کنید جمله مثلاً $a_۲x^۲$ نیاز به ۲ عمل ضرب دارد $(a_۲ \times x \times x)$.

روش هورنر، $A(x_۰)$ را به شکل زیر محاسبه می‌کند:

$$A(x_۰) = \left(\ldots(a_n x_۰ + a_{n-۱})x_۰ + \ldots + a_۱\right)x_۰ + a_۰$$

که n ضرب دارد. ضرایب $a_۰$ و ... $a_n$ و یک مقدار برای x داده می‌شود، الگوریتم زیر با روش هورنر، $A(x)$ را محاسبه می‌کند:

$y \leftarrow ۰$ , $i \leftarrow n$

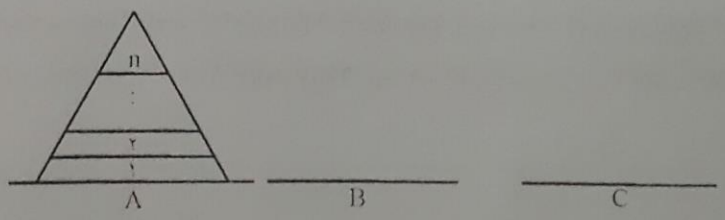
while $i \geq ۰$ do $y \leftarrow a_i + x \cdot y$, $i \leftarrow i - ۱$

۱۱- هانوی:

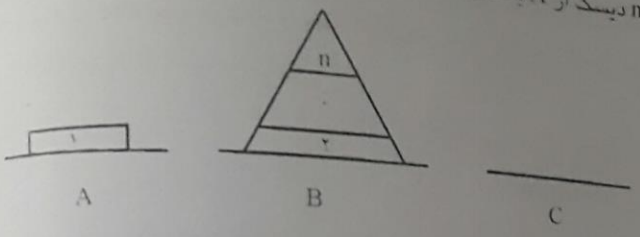
**مثال:** در مسئله برج‌های هانوی، ۳ میله A (مبدا)، B (کمکی)، C (مقصد) وجود دارد.

تعداد n دیسک با اندازه‌های متفاوت از بزرگ به کوچک در میله A قرار گرفته است، هدف انتقال این دیسک‌ها از میله A به میله C با کمک میله B است به شرطی که هر بار فقط یک انتقال مجاز باشد و هیچگاه دیسک کوچک را روی بزرگ نگذاریم. زمان اجرای الگوریتم را محاسبه کنید.

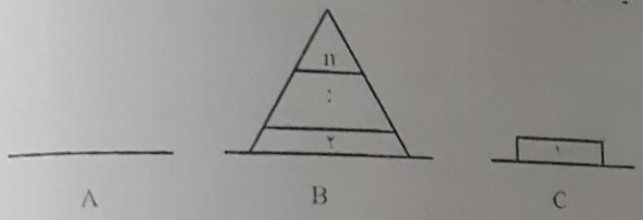
**پاسخ:** مسئله انتقال n دیسک از A به C با کمک B را می‌توان به سه زیر مسئله تقسیم کرد:



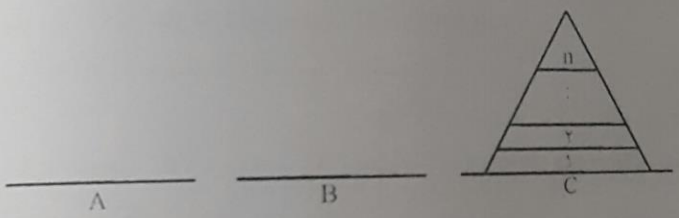
۱) انتقال n-۱ دیسک از A به B با کمک C با شرایط گفته شده:




۲) انتقال یک دیسک از A به C




۳) انتقال n-۱ دیسک از B به C با کمک A




با توجه به مراحل، زیر برنامه‌هانوی را خودتان بنویسید. اگر $T(n)$ را برابر تعداد انتقال‌ها (زمان)هانوی فرض کنیم. با توجه به سه مرحله فوق می‌توان نوشت:

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

یعنی

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

حال می‌توان این رابطه بازگشتی را با جایگذاری (iteration) حل کرد:

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

$$\Rightarrow T(n)=2^3T(n-3)+2^2+2+1 \Rightarrow T(n)=2^{n-1}T(1)+2^{n-2}+2^{n-3}+\ldots+2+1$$

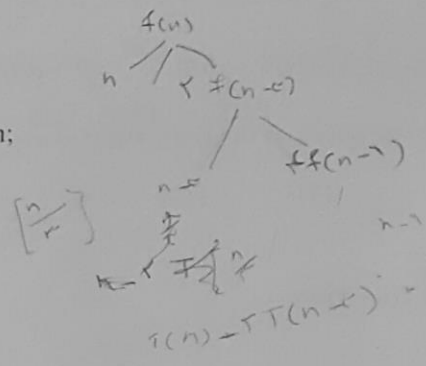
$$\Rightarrow T(n)=2^{n-1}+2^{n-2}+\ldots+2+1=2^n-1 \Rightarrow T(n)=\theta(2^n)$$

## محاسبه زمان اجرای الگوریتم‌های بازگشتی

**مثال:** با توجه به توابع بازگشتی زیر، مرتبه زمان هر کدام را بیان کنید:

۱)
```
f (int n)
{
  if (n <= ۱) return n;
  return f(n − ۳) + f(n − ۳) + n;
}
```

۲)
```
f (int n)
{
  if (n <= ۱) return n;
  return ۲f(n − ۳) + n;
}
```



**پاسخ:** ۱) اگر $T(n)$ زمان اجرای تابع فرض شود می‌توان نوشت:

$$T(n) = T(n-۳) + T(n-۳) + ۱ = ۲T(n-۳) + ۱ \quad , \quad T(۱) = ۱$$

زیرا به ازای هر n که f را فراخوانی می‌کنیم، f خودش را ۲ بار با $n-۳$ فراخوانی می‌کند که زمان هر فراخوانی $T(n-۳)$ است، و در هر فراخوانی یک عمل ثابت انجام می‌شود. با حل این رابطه بازگشتی داریم:

$$T(n) = ۲T(n-۳) + ۱ = \theta\left(۲^{\frac{n}{۳}}\right)$$

۲) این تابع یک بار با $n-۳$ خودش را صدا می‌زند:

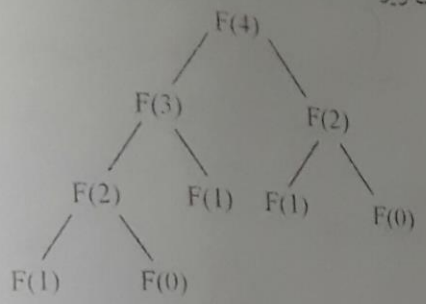
$$T(n) = T(n-۳) + ۱ = \theta(n)$$

**نکته:** *اگر زمان اجرای الگوریتمی $T(n)$ باشد و بتوان نوشت:* $T(n) = a.T(n-b) + c$ *که* $c$ *و* $b$ *و* $a$ *ثابت و* $b > ۰$ *و* $a \neq ۱$ *آنگاه* $T(n) = \theta\left(a^{\frac{n}{b}}\right)$ *و اگر* $a = ۱$ *آنگاه* $T(n) = \theta(n)$

**مثال:** برای محاسبه جمله nام فیبوناچی می‌توان تابع بازگشتی زیر را نوشت. برای محاسبه $F(۴)$ تعداد فراخوانی‌ها چندتاست؟ اگر $T(n)$ تعداد فراخوانی‌ها برای n باشد، برای $T(n)$ یک رابطه بازگشتی بنویسید.

```
F (int n)
{
  if (n <= ۱) return n;
  return F(n − ۱) + F(n − ۲);
}
```

**پاسخ:** با توجه به درخت زیر، تعداد فراخوانی‌ها، ۹ تاست.




می‌توان نوشت:

$$T(n) = T(n-1) + T(n-2) + 1$$
$$T(0) = T(1) = 1$$

**توجه:** می‌توان نشان داد در این مثال $\theta\left(\frac{n}{2^2}\right) < T(n) < \theta(2^n)$.

زیرا:

$$T(n) = T(n-1) + T(n-2) + 1 > 2T(n-2) + 1 = \theta\left(\frac{n}{2^2}\right)$$
$$T(n) = T(n-1) + T(n-2) + 1 < 2T(n-1) + 1 = \theta\left(2^n\right)$$

**قضیه Master:**

اگر در رابطه بازگشتی $T(n) = aT\left(\frac{n}{b}\right) + f(n)$ $a \geq 1$ و $b > 1$ ثابت باشند و $f(n)$ مثبت باشد و $\frac{n}{b}$ ممکن است به صورت $\left\lfloor \frac{n}{b} \right\rfloor$ یا $\left\lceil \frac{n}{b} \right\rceil$ باشد آنگاه:

۱) اگر $f(n) = O\left(n^{\log_b^{a-\varepsilon}}\right)$ برای ثابتی مثل $\varepsilon > 0$ آنگاه $T(n) = \theta\left(n^{\log_b^a}\right)$

۲) اگر $f(n) = \theta\left(n^{\log_b^a}\right)$ آنگاه $T(n) = \theta\left(n^{\log_b^a} . \lg n\right)$

۳) اگر $f(n) = \Omega\left(n^{\log_b^{a+\varepsilon}}\right)$ برای ثابتی مثل $\varepsilon > 0$ و اگر $af\left(\frac{n}{b}\right) \leq cf(n)$ برای بعضی ثابت $c < 1$ و همه مقادیر به اندازه کافی بزرگ n آنگاه $T(n) = \theta(f(n))$

**توجه:** برای درک بهتر (ولی نادقیق) اگر درجه $n^{\log_b^a}$ از $f(n)$ بیشتر بود جواب $\theta\left(n^{\log_b^a}\right)$ است (حالت ۱)، اگر درجه $n^{\log_b^a}$ از $f(n)$ کمتر بود آنگاه جواب $\theta\left(f(n)\right)$ است (حالت ۳)، و اگر $n^{\log_b^a}$ و $f(n)$ هم درجه بودند جواب $\theta\left(n^{\log_b^a}.\lg n\right)=\theta\left(f(n).\lg n\right)$ است (حالت ۲)

اگر $f(n)$ چند جمله‌ای باشد توجه نادقیق فوق همواره صحیح است.

**مثال:**

$$T(n)=۹T\left(\frac{n}{۳}\right)+n=\theta(n^۲)$$

$$T(n)=T\left(۲\frac{n}{۳}\right)+۱=\theta(\lg n)$$

$$T(n)=۳T\left(\frac{n}{۴}\right)+n\lg n$$

$a=۳$ و $b=۴$ و $f(n)=n\lg n$ و $n^{\log_۴^۳}=O\left(n^{۰/۷۹۳}\right)$ و از آنجایی که $f(n)=\Omega\left(n^{\log_۴^{۲+\varepsilon}}\right)$

که $\varepsilon\simeq ۰/۲$ حالت ۳ به شرطی برقرار است که نشان دهیم $af\left(\frac{n}{b}\right)\le cf(n)$:

$$af\left(\frac{n}{b}\right)=۳\left(\frac{n}{۴}\right)\lg\left(\frac{n}{۴}\right)\le\frac{۳}{۴}n\lg n$$

که به ازای $c=\frac{۳}{۴}$ درست است پس $T(n)=\theta(n\lg n)$

**توجه:** قضیه Master برای $T(n)=۲T\left(\frac{n}{۲}\right)+n\lg n$ برقرار نیست اگرچه ظاهراً a=۲ و b=۲ و $f(n)=n\lg n$ و $n^{\log_b^a}=n$ به نظر می‌رسد حالت ۳ برقرار است. ولی مشکل این است که $f(n)$ به صورت چند جمله‌ای از n بزرگتر نیست. نسبت $\frac{f(n)}{n^{\log_b^a}}=\lg n$ که از $n^{\varepsilon}$ (چند جمله ای) کمتر است و در نتیجه نمی توان از Master استفاده کرد. برای این روابط می‌توان از تبصره زیر کمک گرفت:

**تبصره:** اگر $T(n)=aT\left(\frac{n}{b}\right)+n^{\log_b^a}\times(\lg n)^k$ آنگاه $T(n)=\theta\left(n^{\log_b^a}\times(\lg n)^{k+۱}\right)$

پس

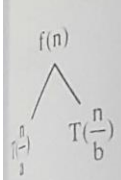
$$T(n)=۲T\left(\frac{n}{۲}\right)+n\lg n=\theta(n\lg^۲ n)$$

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2 \lg^2 n = \theta(n^2 \lg^4 n)$$

## درخت بازگشت

برای حل یک رابطه بازگشتی و یافتن مرتبه اجرایی، یک روش، رسم درخت بازگشت است.

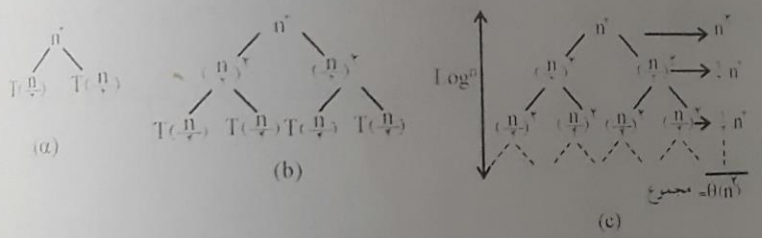
برای رابطه $T(n) = T(\frac{n}{a}) + T(\frac{n}{b}) + f(n)$ شکل درخت به صورت مقابل است:




که $T(\frac{n}{a})$ و $T(\frac{n}{b})$ طبق رابطه بازگشتی داده شده بسط داده می‌شوند تا به شرایط اولیه برسیم.

در نهایت همه نودهای درخت را جمع می‌کنیم.

به عنوان مثال درخت بازگشت برای رابطه $T(n) = 2T\left(\frac{n}{2}\right) + n^2$ به شکل زیر است:



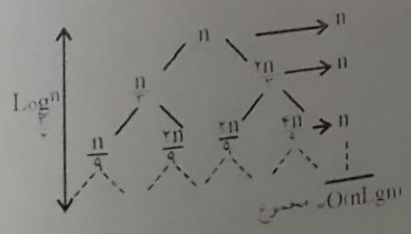

(c)

برای راحتی فرض کرده ایم n توانی از ۲ است. درخت را باید آنقدر بشکنیم تا به شرایط اولیه برسیم. در شکل فوق قسمت (c) نتیجه نشان داده شده است. حال باید مجموع نودها را بیابیم.

در اولین سطح $n^2$، در سطح دوم $\frac{n^2}{4} + \frac{n^2}{4} = \frac{n^2}{2}$، در سطح سوم $4\left(\frac{n}{4}\right)^2 = \frac{n^2}{4}$ و به همین ترتیب

چون تصاعدی کم می‌شوند پس مجموع آنها ضریبی از خود $n^2$ خواهد بود یعنی $T(n) = \theta(n^2)$

**مثال:** رابطه بازگشتی $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n$ را در نظر بگیرید درخت آن به شکل زیر است:




(c)

در هر سطح مجموع n است. طولانی ترین مسیر از ریشه به برگ‌ها مسیر $n \to \frac{۲}{۳}n \to \left(\frac{۲}{۳}\right)^{۲} n \to \ldots \to ۱$ می‌باشد و از آنجایی که $\left(\frac{۲}{۳}\right)^{k} n = ۱$ وقتی که $k = \text{Log}_{\frac{۳}{۲}}^{n}$ بنابراین ارتفاع درخت $\text{Log}_{\frac{۳}{۲}}^{n}$ می‌باشد. البته ارتفاع سمت چپ درخت $\text{Log}_{۳}^{n}$ است پس جمع همه سطوح بین $n\text{Log}_{۳}^{n}$ و $n\text{Log}_{\frac{۳}{۲}}^{n}$ است بنابراین با توجه به اینکه لگاریتم‌ها با هر پایه‌ای هم رشد هستند:

$$T(n) = \theta(n\text{Lg}n)$$

**مثال:** فرض کنید $T(n)$ برای مقادیر کوچک n ثابت است. برای هر قسمت $T(n)$ را با نمادهای مجانبی بیان کنید.

a) $T(n) = ۳T\left(\frac{n}{۲}\right) + n \lg n$

b) $T(n) = ۵T\left(\frac{n}{۵}\right) + \frac{n}{\lg n}$

c) $T(n) = ۳T\left(\frac{n}{۳} + ۵\right) + \frac{n}{۲}$

d) $T(n) = ۲T\left(\frac{n}{۲}\right) + \frac{n}{\lg n}$

e) $T(n) = T\left(\frac{n}{۲}\right) + T\left(\frac{n}{۴}\right) + T\left(\frac{n}{۸}\right) + n$

f) $T(n) = T(n-۱) + \frac{۱}{n}$

g) $T(n) = T(n-۱) + \lg n$

h) $T(n) = T(n-۲) + ۲\lg n$

i) $T(n) = \sqrt{n}\, T(\sqrt{n}) + n$

**پاسخ:**

a) $T(n) = \theta(n^{\lg ۳})$

b) مشابه d بررسی کنید

c) $T(n) = \theta(n.\lg n)$

d) $T(n) = \theta(n.\lg\lg n)$

e) $T(n) = \theta(n)$

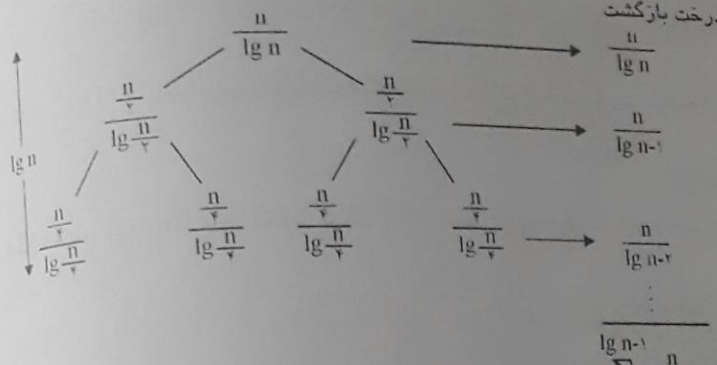
f) $T(n) = \theta(\lg n)$

g) $T(n)=\theta(n\lg n)$

h) $T(n)=\theta(n\lg n)$

i) $T(n)=\theta(n\cdot LgLgn)$

d) اثبات درخت بازگشت




$$\sum_{i=0}^{\lg n-1}\frac{n}{\lg n-i}=\theta(n\lg\lg n)$$

دقت کنید $1+\frac{1}{2}+\ldots+\frac{1}{n}=\theta(\lg n)$ پس $1+\frac{1}{2}+\ldots+\frac{1}{\lg n}=\theta(\lg\lg n)$.

اثبات i) طرفین را به n تقسیم می‌کنیم.

$$\frac{T(n)}{n}=\frac{T(\sqrt{n})}{\sqrt{n}}+1$$

تعریف می‌کنیم $h(n)=\frac{T(n)}{n}$ پس:

$$h(n)=h(\sqrt{n})+1$$

تغییر متغیر $n=2^m$ پس:

$$h(2^m)=h\left(2^{\frac{m}{2}}\right)+1$$

تعریف می‌کنیم $F(m)=h(2^m)$ پس:

$$F(m)=F\left(\frac{m}{2}\right)+1=\theta(\lg m)$$

در نتیجه $h(n)=\theta(\lg\lg n)$ بنابراین:

$$T(n)=\theta(n\lg\lg n)$$

**تمرین:**

۱- تابع هورنر را به زبان c بنویسید.

۲- n متغیر بولی $x_1$ تا $x_n$ دارای $2^n$ حالت هستند (هر متغیر بولی دو حالت T یا F دارد) برنامه‌ای بنویسید که تمام حالات را چاپ کند.

۳- اگر n یک عدد صحیح مثبت باشد، مشخص کنید که آیا n برابر مجموع عوامل مثبت خودش (بجز خودش) است.

۴- تابع $F(x)$ را در نظر بگیرید:

if (x is even) $F = \frac{x}{۲}$;

else $F = F(F(۳x + ۱))$;

ثابت کنید $F(x)$ برای همه اعداد صحیح پایان پذیر است. (راهنمایی: اعداد صحیح به شکل $(۲i + ۱)۲^k - ۱$ را برای استقراء استفاده کنید)

۵- تابع بازگشتی بنویسید که با دریافت یک مجموعه، مجموعه توانی (Powerset) آن را محاسبه کند. مثلاً آنگاه $s=\{a , b\}$ $powerset(s) = \{\{\}, \{a\}, \{b\}, \{a , b\}\}$

۶- می‌خواهیم ببینیم در لیست $< x_۱, x_۲, ...., x_n >$ بعضی اعداد تکراری هستند یا خیر. نشان دهید که می‌توان با الگوریتم مرتبه $\theta(n \log n)$ این عمل را انجام داد.

۷- می‌توان مرتب سازی درجی را به صورت بازگشتی نوشت. به منظور مرتب سازی $A[۱..n]$، به صورت بازگشتی، آرایه $A[۱..n-۱]$ مرتب می‌شود و سپس $A[n]$ درج می‌شود. یک رابطه بازگشتی برای زمان اجرای این نوع مرتب سازی درجی بنویسید.

۸- در مرتب سازی درجی، در حلقه while خطوط ۷-۵ برای درج $A[j]$، جستجوی خطی به سمت عقب در $A[۱..j-۱]$ انجام می‌شود. آیا می‌توان با جستجوی دو دویی، مرتبه الگوریتم را به $\theta(n \log n)$ کاهش داد.

۹- یک الگوریتم مرتبه $\theta(n \log n)$ ارائه دهید که یک عدد $x$ و آرایه $s$ را دریافت کند و مشخص کند آیا هیچ ۲ عنصری از $s$ وجود دارند که جمعشان $x$ شود.

۱۰- فرض کنید مرتب سازی درجی در $۸n^۲$ مرحله، آرایه $n$ عنصری را مرتب می‌کند و مرتب سازی ادغامی در $۶۴n \log n$ مرحله. برای چه مقادیری از $n$، درجی از ادغامی سریعتر است؟

۱۱- کوچکترین مقدار $n$ که به ازای آن الگوریتمی که زمانش $۱۰۰n^۲$ می‌باشد از الگوریتمی که زمانش $۲^n$ می‌باشد سریعتر است را بیابید.

۱۲- اگر چه مرتب‌سازی ادغامی در بدترین حالت $\theta(n \log n)$ و درجی $\theta(n^۲)$ می‌باشد، ولی برای مقادیر کوچک $n$، درجی از ادغامی سریعتر است (تمرین ۱۰).

می‌توان در مرتب‌سازی ادغامی وقتی تقسیم به اندازه کافی انجام شد، برای هر زیر لیست از مرتب‌سازی درجی استفاده کرد. فرض کنید در مرتب سازی ادغامی، $\frac{n}{k}$ زیر لیست به طول $k$ توسط درجی مرتب شوند. و سپس با هم ادغام شوند:

الف) نشان دهید هر یک از $\frac{n}{k}$ زیر لیست به طول $k$، توسط درجی با مرتبه $\theta(nk)$ در بدترین حالت مرتب می‌شوند.

ب) نشان دهید، زیر لیست‌ها در بدترین حالت با مرتبه $\theta\left(nLg(\frac{n}{k})\right)$ ادغام می‌شوند.

ج) با فرض اینکه این نوع مرتب‌سازی ادغامی از مرتبه $\theta\left(nk + nLg(\frac{n}{k})\right)$ می‌باشد، $k$ حداکثر برحسب $n$ از چه مرتبه‌ای باشد $(k = \theta(f(n)))$ که مرتب‌سازی ادغامی توصیف شده، با مرتب‌سازی ادغامی استاندارد، مرتبه یکسان داشته باشند.

۱۳- فرض کنید $A[1..n]$ آرایه‌ای از $n$ عدد متمایز باشد. اگر $i < j$ و $A[i] > A[j]$ در این صورت $(i, j)$ را یک وارونگی (inversion) در $A$ گویند.

الف) ۵ وارونگی در لیست $<2,3,8,6,1>$ را نام ببرید.

ب) چه نوع آرایه‌ای با اعداد $\{1,2,...,n\}$ بیشترین وارونگی را دارد و چند تا وارونگی دارد؟

ج) چه رابطه‌ای بین زمان اجرای مرتب سازی درجی و تعداد وارونگی‌های آرایه ورودی هست؟

د) الگوریتمی از مرتبه $\theta(nLog\,n)$ ارائه دهید که تعداد وارونگی‌های آرایه $n$ عنصری را مشخص کند. (مرتب‌سازی ادغامی را تغییر دهید)

۱۴- با درخت بازگشتی، مرتبه رابطه بازگشتی $T(n) = T(\alpha n) + T((1-\alpha)n) + n$ که $\alpha$ ثابت و $0 < \alpha < 1$ را بیابید.

۱۵- مرتبه روابط بازگشتی زیر را بیابید (و یا کران بالا یا پایین بیابید. فرض کنید به ازای $n \leq 2$، $T(n)$ ثابت است)

a) $T(n) = 4T\left(\frac{n}{4}\right) + \sqrt{n}$

b) $T(n) = T(\sqrt{n}) + 1$

c) $T(n) = 4T(\sqrt{n}) + 1$

d) $T(n) = T(n-1) + n$

۱۶- یک آرایه $n$ عنصری $A$ مفروض است. می‌خواهیم با روش Selection Sort آن را مرتب کنیم به این صورت که کوچکترین عنصر را با $A[1]$ عوض کنیم. دومین کوچکترین را با $A[2]$ عوض کنیم و این عمل را $n-1$ بار انجام دهیم. الگوریتم مربوطه را بنویسید و مرتبه آن را در بهترین و بدترین حالت بیابید.

۱۷- دو عدد باینری $n$ بیتی در آرایه‌های $n$ عنصری $A$ و $B$ ذخیره شده‌اند، شبه کدی برای جمع این دو عدد و تولید آرایه $n+1$ عنصری $C$ بنویسید.

۱۸- یک آرایه $A[1..n]$ شامل اعداد صحیح $0$ تا $n$ بجز یک عدد است. به راحتی می‌توان با کمک آرایه $B[0..n]$، آن عدد را با مرتبه $O(n)$ مشخص کرد. ولی فرض کنیم که ما به عناصر $A$ با یک عمل مشخص دسترسی نداریم بلکه تنها عملی که داریم «دسترسی به بیت $j$ ام عنصر $A[i]$» است (یعنی عناصر $A$ باینری هستند). نشان دهید تنها با همین عمل می‌توان عنصری که نیست را با مرتبه $O(n)$ مشخص کرد.

۱۹- یک آرایه $m \times n$ از اعداد حقیقی را monge گویند اگر:

$$\forall i,j,k,l, 1 \leq i < k \leq m, 1 \leq j < L \leq n : A[i,j] + A[k,L] \leq A[i,L] + A[k,j]$$

به عبارتی هر دو سطر و دو ستون از A را که در نظر بگیریم ۴ عنصر که در گوشه‌ها هستند، طوری باشد که جمع عنصر بالا چپ با عنصر پایین راست کمتر یا مساوی جمع عنصر راست بالا با چپ پایین شود. مثلاً آرایه زیر monge است (مثلاً تقاطع سطر اول و چهارم با ستون دوم و چهارم را در نظر بگیرید:

$$\begin{bmatrix} ۱۰ & ۱۷ & ۱۳ & ۲۸ & ۲۳ \\ ۱۷ & ۲۲ & ۱۶ & ۲۹ & ۲۳ \\ ۲۴ & ۲۸ & ۲۲ & ۳۴ & ۲۴ \\ ۱۱ & ۱۳ & ۶ & ۱۷ & ۷ \\ ۴۵ & ۴۴ & ۳۲ & ۳۷ & ۲۳ \\ ۳۶ & ۳۳ & ۱۹ & ۲۱ & ۶ \\ ۷۵ & ۶۶ & ۵۱ & ۵۳ & ۳۴ \end{bmatrix}$$

$۱۷+۱۷ \le ۱۳+۲۸$

الف) نشان دهید یک آرایه monge است اگر و فقط اگر:

$$\forall i = ۱,۲,\ldots,m-۱ \text{ and } j = ۱,۲,\ldots,n-۱$$
$$: A[i,j]+A[i+۱,j+۱] \le A[i,j+۱]+A[i+۱,j]$$

## پاسخ تمرین:

۸- خیر. بخاطر شیفت عناصر مرتبه الگوریتم بهتر نمی‌شود.

۹- **روش اول:** A را مرتب کنید ($\theta(n\,Lgn)$) و سپس با یک حلقه i از ۱ تا n، $x - A[i]$ را با روش دودویی، جستجو کنید ($\theta(n\,Lgn)$):

```
CHECKSUMS(A,x)
A ← sort(A), n ← length(A)
for i ← ۱ to n do
    if A[i] ≥ ۰ and bsearch (A,۱,n,x − A[i])
        then return true
return false
```

فرض شده است که جستجوی دودویی bsearch، خروجی true یاfalse برمی‌گرداند.

**روش دوم:** الگوریتم دیگری می‌توان به شکل زیر ارائه داد:

۱) S را مرتب کنید.

۲) مجموعه $S'$ را بسازید $S' = \{z : z = x - y \text{ for } y \in s\}$

۳) $S'$ را مرتب کنید.

۴) اگر در S عنصر تکراری هست، فقط یک مورد را نگهدارید و بقیه را حذف کنید. با $S'$ نیز همین طور.

۵) S و $S'$ را ادغام کنید.

۶) اگر حاصل عنصر تکراری داشت پس جواب true در غیر این صورت false است. مراحل ۱ و ۳ از مرتبه $O(n\,Lgn)$ هستند و مراحل ۲ و ۴ و ۵ و ۶ از مرتبه $O(n)$ هستند.

۱۰- $۸n^۲ < ۶۴\,n\,Lgn \Rightarrow \frac{n}{۸} < Lgn \Rightarrow ۲^{\frac{n}{۸}} < n$

بررسی کنید که $۲ \le n \le ۴۳$

۱۱- $۱۰۰n^۲ < ۲^n \Rightarrow n \ge ۱۵ \Rightarrow n_{min} = ۱۵$

۱۲- الف) مرتبه مرتب‌سازی درجی برای مرتب‌سازی لیست k عنصری در بدترین حالت $\theta(k^2)$ است پس مرتبه مرتب‌سازی $\frac{n}{k}$ تا لیست k عنصری برابر $\theta\left(\frac{n}{k}.k^2\right)=\theta(nk)$ است.

ب) اگر بخواهیم همانند ادغام دو لیست، همه لیست‌ها را با هم ادغام کنیم، مرتبه $\theta\left(n.\frac{n}{k}\right)=\theta\left(\frac{n^2}{k}\right)$ است (n بخاطر اینکه هر عنصر یک بار در لیست نهایی کپی می‌شود، $\frac{n}{k}$ چون $\frac{n}{k}$ لیست در هر مرحله بررسی می‌شوند که عنصر بعدی انتخاب و به لیست نهایی کپی شود).

ولی برای رسیدن به مرتبه $\theta\left(n.Lg\left(\frac{n}{k}\right)\right)$، باید لیست‌ها جفت جفت با هم ادغام شوند و سپس لیست‌های حاصل نیز جفت جفت ادغام شوند و به همین ترتیب، تا وقتی که فقط یک لیست بماند.
ادغام جفت جفت نیاز به زمان $\theta(n)$ در هر مرحله دارد چون روی n عنصر عمل می‌کند، با اینکه روی زیر لیست‌ها افراز شده است. تعداد سطوح با شروع از $\frac{n}{k}$ لیست (هر یک k عنصر) و در نهایت ۱ لیست (با n عنصر)، برابر $\left\lceil Lg\left(\frac{n}{k}\right)\right\rceil$ است. پس زمان کل برای ادغام $\theta\left(n\,Lg\left(\frac{n}{k}\right)\right)$ است.

ج) باید $\theta\left(nk+n\,Lg\frac{n}{k}\right)=\theta(n\,Lgn)$ که نتیجه می‌شود k حداکثر $k=\theta(Lgn)$ است.

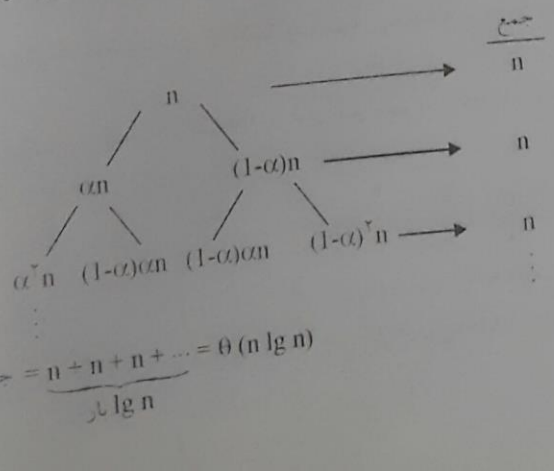
۱۳- الف) وارونگی‌ها عبارتند از (۵ و ۱)، (۵ و ۲)، (۴ و ۳) و (۵ و ۳) و (۵ و ۴) دقت کنید که اندیس عناصر و نه خود عناصر ذکر شده است.

| ۱ | ۲ | ۳ | ۴ | ۵ |
|---|---|---|---|---|
| ۲ | ۳ | ۸ | ۶ | ۱ |

ب) آرایه نزولی مرتب $<n,n-1,\ldots,2,1>$ و تعداد وارونگی‌هایش $\frac{n(n-1)}{2}$ است.

ج) درواقع هر بار اجرای حلقه while در مرتب‌سازی درجی متناظر با کاهش یک وارونگی در آرایه است.

۱۴-



۱۶- فرض کنید $FIND-MIN(A,r,s)$ اندیس کوچکترین عنصر در A از $A[r]$ تا $A[s]$ را برمی‌گرداند. واضح است که می‌توان این الگوریتم را با مرتبه $O(s-r)$، $S \geq r$ پیاده‌سازی کرد. بنابراین الگوریتم مرتب‌سازی انتخابی:

Algorithm SELECTION-SORT(A)
**Input:** $A = < a_1, a_2, ..., a_n >$
**Output:** Sorted A.
**for** $i \leftarrow 1$ **to** $n-1$ **do**
$j \leftarrow$ FIND-MIN (A,i,n)
$A[j] \leftrightarrow A[i]$
**end for**

با توجه به اینکه $n-1$ بار FIND-MIN فراخوانی می‌شود مرتبه بهترین و بدترین برابر است با:

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