
بسم الله الرحمن الرحيم

ساختمان داده ها

دانشگاه صنعتی نوشیروانی بابل

دکتر حسام عمران پور

طراحان اسلاید:
زهرا ریحانیان و دانیال علیزاده

حل تمرین :
علی باقری

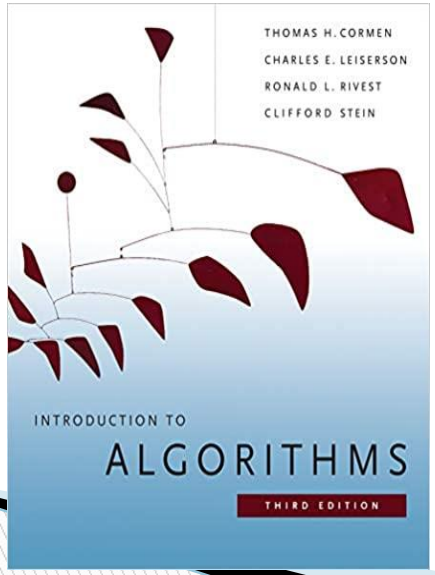
لینک کانال تلگرام اطلاع رسانی و حل تمرین:
t.me/ds_nit_4011

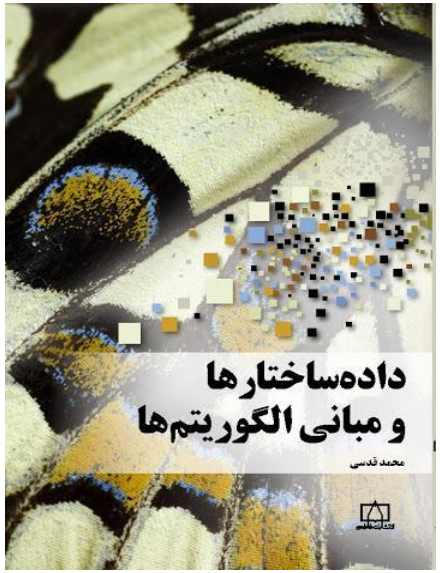
هدف

- هدف از این درس ارائه راهکارهایی است که دانشجو بتواند با استفاده از آنها برنامه بهینه تولید نماید .
- منظور از یک برنامه بهینه برنامه ایست که بتواند هدف خاصی را برآورده کند و در راستای رسیدن به این هدف خاص، سریع ترین راه حل ممکن و کم ترین حافظه ممکن استفاده شود

مراجع

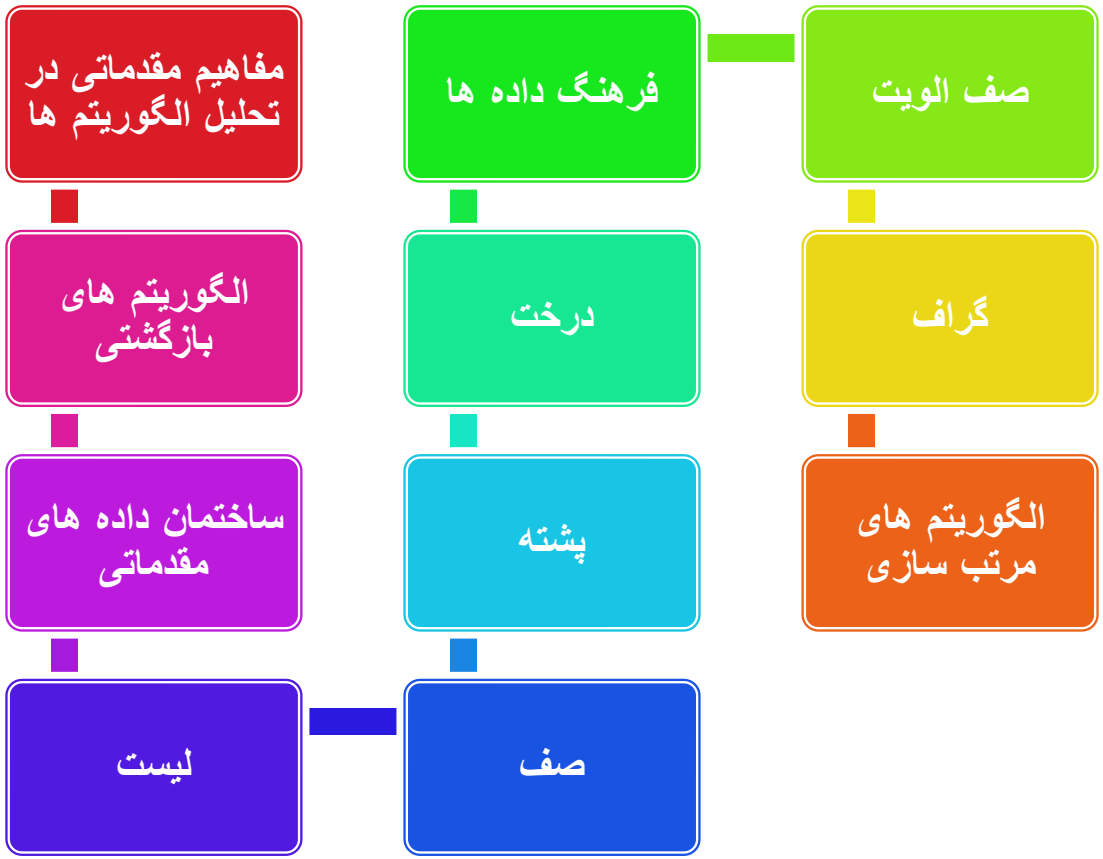
- مرجع اصلی :محسن ابراهیمی مقدم ,ساختمان داده ها و الگوریتمها ,انتشارات نصیر1392,
- محد قدسی، داده ساختارها و الگوریتم ها، انتشارات فاطمی، شهریور 1388
- T.Crmen, C.Leiscrson, and R.Rivest. Introduction to Algorithms. McGraw-Hill Inc., 2001.





بارم بندی

سرفصل های کتاب مرجع



فصل اول
مفاهیم مقدماتی در تحلیل الگوریتم ها

هدف

- هدف از تحلیل الگوریتم بررسی تغییر زمان اجرای یک الگوریتم یا یک برنامه نوشته شده یا به عبارتی دیگر بررسی تغییر سرعت اجرای یک برنامه در مقابل تغییر اندازه ورودی است.

- نتیجه نهایی تحلیل یک الگوریتم تابعی خواهد بود به نام T(n)که سعی می کند زمان اجرای یک الگوریتم را بر اساس nکه nاندازه ورودی یا تعداد داده های وارد شده به یک الگوریتم است تخمین بزند.

عوامل مهم در سرعت اجرای الگوریتم

- 1نوع سخت افزار

عوامل مهم در سرعت اجرای الگوریتم

2-نوع برنامه نویسی

نوشتن یک برنامه با سه حلقه تو در تو

```
{
    int n = 10;
    int arr[10] = {1, 2, 31, -3, 51, 12, -102, 1, 31, 1};
    int maxSum = INT_MIN;

    for(int i = 0; i < n; i++) {
        for(int j = i; j < n; j++) {
            int sum = 0;
            for(int k = i; k <= j; k++)
                sum += arr[k];
            if(sum > maxSum)
                maxSum = sum;
        }
    }
    cout << maxSum;
}
```

نوشتن همان برنامه با یک حلقه

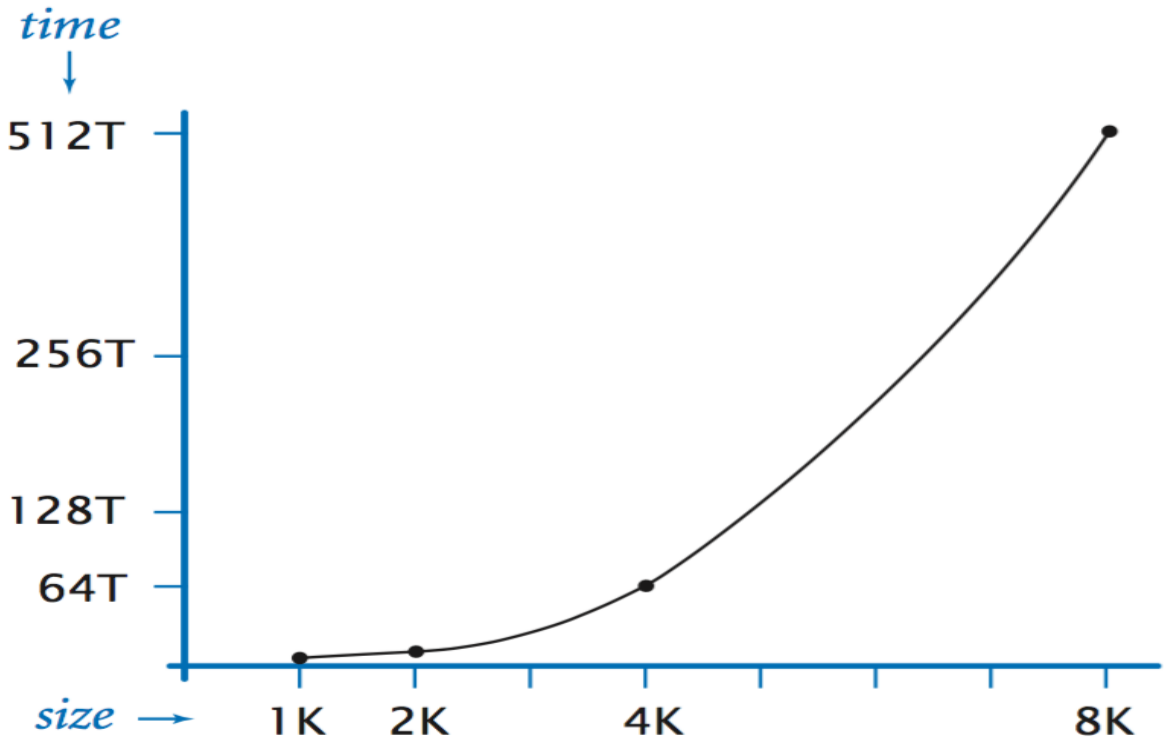
```
{
    int n = 10;
    int arr[10] = {1, 2, 31, -3, 51, 12, -102, 1, 31, 1};
    int maxSum = INT_MIN;

    int sum = 0;
    for(int i = 0; i < n; i++) {
        if(sum < 0)
            sum = 0;
        sum += arr[i];
        if(sum > maxSum)
            maxSum = sum;
    }
    cout << maxSum;
}
```

عوامل مهم در سرعت اجرای الگوریتم

-3زبان برنامه نویسی

عوامل مهم در سرعت اجرای الگوریتم

4- اندازه ورودی



عوامل مهم در سرعت اجرای الگوریتم

تابعی که آرایه Aبه اندازه nرا به عنوان آرگومان دریافت نماید و مشخص کند عدد صفر در آرایه هست یا نیست؟

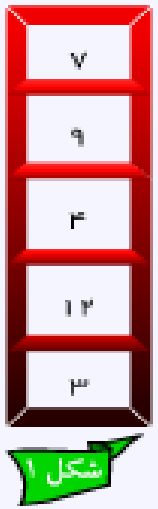
```
for (i = 0 ; i < n ; i++)
    if A[i] == 0
        break;

if (i == n)
    cout << “no”
else
    cout << “yes”
```

عوامل مهم در سرعت اجرای الگوریتم

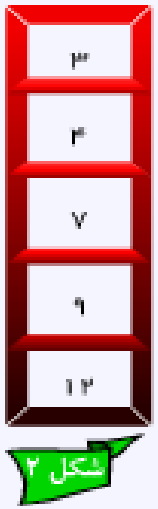
5-ترکیب ورودی

- منظور از ترکیب ورودی نحوه چیدمان عناصر در کنار یکدیگر و نوع داده های ورودی به یک الگوریتم است.

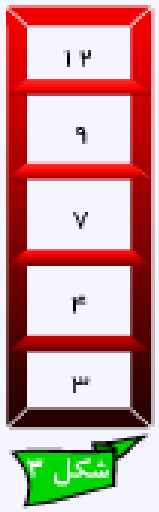
Average Case



Best Case



Worst Case



عوامل مهم در سرعت اجرای الگوریتم

بنابراین با توجه به این 5عامل میتوانیم بیان کنیم که در تحلیل الگوریتم ها ما تنها به گزینه های چهارم و پنجم در سرعت اجرای الگوریتم ها دقت میکنیم به عبارت دیگر معمولا فرض می کنیم که نوع سخت افزار ثابت است، برنامه نویسی به بهترین نحوه ممکن انجام می شود و خطاهای متداول برنامه نویسی در الگوریتم ها وجود ندارد و زبان برنامه نویسی نیز از یک زبان شبه کد استفاده می کنیم.

بنابراین عواملی که در درس ساختمان داده ها و برای تحلیل الگوریتم ها مورد دقت قرار میگیرد دو عامل اندازه ورودی و ترکیب ورودی است

مثال :برنامه های زیر را از لحاظ زمان مقایسه کنید.

```
for (i = 1; i <= 10; i++)
        cout << 2 * i
```

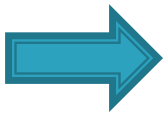
دستورالعمل	تکرار
i = 1	1
i <= 10	11
i++	10
cout	10

در مجموع ⇐ 32

```
for (i = 1; i <= 20; i++)
    if i % 2 == 0
        cout << i
```

دستورالعمل	تکرار
i = 1	1
i <= 20	21
i++	20
if	20
cout	10

در مجموع



72

```
for (i = 2; i <= 20; i+=2)
    cout << i
```

دستورالعمل	تکرار
i = 2	1
i <= 20	11
i+=2	10
cout	10

در مجموع ⇐ 32

مثال :در شبه کد های زیر تعداد تکرار some operation را برحسب n بیابید.

```
for i=1 to n
      s.o
```

→ **n**

```
for i=3 to n
      s.o
```

→ **n – 3 + 1**

```
for i=a to b
      s.o
```

→ **b– a+ 1**

```
for i=a downto b
      s.o
```

→ **a – b + 1**

```
for i=1 to n
    for j=1 to n
        s.o
```

n^2

```
for i=4 to n
{
    sum = L[i-3]
    for j=i-3 to I
    {
        sum += L[j+n]
        s.o
     }
}
```

(n-3) * 4

```
i=1
while  i<=n
{
      i*=2
      s.o
}
```

تکرار	i
1	
1	
1	
1	
…	…
1	

زمان اجرای
s.o

$\log_2 n$

```
while n >= 1
{
        n=[n/2]
        s.o
}
```

$\log_2 n$

```
i = n
while i > 1
{
        j = 1
        while j < n
        {
            j = j * 3
            s.o
         }
         i = i / 2
}
```

$\log_2 n * \log_3 n$

```
for i=1 to n
    for j=1 to i
        s.o
```

i	j	تکرار s.o
1	1 => 1	1
2	1 => 2	2
3	1 => 3	3
…	…	…
n	1 => n	n

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

```
for i=1 to n
{
    for j=1 to n
        s.o
    n = n/2
}
```

$$n + n/2 + n/4 + n/8 + \ldots + n/2^k$$

$$n = 2^k \Rightarrow k = \log_2 n$$

$$= n(1 + 1/2 + 1/4 + 1/8 + \ldots + 1/2^k)$$

$$= n(1/2^0 + 1/2^1 + 1/2^2 + 1/2^3 + \ldots + 1/2^{\log_2 n})$$

$$n * \left(\frac{1 - (\frac{1}{2})^{\log_2 n+1}}{1 - 1/2}\right)$$

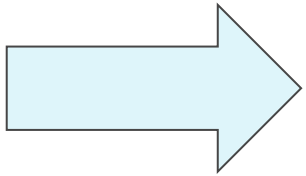
با توجه به رابطه سری هندسی داریم :

و در نهایت جواب مساله :

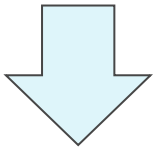
2n - 1

```
i = n
while  i > 1
{
      j = 1
      while  j < n
      {
            j = j * 3
            i = i / 2
            s.o
      }
}
```

```
i = n
while  i > 1
{
      j = 1
      while  j < n
      {
            j = j * 3
            i = i / 2
            s.o
      }
}
```



```
i = n
while  i > 1
{
      i = i / x
      s.o
}
```



```
i = n
while  i > 1
{
      i = i / ( 2 ^ log(n, 3) )
      s.o
}
```

در هر نوبت از تکرار حلقه بیرونی ، حلقه درونی $\log_3 n$ بار اجرا می شود. اما نکته اصلی مساله در مقدار i است، زیرا مقدار i در حلقه درونی دچار تغییر می شود. می توان این مساله را ساده کد که که مقدار i در هر نوبت از اجرای بدنه حلقه بیرونی، تقسیم بر $2^{\log_3 n}$ می شود، تا زمانی که $i = 1$ گردد، بنابراین می توان گفت حلقه بیرونی $\log_{(2^{\log_3 n})} n$ مرتبه تکرار می شود. پس تعداد کل دفعات تکرار دستور S.O برابر خواهد بود با :

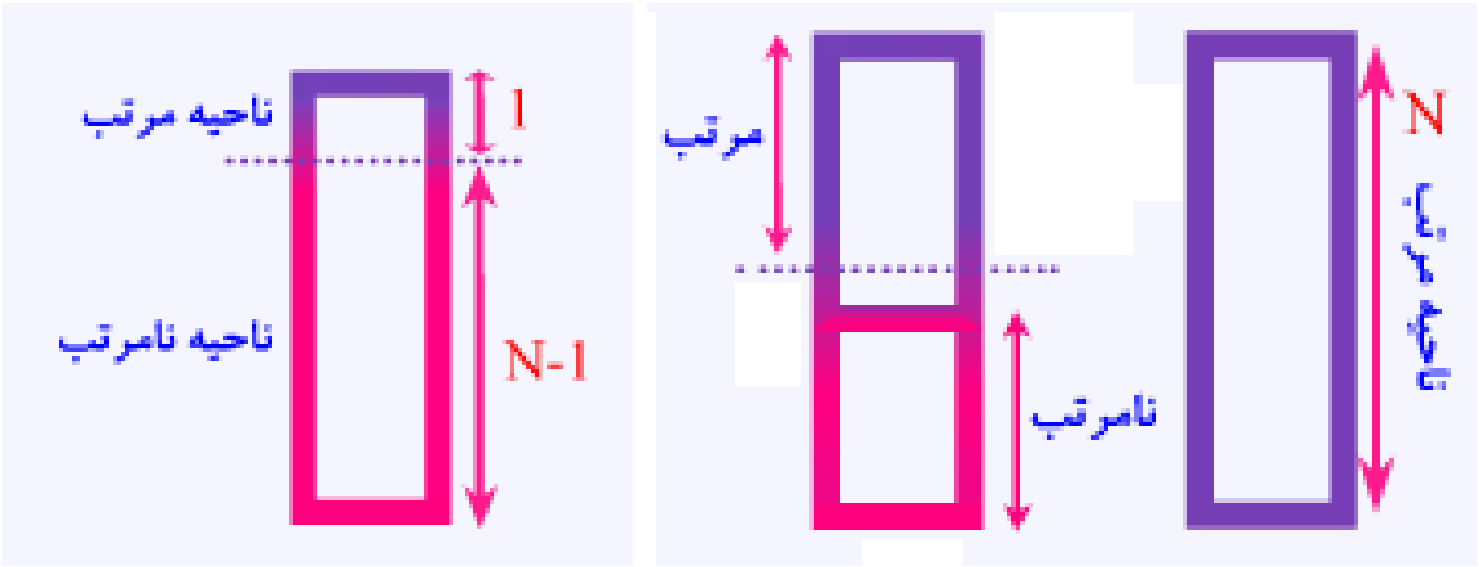
$$(\log_{(2^{\log_3 n})} n) * \log_3 n = \frac{1}{\log_3 n} * \log_2 n * \log_3 n = \log_2 n = \log n$$

هدف از تحلیل الگوریتم

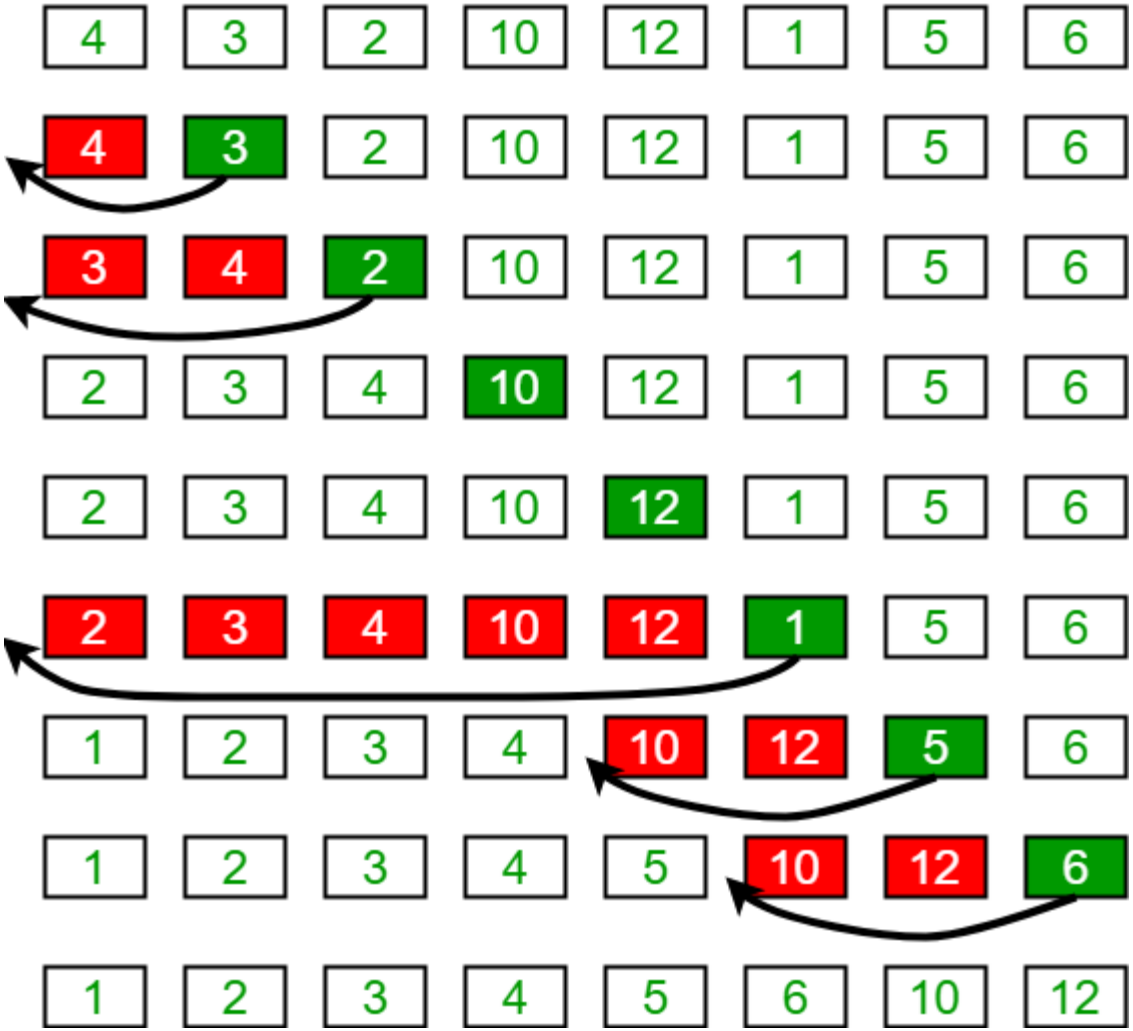
هدف از تحلیل رسیدن به سه تابع زیر است :

تحلیل الگوریتم مرتب سازی درجی

- داده ها را به دو ناحیه مرتب و نامرتب تقسیم میکند
- برنامه و تحلیل سه حالت بهترین بدترین و میانگین:



Insertion Sort Execution Example



شبه کد مرتب سازی درجی

```
function isertionSort(L: Array; n: integer){
    // L is Array of elements
    // n is number of elements
    var i: integer
    for j = 2 to n do                    1
    {
        Current = L[j];                  2
        i = j-1                          3
        while i > 0 and L[i] > Current do    4
        {
            L[i + 1] = L[i];             5
            i = i - 1;                   6
        }
        L[i + 1] = Current;              7
    }
}
```

تعداد تکرار کل دستورالعمل ها برای آرگومان n

دستورالعمل	تکرار
1	n
2	n-1
3	n-1
4	
5	
6	
7	n-1

تحلیل الگوریتم مرتب سازی درجی در حالت های مختلف

	تعداد کل دستورالعمل ها	
بهترین حالت	5n – 4	1
بدترین حالت		j
حالت متوسط		

Analysis of Insertion Sort

	Cost	Times
• ***For*** j ← 2 ***to*** n ***do***	c1	n
• ***key*** ← ***A[j]***	c2	n-1
• Insert A[j] into sorted sequence A[1…j-1]		
• ***i*** ← ***j-1***	c3	n-1
• ***while i >0 and A[i] >key***	c4	$\sum_{j=2}^{n} t_j$
• ***do A[i+1]*** ← ***A[i]***	c5	$\sum_{j=2}^{n} t_j-1$
• ***i- -***	c6	$\sum_{j=2}^{n} t_j-1$
• ***A[i+1]*** ← ***key***	c7	n-1

Total Time = $n(c1 + c2 + c3 + c7) + \sum_{j=2}^{n} t_j\ (c4 + c5 + c6) - (c2 + c3 + c5 + c6 + c7)$

t_j counts the number of times the values have to be shifted in one iteration

نرخ رشد

به جمله ای که در رابطه چندجمله ای بیشترین نقش را در رشد آن داشته باشد، نرخ رشد آن چندجمله ای گفته میشود.

مثال:

$$t(n) = n^3 + 10n$$
$$f(n) = n^4 + 10000000n + 10^{100}$$
$$g(n) = n^2 + \log n \times n^2 + n$$
$$h(n) = n^3 + \log n \times n^2 + \log n + 10$$

ضریب ثابت در نرخ رشد تاثیری ندارد.

$h(n) = 3n^2$

$\hookrightarrow h(n) \equiv g(n)$

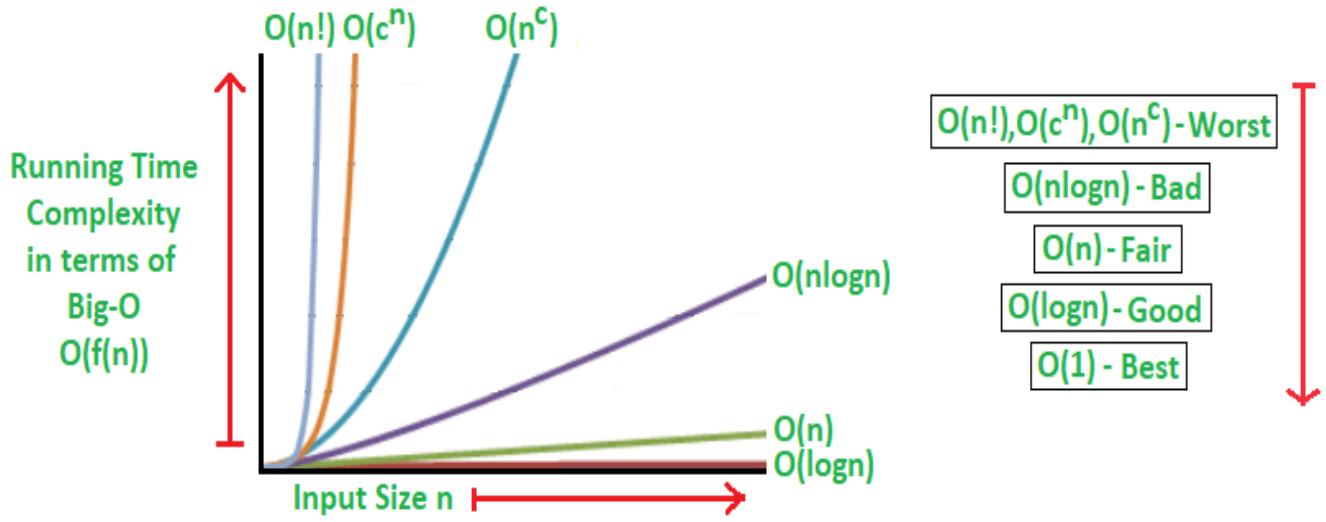
$g(n) = 10000n^2$

پایه لگاریتم در نرخ رشد تاثیری ندارد.

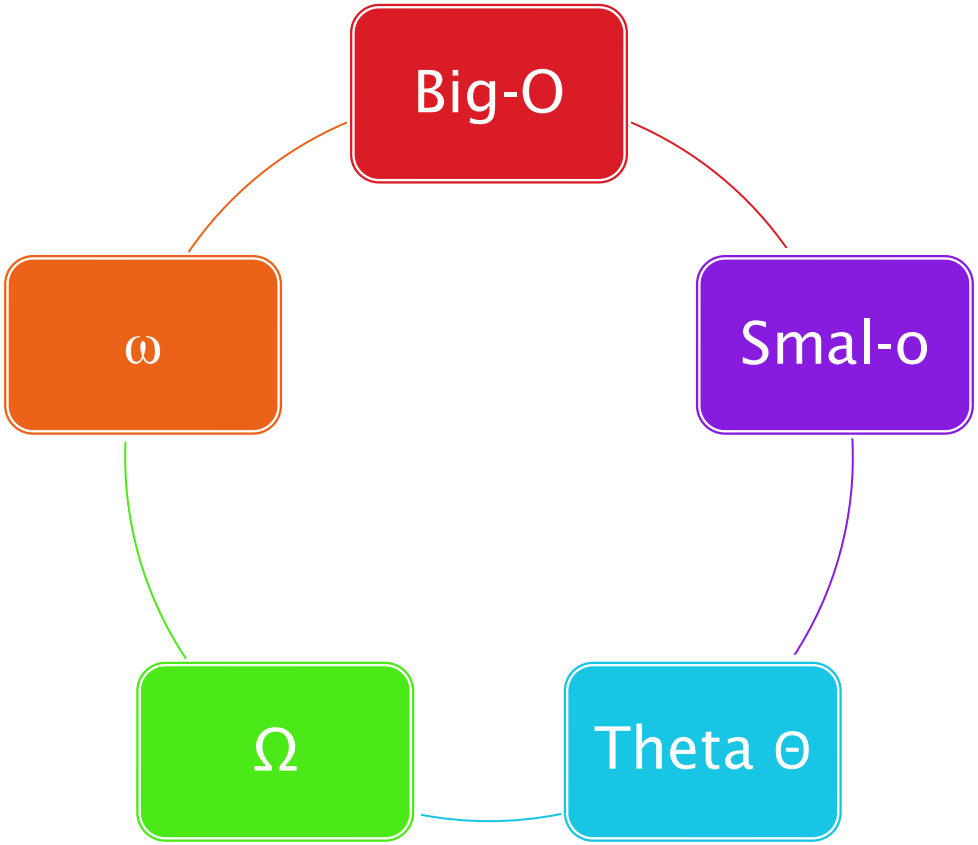
$\log_{10} n = \log_2 n = \log_b n = \log_c n$

به عنوان تمرین ثابت کنید.

جدول نرخ رشد برخی توابع معروف



نماد های مجانبی



big-O

نماد O یا big O نشان دهنده مرتبه یک الگوریتم است.

مرتبه یک الگوریتم : مرتبه یک الگوریتم از نظر ما چند جمله ای حاصل از تحلیل آن الگوریتم می باشد .بنابراین اگر بخواهیم حاصل تحلیل الگوریتم Insertion Sort را بر اساس نماد O بازنویسی کنیم، خواهیم داشت :

$$T_B(n) = O(n)$$

که این یعنی مرتبی الگوریتم Insertion Sort در بهترین حالت از مرتبه n خواهد بود.

big-O

با توجه به این توضیحات ، مرتبه الگوریتم Insertion Sort در بدترین حالت یا همان $T_W(n)$ برابر است با $O(n^2)$

حاصل نهایی تحلیل هر الگوریتم ، مرتبه آن الگوریتم است که برای ما اهمیت دارد و مرتبه الگوریتم در دید ساده برابر با جمله با بالاترین توان و در دید کلی، جمله با بیشترین نرخ رشد می باشد.

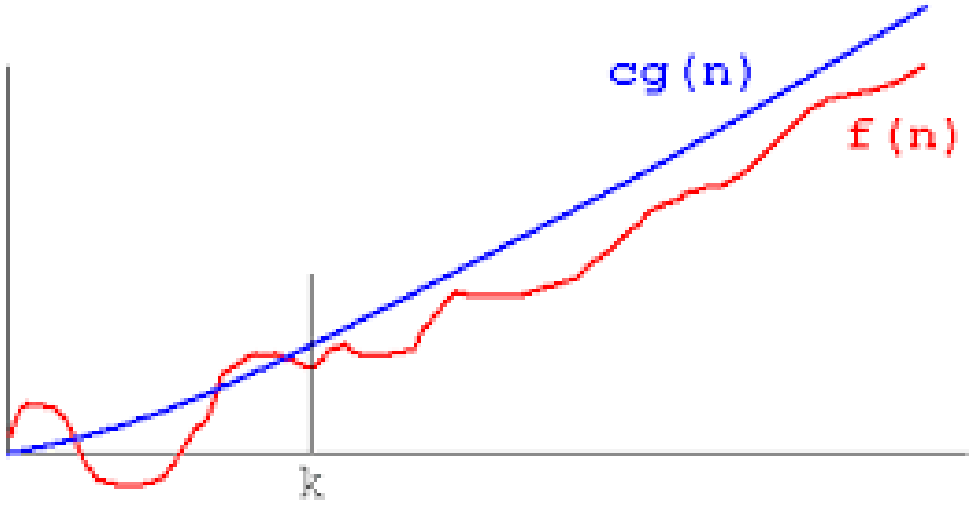
big-O

تعریف ریاضی

$$f(n) \in O(g(n)) \leftrightarrow \{\exists c > 0,\ n_0 > 0 \mid \forall n \geq n_0; f(n) \leq c.g(n)\}$$

big-O

به عبارت دیگر وقتی می گوییم f(n)، O(g(n)) است یا f(n) از مرتبه g(n) است به این معنی است که رشد تابع f(n) در حالت عادی از رشد تابع g(n) همواره کمتر است .به عبارت دیگر حالت حدی و نقاط اولیه در این تعریف مدنظر نیستند.

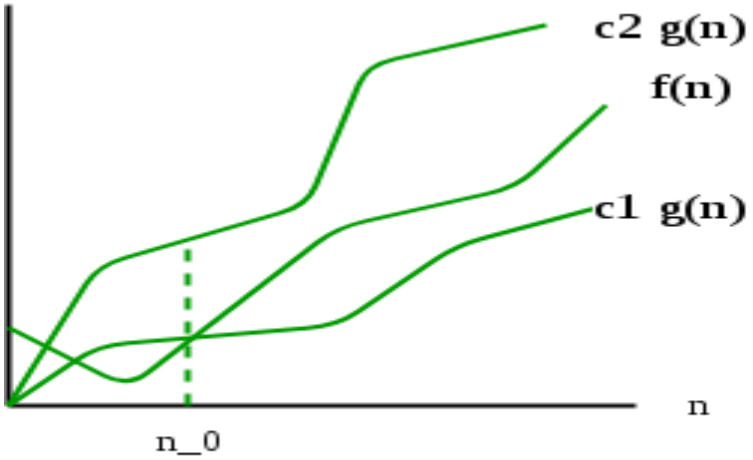


Theta Θ

این نماد وقتی به کار میرود که نرخ رشد دو تابع مساوی باشد.

تعریف ریاضی

$$f(n) \in \theta(g(n)) \leftrightarrow \{\exists c_1,\ c_2, n_0 > 0 \mid \forall n > n_0 ; c_1 . g(n) \leq f(n) \leq c_2 . g(n)\}$$



f(n) = theta(g(n))

Theta Θ

سوال :

اگر داشته باشیم : $f(n) = \theta(n^2)$ آیا میتوان از این نتیجه گرفت که : $f(n) = \theta(n^3)$ برای پاسخ به این سوال سه گزینه پیشنهاد می شود:

1- نمی توان نتیجه گرفت.
2- لزوما نمیتوان نتیجه گرفت.
3- میتوان نتیجه گرفت.

باید دقت کرد که هیچ گاه نمی توان چنین نتیجه ای را گرفت. به عبارت دیگر وقتی میگوییم: $f(n) = \theta(n^2)$ یعنی رشد تابع $f(n)$ در بینهایت n^2 است. بنابراین هیچگاه نمی تواند n^3 باشد

big-Omega(Ω)

بصورت خلاصه میتوان بعنوان معکوس نماد Oدر نظر گرفت.

تعریف ریاضی

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

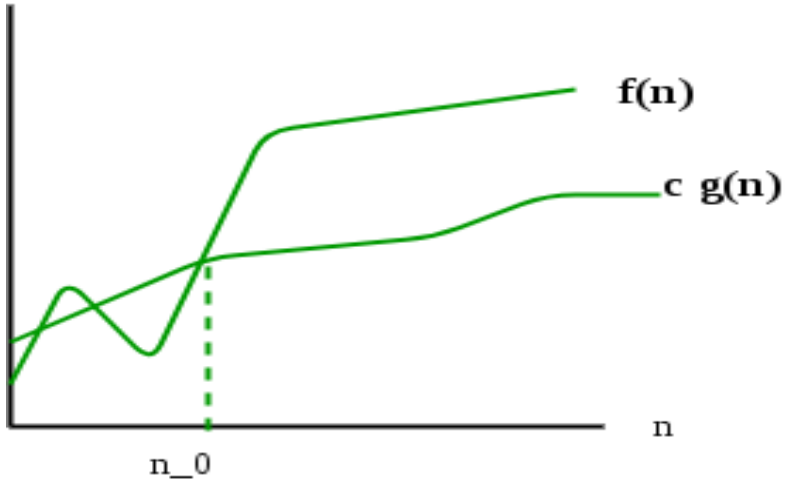
به عبارتی دیگر میتوانیم بگوییم اگر $f(n) = \Omega(g(n))$ است، آنگاه رشد تابع f(n) همواره بیشتر از رشد تابع g(n) است، عبارتی دیگر تابع g(n) حد پایین رشد تابع f(n) را نمایش میدهد.

big-Omega(Ω)

نماد Ω در الگوریتم ها در هنگامی کاربرد دارد که میخواهیم بهترین حاصل یک الگوریتم را بررسی کنیم.

بعنوان مثال بیان میشود که الگوریتم های مرتب سازی مقایسه ای همواره از مرتبه $\Omega(n \log n)$ هستند.

به این معنی است که هیچگاه الگوریتم مرتب سازی مقایسه ای نمیتوان پیدا کرد که تابع تحلیل آن مقداری کمتر از $n \log n$ داشته باشد یا رشد آن از $n \log n$ کمتر باشد.



f(n) = Omega(g(n))

small-o

تعریف ریاضی

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

در تعریف مطرح شده برای نماد small o دقت کنید، دقت در این تعریف نشان می دهد که این تعریف نسبت به Big O در دو قسمت متفاوت است :

small-o

تفاوت ها :

۱. در قسمت تعریف یا Declaration مربوط به Constant c .

۲. در نماد مطرح شده برای رشد توابع، به عبارت دیگر وقتی $f(n) = o(g(n))$ ، آنگاه رشد آن همواره از g(n) کمتر است و هیچگاه به آن نخواهد رسید.

باتوجه به این نکته و باتوجه به نکاتی که در مورد Big O مطرح شد، میتوان این نتیجه را گرفت :

اگر داشته باشیم $f(n) = o(g(n))$ است ، آنگاه :

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

وقتی که n به سمت بینهایت میل میکند برابر خواهد بود با صفر. به عبارت دیگر بدلیل اینکه رشد تابع g(n) همواره بزرگتر از f(n) است ، این رابطه حدی قابل استنتاج است.

small-o

نتیجه دیگری که باتوجه به تعریف Small o میتوان گرفت این است که :

اگر $f(n) = o(g(n))$ آنگاه میتوان نتیجه گرفت $Lim\ \frac{g(n)}{f(n)}$ وقتی n به سمت بینهایت میل میکند برابر است با بینهایت. نتیجه معکوس نیز از این حد قابل استنتاج است.

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

small-omega(ω)

همانطور که دیدیم نماد Ω برای نشان دادن حد پایین رشد یک تابع استفاده میشود.

نماد ω نیز با تعریفی مشابه به صورت زیر تعریف میشود :

تعریف ریاضی

$$f(n) = \omega\big(g(n)\big) \Leftrightarrow \{\exists\, n_0 > 0\,, \forall\, c > 0\,, \forall\, n > n_0\,, f(n) > cg(n)\}$$

خلاصه

با توجه به مطالب ذکر شده سعی میکنیم در اینجا خلاصه ای از رفتار نمادها و نتایج حاصل از آنها را بر روی دو تابع بررسی کنیم.

اگر فرض کینم توابع f(n) و g(n) موجود باشند که رشد تابع f(n) برابر a و رشد تابع g(n) برابر b است ، آنگاه حالت های زیر قابل استنتاج است :

$$a \geq b \Rightarrow f(n) = \Omega\big(g(n)\big), g(n) = \mathrm{O}\big(f(n)\big)$$

$$a > b \Rightarrow f(n) = \omega\big(g(n)\big), g(n) = o\big(f(n)\big)$$

$$a = b \Rightarrow f(n) = \theta\big(g(n)\big), g(n) = \theta\big(f(n)\big)$$

$$a \leq b \Rightarrow f(n) = \mathrm{O}\big(g(n)\big), g(n) = \Omega\big(f(n)\big)$$

$$a < b \Rightarrow f(n) = o\big(g(n)\big), g(n) = \omega\big(f(n)\big)$$

?? چند مثال

موارد زیر را با توجه به $f_n = 3n^2 + n + 10$ ثابت کنید.

الف) $f_n \in O(n^2)$

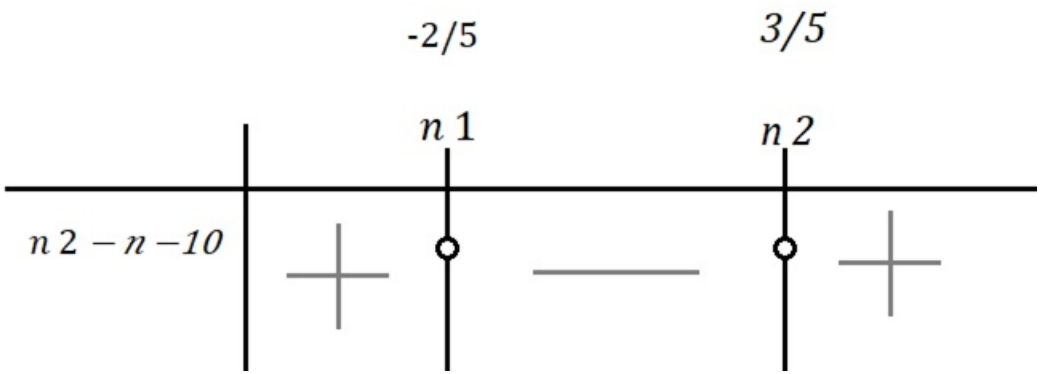
جواب :

$3n^2 + n + 10 \leq cn^2$; دلخواه $c = 4$

$\Rightarrow 4n^2 - 3n^2 - n - 10 \geq 0$

$\rightarrow n^2 - n - 10 \geq 0$

$$n_1 , n_2 = \frac{1 \pm \sqrt{41}}{2}$$



$n_0 = 4$

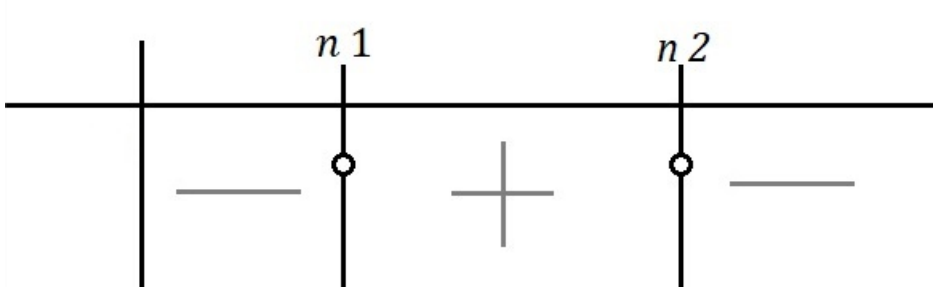
ب) $f_n \notin O(n)$

جواب :

$$3n^2 + n + 10 \leq cn$$
$$\rightarrow -3n^2 + (c-1)n - 10 \geq 0$$

$$\Delta = (c-1)^2 - 120$$

$$n_1 , n_2 = \frac{1 - c \pm \sqrt{\Delta}}{-6}$$



پس $f_n \notin O(n)$.

ثابت کنید اگر $f_n = n + 10$ باشد، آنگاه $f_n \in o(n^2)$.

جواب :

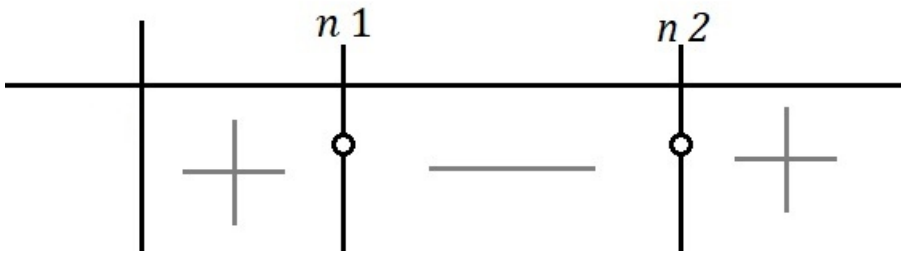
برای هر c باید یک n_0 پیدا شود

$$n + 10 < cn^2$$

$$\rightarrow cn^2 - n - 10 > 0$$

$$\Delta = 1 + 40c$$

$$n_1 , n_2 = \frac{1 \pm \sqrt{\Delta}}{2c}$$



$$n_0 = \lceil n_2 \rceil = \left\lceil \frac{1 + \sqrt{\Delta}}{2c} \right\rceil$$

پایان فصل اول