الگوریتم عبارت است از روشی برای حل یک مسئله ،اجزای یک الگوریتم باید دارای ویژه گیهای زیر باشد.

1-واضح و غیر مبهم باشد. 2-خاتمه پذیر باشد. 3-ترتیبی باشد.

در درس طراحی الگوریتم بطور عام به بررسی زمان اجرای یک الگوریتم و فضای اشغال شده توسط الگوریتم پرداخته میشود،زیرا برای حل یک مسئله ممکن است روشهای متفاوتی وجود داشته باشد و این وظیفه تحلیل گر مسئله است که بهینه ترین روش حل مسئله را انتخاب کند.

مثال:

برای ضرب دو عدد **n** و **m** چهار روش وجود دارد.

1-روش انگلیسی (روش معمول) 2-روش امریکایی(معکوس روش معمول) 3-روش تقسیم و حل (نصف کردن دو عدد و ضرب تکه اعداد بدست آمده در یکدیگر) 4-روش رسی russi

12*5=60

Russi

m	n
12	5
6	10
3	20
1	40

/2 *2

n هایی باهم جمع میشوند که m های آنها فرد باشند

20+40=60

m=1234 n=105

$m=\begin{cases} m_1=34 \\ m_2=12 \end{cases}$ $n=\begin{cases} n_1=05 \\ n_2=01 \end{cases}$

$m_2 m_1 \times n_2 n_1$

$m_2 n_1 \quad m_1 n_1$

$m_2 n_2 \quad m_1 n_2$

$m_2 n_2 \quad (m_2 n_1 + n_2 m_1) \quad m_1 n_1$

$=m_1 n_2 + (m_2 n_1 + n_2 m_1)*10 + (m_2 n_2)*100$

آنالیز الگوریتمهای مقدماتی:

1) اصل پایانی:

اگر برای یک الگوریتم دو پیاده سازی متفاوت وجود داشته باشد که هر کدام به ترتیب در زمانهای t_1 و t_2 اجراء شوند آنگاه:

$t_1 = \theta(t_2)$ $t_2 = \theta(t_1)$

2) اصل ترتیب:

اگر دو قسمت از بر نامه به نامهای p_1 و p_2 که مستقل از یکدیگر عمل میکنند و بترتیب دارای زمانهای t_1 و t_2 هستند،بصورت متوالی اجراء شوند زمان اجرای کل برنامه برابر $t_1 + t_2$ است.

3) اصل دستورات مقدماتی:

دستوری تجزیه ناپذیر (اتمیک) است که به دستورات کوچکتر تقسیم نشود و در زمان t اجرا شود،دستور مقدماتی دستوری است که از دستورات اتمیک ایجاد شود. بعنوان مثال چهار عمل اصلی ،دستورات شرطی،دستور انتساب (=) و.....

آنالیز حلقه While

تعداد دفعات اجرای قطعه کد زیر بهمراه مرتبه زمانی آن را مشخص کنید؟

1)
```
K=0;
Cin>>i;
While (i>0)
{
K++;
i/=2;
}
Cout<<k;
```

i=8: K=1 i=4; K=2 i=2; K=3 i=1; K=4 i=0

i=7: K=1 i=7; K=2 i=3; K=3 i=1; K=4 i=0

i=16: K=1 i=8; K=2 i=4; K=3 i=2; K=4 i=1; K=4 i=0

/= ⟶ Log n

تعداد دفعات اجراء: $\lfloor \text{Log}_2 n \rfloor + 1$

مرتبه زمانی (در بینهایت): $\text{Log}_2 n$

2)
```
K=0;
Cin>>i;
While (i>0)
{
K++;
i/=5;
}
Cout<<k;
```

تعداد دفعات اجراء: $\lfloor \text{Log}_5 n \rfloor + 1$

مرتبه زمانی (در بینهایت): $\text{Log}_5 n$

3)
```
K=0;
Cin>>i;
While (i>1)
{
K++;
i/=2;
}
Cout<<k;
```

تعداد دفعات اجراء: $\lfloor \text{Log}_2 n \rfloor$

مرتبه زمانی (در بینهایت): $\text{Log}_2 n$

4)
```
n=0;
Cin>>i;
While (i>1)
{
cin>>j;
While (j>0)
   {
    j/=3;
    n++;
   }
i/=2;
n++;
}
```

$\lfloor \text{Log}_3 j \rfloor + 1$; $\lfloor \text{Log}_2 i \rfloor$ ⟶

تعداد دفعات اجراء: $\lfloor \text{Log}_2 i \rfloor (\lfloor \text{Log}_3 j \rfloor + 1) = \lfloor \text{Log}_2 i \rfloor \lfloor \text{Log}_3 j \rfloor + \lfloor \text{Log}_2 i \rfloor$

مرتبه زمانی (Order): $\text{Log}_2 i \cdot \text{Log}_3 j$

آنالیز حلقه For:

for (i=n;i<=m;i++) $\xrightarrow[i++=1]{\text{تعداد دفعات اجراء:}}$ $\frac{m-n+1}{1}$ for (i=n;i<m;i++) $\xrightarrow{\text{تعداد دفعات اجراء:}}$ $\frac{m-n}{1}$

1)
```
for (i=0;i<=5;i++)
{
K++;
}
Cout <<k;
```
تعداد دفعات اجراء: $\frac{5-0+1}{1}=6$

2)
```
for (i=0;i<=n;i+=2)
{
K++;
}
Cout <<k;
```
تعداد دفعات اجراء: $\frac{n-0+1}{2}=\frac{n+1}{2}$

مرتبه زمانی (Order): n

3)
```
k=0;
for (i=1;i<=n^2;i++)
{
Cin>>m;
x=fac(m);
y+=x;
}
Cout <<y;
```
$\frac{n^2-1+1}{1}=n^2$

```
fac(m)=
P=1;
for(j=1;j<=m;j++)
{
P*=j;
}
```
$\frac{m-1+1}{1}=m$

تعداد دفعات اجراء: $n^2 m$

تعداد دفعات اجراء ومرتبه زمانی یکی است چون m نیز پارامتر متغییر است.

$$y=\sum_{i=1}^{n^2} fac(m)$$

حلقه های for مستقل:

1)
```
k=0;
for (i=1;i<=n ;i++)
{
k++;
}
for (j=1;i<=m j++)
{
k*=2;
}
cout <<k;
```
(n) (m)

تعداد دفعات اجراء: n+m

مرتبه زمانی (Order): n+m

در صورت خیلی بزرگتر بودن هر کدام از پارامترها آن پارامتر بعنوان مرتبه زمانی در نظر گرفته میشود.

$k=2^{m}n$

حلقه های تو در تو for مستقل:

1)
```
k=1;
for (i=1;i<=n ;i+=2)        → n/2
{
  for (j=1;i<=n/2 ;j++)     → n/2
  {
   k*=2;
  }
}
cout <<k;
```
تعداد دفعات اجراء: $\frac{n}{2}\times\frac{n}{2}=\frac{n^2}{4}$

مرتبه زمانی (Order): n^2

فرض $n=4$, $\frac{n}{2}=2$

i=1 K=4

i=3 $K=16=(2^{\frac{n}{2}})^{\frac{n}{2}}$

$K=(2)^{\frac{n^2}{4}}$

$$\sum_{i=1}^{n/2} 2^{\frac{n}{2}}=(2)^{\frac{n^2}{4}}$$

2)
```
x=0;
for (i=n^2 ;i>=1 ;i-=2)   → n^2/2
{
  for (j=1;i<=m; j*=2)    → فرض m=8, J=1,2,4,8, x=1,2,3,4
  {
   x++;
  }
}
cout <<x;
```
تعداد دفعات اجراء: $\frac{n^2}{2}$

تعداد دفعات اجراء: $\lfloor \text{Log}_2 m \rfloor + 1$

مرتبه زمانی (Order): $n^2 \text{Log}_2 m$

$X=\frac{n^2}{2}(\lfloor \text{Log}_2 m \rfloor + 1)$

تعداد دفعات اجراء: $\frac{n^2}{2}(\lfloor \text{Log}_2 m \rfloor + 1)$

حلقه های تو در تو for وابسته:

1)

```
for (i=1  ;i<=n ;i++)
{
  for (j=i;i<=n; j++)
  {
   cout <<"*";
  }
}
```

تعداد دفعات اجراء: n

تعداد دفعات اجراء: n-i+1

فرض n=3

i	*		
1	3	n	j:1→n
2	2	n-1	j:2→n
3	1	n-2	j:3→n

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

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

تعداد دفعات اجراء(ستاره ها)

مرتبه زمانی (Order): n^2

2)

```
for (i=1  ;i<=n ;i++)
{
  for (j=i;i<=n; j++)
  {
   for (k=1;k<=m;k++)
     {
      cout <<"*";
     }
  }
}
```

وابسته تو در تو $\frac{n(n+1)}{2}$

مستقل تو در تو m

مرتبه زمانی (Order): $m\times\sum_{i=1}^{n}(n-i+1) = m\times\frac{n(n+1)}{2}$

3)

```
for (i=1  ;i<=n ;i++)
{
  for (j=i;i<=n; j++)
  {
   for (k=j;k<=n;k++)
     {
      cout <<"*";
     }
  }
}
```

مرتبه زمانی (Order): $\sum_{i=1}^{n}\sum_{j=i}^{n}(n-i+1)$

نمادهای مرتبه زمانی:

بطور کلی 5 نماد برای بیان مرتبه زمانی یک الگوریتم وجود دارد.(نمادها از راست به چپ خوانده میشود.)

نماد Big O (>=):

اگر داشته باشیم $f(n)=O\,g(n)$ آنگاه رشته تابع g(n) بیشتر مساوی رشد تابع f(n) است.

نماد Big Ω (<=):

اگر داشته باشیم $f(n)=\Omega\,g(n)$ آنگاه رشته تابع g(n) کمتر مساوی رشد تابع f(n) است.

نماد Θ (=):

اگر داشته باشیم $f(n)=\Theta\,g(n)$ آنگاه رشته تابع f(n),g(n) هم رشدهستند.

نماد Small o (>):

اگر داشته باشیم $f(n)=o\,g(n)$ آنگاه رشته تابع g(n) مطلقا بیشتر از رشد تابع f(n) است.

نماد Small ω (<):

اگر داشته باشیم $f(n)=\omega\,g(n)$ آنگاه رشته تابع g(n) مطلقا کمتر از رشد تابع f(n) است.

معمولا ازنماد $\in$ بجای $=$ استفاده میشود.

1) $f(n)=\frac{n(n+1)}{2} \xrightarrow[\text{زمانی}]{\text{مرتبه}} n^2$, $g(n)=\frac{n^2(n+1)}{3} \longrightarrow n^3$

$f=O\ g$, $f=o\ g$, $g=\Omega f$, $g=\omega f$

2) $f(n)=n^3+10^7 \longrightarrow n^3$, $g(n)=n^3+2 \longrightarrow n^3$

$f=O\ g$, $g=\Omega f$, $g=\Theta f$, $g\neq o\ f$

3) $\log n \in O\ 2\log n = \log n \in O \log n^2$

$\log n \in \Theta\ 2\log n$

$\log n \notin o\ 2\log n$

4) $n\log n \in \Omega\ n^2 \longrightarrow \not{n}\log n \in \Omega\ \not{n} \times n$

$\log n \notin \Omega\ n$, $\log n \in O\ n$, $\log n \in o\ n$

5) $n\log^2 n \in \Theta\ n^2 \log n \longrightarrow n\log 2n \in \Theta\ n^2 \log n \longrightarrow \not{n}\log n \log n \in \Theta\ \not{n} \times n \log n$

$\log n \notin \Theta\ n$, $\log n \in O\ n$, $\log n \in o\ n$

5) $2^n \in O\ n!$

$2^n \in O\ n!$, $2^n \in o\ n!$

$$C < \log n < \log^{k>1} n < \log(n!) < n < n\log n < n^{a>1} < n^a \log n < b^n < n! < n^n$$

$$x=\log_b a \iff b^x=a$$

$$n^{(\log_2 m)} = m^{(\log_2 n)}$$

$$\log_a MN = \log_a M + \log_a N$$

$$\log n^k = k \log n$$

$$\log^k n = \log kn$$

$$\log_a \frac{M}{N} = \log_a M - \log_a N$$

$$\log_a a = 1$$

$$\log_a 1 = 0$$

$$\log_b a^M = M \log_b a$$

اگر داشته باشیم $f(n) \in O\ g(n)$ آنگاه کدام یک از موارد صحیح است

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

$\log(f(n)) \in O\ \log(g(n))$

$$\log_{b^M} a = \frac{1}{M} \log_b a$$

$$\log_{b^n} a^m = \frac{m}{n} \log_b a$$

$$\log_b a = \frac{1}{\log_a b}$$

$$\log_b a \times \log_c b = \log_c a$$

$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$a^{\log_a b} = b$$

مرتبه زمانی و مقدار دقیق k را محاسبه کنید.

```
for (i=1;i<=n²/2;i++)  ⟶ n²/2 -1+1= n²/2
{
  for (j=m;j>=i; j--)  ⟶ m-1+1
  {
    for (p=n;p>=1;p-=2) ⟶ n-1+1/2 = n/2
      {
        k++;
      }
  }
}
```

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

$$\sum_{i=1}^{\frac{n^2}{2}}m = m\frac{n^2}{2} = \frac{mn^2}{2}$$

$$\sum_{i=1}^{\frac{n^2}{2}}i = \frac{\frac{n^2}{2}(\frac{n^2}{2}+1)}{2} = \frac{\frac{n^4}{4}+\frac{n^2}{2}}{2} = \frac{\frac{n^4+2n^2}{4}}{2} = \frac{n^4+2n^2}{8}$$

$$\sum_{i=1}^{\frac{n^2}{2}}1 = \frac{n^2}{2}$$

$$= \frac{mn^2}{2} + \frac{n^4+2n^2}{8} + \frac{n^2}{2} = \frac{4mn^2-n^4+2n^2}{8}$$

n	m	i	j	p	k
2	5	1	5	2	1
		2	4	0	2
			3		3
			2		4
			1		5
			5	2	6
			4		7
			3		8
			2		⑨

$$\frac{4mn^3-n^5+2n^3}{8} = \frac{4\times5\times2^3-2^5+2\times2^3}{8} = 9 = k$$

مقدار دقیق k و تعداد دفعات اجراء $\frac{n}{2}\sum_{i=1}^{\frac{n^2}{2}}(m-i+1) = \frac{n}{2}(\frac{4mn^2-n^4+2n^2}{8}) = \frac{4mn^3-n^5+2n^3}{8}$

مرتبه زمانی: $O(mn^3)$ Order هیچگاه منفی نمیشود $mn^3-n^5>0$

کدام یک از معادلات صحیح است.

1) غلط $n\log^2 n \in \Theta(\log(\log n))$

$\log(2^n) \in \Theta \log(\log n)$

$2^n > \log n$

غلط ⟶ $n\log^2 n \in \Theta(\log(\log n))$ ⟶ صحیح $n\log^2 n \in \Omega(\log(\log n))$

2) صحیح $2^n \in \Omega(\log n!)$

$2^n < n!$

$2^n > \log(n!)$

صحیح ⟶ $2^n \in \Omega(\log n!)$ ⟶ صحیح $2^n \in \Theta(\log n!)$

3) صحیح $n2^n \in O(n!)$

if $n>5$

$n2^n < n(n-1)!$

صحیح ⟶ $n2^n \in O(n!)$

4) صحیح $n^2 2^n \in \Omega(\log n)^n$

$\log n = k$

$10^k = n$

$n^2 2^n = (10)^{k^2} \times 2^n = \longrightarrow 10^{2k} \times 2^n > k^n \longrightarrow$ صحیح $n^2 2^n \in \Theta(\log n)^n$

$n! \in O(n^n)$

$\text{Log } n! \in O \text{ Log } n^n \longrightarrow \text{Log } n! = n \text{ Log } n$

$\text{Log } n! \in O \text{ n Log n}$

توابع بازگشتی:

توابعی هستند که در داخل برنامه خود را فراخوانی میکنند.

```
int f(int n)
{
if (n==1) return 1;
else
return (n×f(n-1))
}
```

stopper

$$f(n)=\begin{cases}1 & \text{If } n=4\\ n\times f(n-1) & \text{else}\end{cases}$$

If n=4 $\rightarrow 4\times\overbrace{f(3)}^{6} \rightarrow 24$

$3\times\overbrace{f(2)}^{2}$

$2\times\overbrace{f(1)}^{1}$

دستور return در انتها محاسبه شده و بازگردانده میشود.

با توجه به مثال بالا تابع f برای محاسبه فاکتوریل مقدار ورودی بصورت بازگشتی بکار میرود.

---

1) $\begin{cases}n=2^k\\ T(n)=2T(\frac{n}{2})+1\\ T(1)=0\end{cases}$ $\xrightarrow{n=\frac{n}{2}}$ $T(n)=2\underbrace{T(\frac{n}{2})}_{2T(\frac{n}{4})+1}+1=4\underbrace{T(\frac{n}{4})}_{2T(\frac{n}{8})+1}+2+1=8\underbrace{T(\frac{n}{8})}_{2T(\frac{n}{16})+1}+4+2+1=16T(\frac{n}{16})+8+4+2+1$

$$=\underbrace{2^kT(\frac{n}{2^k})}_{=0}+1+2+4+8+.....+2^{k-1}=\begin{cases}n=2^k\\ T(\frac{n}{n})=T(1)=0\end{cases}\longrightarrow=1+2+4+8+.....+2^{k-1}=\sum_{i=0}^{k-1}2^i$$

سری هندسی با قدر نسبت بزرگتر از 1 $\sum_{i=0}^{k-1}a^i=a+a^2+a^3+.....+a^n=\frac{a^{n+1}-a}{q-1}=\frac{\text{اولین عدد - عدد بعد از آخر}}{\text{1 - قدرنسبت}}$

$$\longrightarrow\sum_{i=0}^{k-1}2^i=1+2+4+8+.....+2^{k-1}=\frac{2^k-1}{2-1}=2^k-1=n-1$$

مرتبه زمانی $O=(n)$

---

2) $\begin{cases}n=2^k\\ T(n)=T(\frac{n}{2})+1\\ T(1)=1\end{cases}$ $\xrightarrow{n=\frac{n}{2}}$ $T(n)=\underbrace{T(\frac{n}{2})}_{T(\frac{n}{4})+1}+1=\underbrace{T(\frac{n}{4})}_{T(\frac{n}{8})+1}+1+1=\underbrace{T(\frac{n}{8})}_{T(\frac{n}{16})+1}+1+1+1=T(\frac{n}{16})+1+1+1+1$

$$=\underbrace{T(\frac{n}{2^k})}_{=1}+K=K+1=2^k+1=\log_2 n+1$$

مرتبه زمانی $O=(\log_2 n)$

در دو مثال بالا فرمول را از حالت بازگشتی به حالت غیر بازگشتی تبدیل کرده ایم.

Master theory (قضیه اساسی تعمیم یافته)

اگر داشته باشیم $T(n)=aT(\frac{n}{b})+f(n)$ با شرط اولیه $f:N\longrightarrow R^+$, b>1, a>1 سه حالت ایجاد میشود.

1) $f(n)\in O(n^{\log_b a-\varepsilon})\longrightarrow T(n)\in\theta(n^{\log_b a})$

2) $f(n)\in O(n^{\log_b a}.\log^k n)\longrightarrow T(n)\in\theta(n^{\log_b a}.\log^{k+1} b)$

3) $f(n)\in\Omega(n^{\log_b a+\varepsilon})\longrightarrow T(n)\in\theta f(n)$

با استفاده از این قضیه فقط مرتبه زمانی قابل محاسبه است.

$T(n)=2T(\frac{n}{2})+1$ $\begin{cases}n^{\log_b a}=n^{\log_2 2}=n\\ f(n)=1\end{cases}$ $\xrightarrow{①}$ $n^{\log_b a}>f(n)\longrightarrow\theta(n)$

$T(n)=4T(\frac{n}{2})+n$ $\begin{cases}n^{\log_b a}=n^{\log_2 4}=n^2\\ f(n)=n\end{cases}$ $\xrightarrow{①}$ $n^{\log_b a}>f(n)\longrightarrow\theta(n^2)$

$$T(n)=2T(\frac{n}{2})+n \begin{cases} n^{\log_b a}=n^{\log_2 2}=n \\ f(n)=n \end{cases} \xrightarrow{②} n^{\log_b a}=f(n) \longrightarrow n^{\log_b a}.\log n > f(n) \longrightarrow \theta(n \log n)$$

$$T(n)=3T(\frac{n}{2})+n^2 \begin{cases} n^{\log_b a}=n^{\log_2 3} \\ f(n)=n^2 \end{cases} \xrightarrow{③} f(n) > n^{\log_b a} \longrightarrow n^2 > n^{\log_2 3} \longrightarrow \theta(n^2)$$

$$T(n)=2T(\frac{n}{4})+\log n \begin{cases} n^{\log_b a}=n^{\log_4 2}=n^{\frac{1}{2}}=\sqrt{n} \\ f(n)=\log n \end{cases} \xrightarrow{①} n^{\log_b a} > f(n) \longrightarrow \sqrt{n} > \log n \longrightarrow \theta(\sqrt{n})$$

$$T(n)=4T(\frac{n}{2})+n^2 \begin{cases} n^{\log_b a}=n^{\log_2 4}=n^2 \\ f(n)=n^2 \end{cases} \xrightarrow{②} n^{\log_b a}=f(n) \longrightarrow n^{\log_b a}.\log n > f(n) \longrightarrow \theta(n^2 \log n)$$

$$T(n)=8T(\frac{n}{2})+n^3\log n \begin{cases} n^{\log_b a}=n^{\log_2 8}=n^3 \\ f(n)=n^3\log n \end{cases} \xrightarrow{③} f(n) > n^{\log_b a} \longrightarrow n^3\log n > n^3 \longrightarrow \theta(n^3 \log n)$$

$$T(n)=2T(\frac{n}{2})+n^2-2 \begin{cases} n^{\log_b a}=n^{\log_2 2}=n \\ f(n)=n^2-2 \end{cases} \xrightarrow{③} f(n) > n^{\log_b a} \longrightarrow n^2-2 > n \longrightarrow \theta(n^2)$$

توابع بازگشتی (بازگشتی همگن و غیر همگن)

توابع بازگشتی به دو دسته همگن و غیر همگن تقسیم میشوند ،که ممکن است ضرایب آنها ثابت یا متغییر باشند .

توابع بازگشتی همگن (خطی با ضرایب ثابت) $a_0t(n)+a_1t(n-1)+\ldots.+a_kt(n-k)=0$

همگن =معادله برابر صفر میشود، خطی=توان ندارد.

$a_0t(n)+a_1t(n-1)+\ldots.+a_kt(n-k)=0 \xrightarrow[\text{مشخصه}]{\text{معادله}} a_0x^n+a_1x^{n-1}+a_2x^{n-2}+\ldots..+a_kx^{n-k}=0$ این معادله دارای k ریشه است. $r_1, r_2, \ldots., r_k$

فرمت غیر بازگشتی تابع بازگشتی فوق بدین شکل است. $c_1(r_1)^n+c_2(r_2)^n+\ldots..+c_k(r_k)^n$ ضرایب عددی =C

مثال: تابع بازگشتی زیر را به فرم غیر بازگشتی تبدیل کرده و order زمانی آنرا محاسبه کنید؟

$$T(n)=\begin{cases} 0 & n=0 \\ 5 & n=1 \\ 3T(n-1)+4T(n-2) & \text{else} \end{cases}$$

$$T(n)=3T(n-1)+4T(n-2) \longrightarrow T(n)-3T(n-1)-4T(n-2)=0$$

$$x^2-3x-4=0 \longrightarrow (x-4)(x+1)=0 \begin{cases} x=-1=r_1 \\ x=4=r_2 \end{cases}$$

$$T(n)=c_1(r_1)^n+c_2(r_2)^n=c_1(-1)^n+c_2(4)^n \begin{cases} n=0 \longrightarrow T(n)=0 \longrightarrow c_1+c_2=0 \longrightarrow c_1=-c_2 \\ n=1 \longrightarrow T(n)=5 \longrightarrow -c_1+4c_2=5 \end{cases}$$

$$\longrightarrow 5c_2=5 \longrightarrow c_2=1, c_1=-1$$

$$T(n)=c_1(-1)^n+c_2(4)^n=-1(-1)^n+1(4)^n=\underbrace{4^n-(-1)^n}_{4^n+(-1)^{n+1}} \longrightarrow O=(4^n)_{\text{order}}$$

توابع بازگشتی غیرهمگن (خطی با ضرایب ثابت) $a_0t(n)+a_1t(n-1)+....+a_kt(n-k)=b^n.f(n)$

$$a_0t(n)+a_1t(n-1)+....+a_kt(n-k)=b^n.f(n) \xrightarrow[\text{مشخصه}]{\text{معادله}} (x-r_1)(x-r_2).....(x-r_k)(x-b)^{d+1}=0$$

d=درجه تابع f(n)
k=حداقل ریشه معادله

$$c_1(r_1)^n+c_2(r_2)^n+.....+c_k(r_k)^n$$

مثال: تابع بازگشتی زیر را به فرم غیر بازگشتی تبدیل کرده و order زمانی آنرا محاسبه کنید؟

$$T(n)=\begin{cases}1 & n\leq 2\\ 2T(n-1)+(n+5)3^n & else\end{cases}$$

بیشترین مقداری که از n در ضرایب T کم میشود درجه معادله میشود.

$$T(n)-2T(n-1)=(n+5)3^n \begin{cases}f(n)=(n+5)\\ d=1\\ b=3\end{cases} \longrightarrow (x-2x^0)(x-3)^2=0 \longrightarrow (x-2)(x-3)^2=0 \begin{cases}x=2=r_1\\ x=3=r_2\\ x=3=r_3\end{cases}$$ ریشه مضاعف

به تعداد ریشه های مضاعف ضرایب C در n ضرب میشوند.یعنی مضاعف اول در n ،مضاعف دوم در n^2 ،مضاعف سوم در n^3 و.....

$$c_1(r_1)^n+c_2(r_2)^n+c_3n(r_3)^n \longrightarrow c_1(2)^n+c_2(3)^n+c_3n(3)^n \longrightarrow O(n3^n) \text{ order}$$

مثال: تابع بازگشتی زیر را به فرم غیر بازگشتی تبدیل کرده و order زمانی آنرا محاسبه کنید؟

$$T(n)=\begin{cases}1 & n\leq 2\\ 4T(n-2)+(n+1)^2 2^n & else\end{cases}$$

$$T(n)-4T(n-2)=(n+1)^2 2^n \begin{cases}f(n)=(n+1)^2\\ d=2\\ b=2\end{cases} \longrightarrow (x^2-x^0 4)(x-2)^3 \longrightarrow (x^2-4)(x-2)^3 \begin{cases}x=-2=r_1\\ x=2=r_2\\ x=2=r_3\\ x=2=r_4\\ x=2=r_5\end{cases}$$

$$c_1(r_1)^n+c_2(r_2)^n+c_3n(r_3)^n+c_4n^2(r_4)^n+c_5n^3(r_5)^n \longrightarrow c_1(-2)^n+c_2(2)^n+c_3n(2)^n+c_4n^2(2)^n+c_5n^3(2)^n \longrightarrow O(n^3 2^n) \text{order}$$

تابع بازگشتی زیر را به فرم غیر بازگشتی تبدیل کرده و order زمانی آنرا محاسبه کنید؟

$$\begin{cases} n=2^k \\ T(n)=2T(\frac{n}{2})+n-1 \\ T(1)=0 \end{cases} \xrightarrow{n=\frac{n}{2}} T(n)=\underbrace{2T(\frac{n}{2})}_{2T(\frac{n}{4})+\frac{n}{2}-1}+n-1=\underbrace{4T(\frac{n}{4})}_{2T(\frac{n}{8})+\frac{n}{4}-1}+\overbrace{2n-2^1-2^0}^{n-2+n-1} = \underbrace{8T(\frac{n}{8})}_{2T(\frac{n}{16})+\frac{n}{8}-1}+\overbrace{n-4+n-2+n-1}^{3n-2^2-2^1-2^0} = 16T(\frac{n}{16})+\overbrace{n-8+n-4+n-2+n-1}^{4n-2^3-2^2-2^1-2^0}$$

$$T(n)=\underbrace{2^k T(\frac{n}{2^k})}_{T(1)=0}+kn-(2^{k-1}+\dots+2^2+2^1+1)=kn-\sum_{i=0}^{k-1}2^i \xrightarrow{2^k=n\ ,k=\log_2 n} =n\log_2 n-(n-1)$$

$$=\frac{2^k-1}{2-1}=2^k-1=(n-1)$$

مرتبه زمانی $O=(n\log_2 n)$

تابع بازگشتی زیر را به فرم غیر بازگشتی تبدیل کرده و order زمانی آنرا محاسبه کنید؟

$$T(n)=\begin{cases} n & n\leqslant 2 \\ 5T(n-1)-8T(n-2)+4T(n-3) & else \end{cases}$$

$$T(n)=5T(n-1)-8T(n-2)+4T(n-3) \longrightarrow T(n)-5T(n-1)+8T(n-2)-4T(n-3)=0 \longrightarrow x^3-5x^2+8x-4=0$$

Vieta's theorem

$$ax^3+bx^2+cx+d=0 \begin{cases} r_1+r_2+r_3=-\frac{b}{a} \\ r_1.r_2.r_3=-\frac{d}{a} \\ r_1r_2+r_1r_3+r_2r_3=\frac{c}{a} \end{cases} \xrightarrow[a+b+c+d=0]{r_1=1} \begin{cases} r_1+r_2+r_3=-\frac{-5}{1} \rightarrow r_2+r_3=4 \\ r_1.r_2.r_3=-\frac{-4}{1} \rightarrow r_2.r_3=4 \end{cases} \longrightarrow \begin{cases} r_1=1 \\ r_2=2 \\ r_3=2 \end{cases}$$

$$x^3-5x^2+8x-4=0$$

(Or)

$$x^3-5x^2+8x-4=0 \rightarrow x^3-5x^2+5x+3x-3-1=0 \rightarrow \underbrace{x^3-1}_{(x-1)(x^2+x+1)}\underbrace{-5x^2+5x}_{5x(x-1)}+\underbrace{3x-3}_{3(x-1)}=0 \rightarrow (x-1)(x^2+x+1)-5x(x-1)+3(x-1) \rightarrow (x-1)(x^2+x+1-5x+3)$$

$$\rightarrow (x-1)(x^2-4x+4)=0 \rightarrow (x-1)(x-2)(x-2)=0 \longrightarrow r_1=1, r_2=2, r_3=2$$

$$T(n)=c_1(r_1)^n+c_2(r_2)^n+c_3n(r_3)^n \longrightarrow T(n)=c_1(1)^n+c_2(2)^n+c_3n(2)^n$$

مرتبه زمانی $O=(n2^n)$

$$\begin{cases} n=2^k \\ T(n)=2T(\frac{n}{2})+2 \\ T(1)=0 \\ T(2)=1 \end{cases} \qquad T(n)=2\underbrace{T(\frac{n}{2})}_{2T(\frac{n}{4})+2}+2=4\underbrace{T(\frac{n}{4})}_{2T(\frac{n}{8})+2}+4+2=4\underbrace{T(\frac{n}{8})}_{2T(\frac{n}{16})+2}+8+4+2=16T(\frac{n}{16})+16+8+4+2$$

$$T(2)=T(\frac{2^k}{2^x}) \rightarrow T(2)=T(2^{k-x}) \rightarrow k-x=1 \rightarrow x=k-1$$

$$T(n)=2^{k-1}\underbrace{T(\frac{2^k}{2^{k-1}})}_{T(2)=1}+2^{k-1}+\ldots.+2^3+2^2+2^1=2^{k-1}+\underbrace{(2^1+2^2+2^3+2^{k-1})}_{\sum_{i=1}^{k-1}2^i=(n-2)}=2^{k-1}+(n-2)=\frac{2^k}{2}+(n-2)=\frac{n}{2}+(n-2)=\frac{3n}{2}-2$$

$$T(n)=\frac{3n}{2}-2$$

$$\begin{cases} n=2^k \\ T(n)=8T(\frac{n}{2})+n^3\log n \\ k=\log_2 n \end{cases} \longrightarrow T(2^k)=8T(\frac{2^k}{2})+k2^{3k} \longrightarrow T(2^k)=8T(2^{k-1})+k2^{3k} \xrightarrow{T(2^k)=g(k)} g(k)=8g(k-1)+k(2^3)^k$$

$$g(k)-8g(k-1)=k8^k \begin{cases} f(k)=k \\ d=1 \\ b=8 \end{cases} \longrightarrow (x-8)(x-8)^2=0 \begin{cases} r1=8 \\ r2=8 \\ r3=8 \end{cases}$$

$$g(k)=c_1(8)^k+c_2k(8)^k+c_3k^2(8)^k \longrightarrow g(k)=c_1(n)^3+c_2\log n(n)^3+c_3\log^2(n)^3$$

$$\begin{cases} 8^k=8^{\log_2 n}=n^{\log_2 8}=n^3 \\ k^2=(\log_2 n)^2=(\log_2^2 n) \end{cases}$$

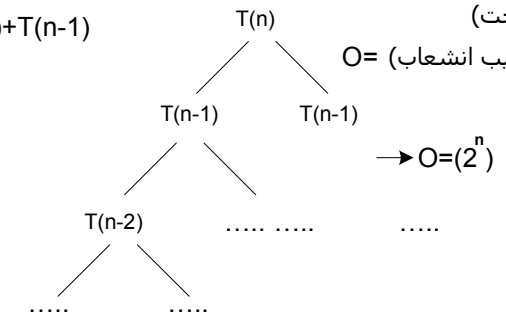
$$O(n^3\log^2 n)$$

$\begin{cases} T(n)=2T(n-1) \\ T(1)=1 \end{cases} \longrightarrow n^{\log_2 2} \longrightarrow O(n)$

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

(عمق درخت)

(ضریب انشعاب) O=



$\longrightarrow O=(2^n)$

ممکن است دو تابع بازگشتی عملکرد و نتیجه کاملا یکسان داشته باشند و لی دارای مرتبه زمانی متفاوت

$\begin{cases} T(n)=T(n-1)^2 \\ T(1)=0 \end{cases} \longrightarrow n^{\log_2 2} \longrightarrow O(n)$

$T(3)=T(2)^2=T(1)^4=0$

$\begin{cases} T(n)=T(n-1)*T(n-1) \\ T(1)=0 \end{cases}$

$T(3)=T(2)*T(2) \longrightarrow O=(2^n)$

$T(1)*T(1)*T(1)*T(1)$

$\begin{cases} T(n)=\dfrac{\sqrt{T(\frac{n}{2})+T(\frac{n}{2})}}{2T(\frac{n}{2})} \end{cases} \quad O=(3^{\log_2 n})=(n^{\log_2 3})$

$\begin{cases} T(n)=\dfrac{\sqrt{T(\frac{n}{3})+T(\frac{n}{3})}}{2T(\frac{n}{2})} \end{cases}$

(طولانیترین عمق درخت)

(ضریب انشعاب) O=

$O=(3^{\log_3 n})=(n^{\log_3 3})=n$

روش تغییر متغییر:

یکی از روشهای حل توابع بازگشتی استفاده از تغییر متغییر است.

1) $T(n)=aT(\frac{n}{2})+f(n) \longrightarrow n=b^k$ ← تغییر متغییر

2) $T(n)=aT(\sqrt[c]{n})+f(n) \longrightarrow n=c^{c^k}$ ← تغییر متغییر

3) $\dfrac{T(n)}{n}=\dfrac{aT(\sqrt[c]{n})}{\sqrt{n}} \longrightarrow n=c^{c^k}$ ← تغییر متغییر

مثال: تابع زیر را به روش بازگشتی حل نمائید؟

$\begin{cases} n=2^k \\ T(n)=2T(\frac{n}{2})+n \end{cases} \longrightarrow T(2^k)=2T(\frac{2^k}{2})+2^k \longrightarrow T(2^k)=2T(2^{k-1})+2^k \xrightarrow{\text{هم ارزی(ریختی)در تابع – تابع هوموورف}} T(2^k)=g(k)$

$\longrightarrow g(k)=2g(k-1)+2^k \longrightarrow g(k)-2g(k-1)=2^k \xrightarrow{\text{تابع بازگشتی غیر همگن}} g(k)-2g(k-1)=2^k.\underbrace{f(k)}_{1}$ $\begin{cases} f(k)=1 \\ d=0 \\ b=2 \end{cases}$

$\longrightarrow (x-2)(x-2)=0 \begin{cases} r1=2 \\ r2=2 \end{cases} \longrightarrow g(k)=c_1(2)^k+c_2 k(2)^k \longrightarrow O(k2^k) \xrightarrow[\text{Log}_2 n=k]{n=2^k} O(n\log_2 n)$

$g(k)=c_1(n)+c_2(n\log n)$

$O(n\log_2 n)$

مثال: تابع زیر را به روش بازگشتی حل نمائید؟

$\begin{cases} n=2^k \\ T(n)=4T(\frac{n}{2})+n\log n \end{cases} \longrightarrow T(2^k)=4T(\frac{2^k}{2})+k2 \longrightarrow T(2^k)=4T(2^{k-1})+k2^k \xrightarrow{T(2^k)=g(k)} g(k)=4g(k-1)+k2^k$ $\begin{cases} f(k)=k \\ d=1 \\ b=2 \end{cases}$

$\longrightarrow g(k)-4g(k-1)=k2^k \longrightarrow (x-4)(x-2)^2=0 \begin{cases} r1=4 \\ r2=2 \\ r3=2 \end{cases} \longrightarrow g(k)=c_1(4)^k+c_2(2)^k+c_3 k(2)^k$

$g(k)=c_1(n^2)+c_2(n)+c_3(n\log n)$

$\begin{cases} 4^k=4^{\log_2 n}=n^{\log_2 4}=n^2 \\ 2^k=2^{\log_2 n}=n^{\log_2 2}=n \\ k2^k=n\log n \end{cases} \longrightarrow n^2>n\log n>n$

$O(n^2)$

روش تقسیم و حل: (Divide and Conquer)

یکی از روشهای حل مسئله است که در ابتدا از مسئله اصلی شروع میکنند و مسئله را به بخشهای کوچکتر (زیر مسئله) و مستقل تقسیم میکنند.این روش منطبق بر الگوی حل مسئله از بالا به پایین (Top-Down) است.در هنگام تقسیم یک مسئله بزرگ به مسائل کوچک عموما تقسیم به دو بخش انجام میشود.بطور کلی تعداد تقسیمات باید به نحوی انتخاب شود که کارائی الگوریتم افزایش یابد.

نکته مهم در روش تقسیم و حل اینست که همه مسائل را نمیتوان از روش تقسیم و حل پاسخ داد .

بعنوان مثال برای یافتن بزرگترین عدد میان n عدد صحیح میتوان مسئله را به دو زیر مسئله تقسیم کرد.به این ترتیب که هر زیر مسئله دارای طول $\frac{n}{2}$ است،پس بزرگترین میتوان بزرگترین عدد در هر زیر مسئله را محاسبه کرد و در نهایت بین دو عدد بدست آمده از دو زیر مسئله بزرگترین عدد را بعنوان خروجی مشخص نمود.ولی دستگاه n معادله و n مجهول را نمیتوان از روش تقسیم و حل بررسی کرد.زیرا مسائل مستقل از یکدیگر نیستند.

در شرایط زیر الگوریتم تقسیم و حل برای حل یک مسئله مناسب است.

1-زمان مورد نیاز برای شکستن یک مجموعه n تایی به زیر مجموعه های کوچکتر و نیز ادغام آنها مناسب باشد.

2-زیر مسائل مستقل از یکدیگر باشند.

3-اندازه زیر مسائل ترجیحا با هم برابر باشند.

کلیات روش تقسیم و حل:

1-اگر مسئله به اندازه کافی کوچک است آنرا به روش مناسبی حل کنیم.

2-اگر مسئله بزرگ است روشی برای تقسیم آن به چند زیر مسئله پیدا میکنیم و عمل تقسیم را انجام میدهیم.

3-مسائل کوچک را حل میکنیم.

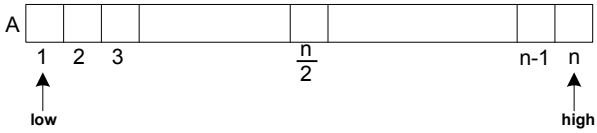
4- پاسخ مسائل کوچک را با یکدیگر ادغام میکنیم تا پاسخ مسئله بزرگ بدست آید.

1 الگوریتم جستجوی دودویی:

فرض میکنیم تا آرایه A به طول n از اعداد صحیح که بصورت صعودی مرتب شده اند وجود دارند و قرار است کلید X در این آرایه جستجو شوند و اندیس خانه ای از آرایه که شامل کلید X است بازگردد.میتوان اینگونه ادعا کرد که تنها دو حالت پیش می آید.

1-جستجوی موفق:در اینحالت لزوما کلید داده شده برابر یکی از خانه های آرایه است.

2-جستجوی ناموفق:در اینحالت کلید کلید X در آرایه وجود ندارد.



حالت بازگشتی

```
Binary-search(A,low,high,x)
{
If (low>high) return 0;
Else
  {
   m=⌊(low+high)/2⌋
   if(x==A[m]) return m;
   else if (x>A[m])
   binary-search(A,m+1,high,x);
   else
   binary-search(A,low,m-1,x);
   }
}
```

تعداد مقایسه 1

یک فراخوانی $\frac{n}{2}$

حالت غیر بازگشتی

```
While(low<high)
{
m=⌊(low+high)/2⌋

If (x==A[m])   return m;
else if (x>A[m])
low=m+1;
else
high=m-1;
}
Cout<<"your number is not existed.";
```

رابطه الگوریتم جستجوی دو دویی $T(n)=T(\frac{n}{2})+1$

مرتبه زمانی $O(\log_2 n)$

جستجوی موفق و ناموق درالگوریتم جستجوی دودویی:

مثال: آرایه A به شکل زیر داده شده است ،مشخص کنید میانگین تعداد جستجوها در حالت جستجوی موفق و ناموفق برای هر یک از اندیسها؟

1	3	4	7	10	14	18	19	25	30	33
0	1	2	3	4	5	6	7	8	9	10

جستجوی ناموفق

تعداد مقایسه	3	4	4	3	4	4	3	4	4	3	4	4
	0	2	3.5	5	8	12	16	18.5	21	28	32	35
شماره اندیس	0	1	2	3	4	5	6	7	8	9	10	11
شماره اندیس بررسی شده	5	5	5	5	5	5	5	5	5	5	5	5
	2	2	2	2	2	2	8	8	8	8	8	8
	0	0	0	3	3	3	6	6	6	9	9	9
		1	1		4	4		7	7		10	10

جستجوی موفق

تعداد مقایسه	3	4	2	3	4	1	3	4	2	3	4
	1	3	4	7	10	14	18	19	25	30	33
شماره اندیس	0	1	2	3	4	5	6	7	8	9	10
شماره اندیس بررسی شده	5	5	5	5	5	5	5	5	5	5	5
	2	2	2	2	2		8	8	8	8	8
	0	0		3	3		6	6		9	9
		1			4			7			10

آرایه ساختگی برای مقایسه با آرایه اصلی

$$\text{میانگین مقایسات در جستجوی موفق} = \frac{44}{12} = 3.66 = P(n)$$

$$\text{میانگین مقایسات در جستجوی موفق} = \frac{\text{طول آرایه}}{} = \frac{33}{11} = 3 = S(n)$$

1-تعداد مقایسات به ازای هر خانه از آرایه در حالت جستجوی ناموفق برابر است با $\lfloor \log_2 n \rfloor$ or $\lfloor \log_2 n \rfloor + 1$

2-اگر S(n) میانگین تعداد جستجوهای موفق و P(n) میانگین جستجوهای ناموفق باشد.

$$S(n)=(1+\frac{1}{n})P(n)-1$$

الگوریتم یافتن بزرگترین و کوچکترین عنصر آرایه:

الگوریتم غیر بازگشتی

```
min-max(A)
{
min=max=A[0];
for (i=1;i<=n-1;i++)
   {
   if (A[i]>=max) max=a[i];
   if (A[i]<min min=A[i];
}
```

تعداد دفعات اجراء n-1-1+1=n-1

مرتبه زمانی O(n)

تعداد مقایسات 2(n-1)

$T(n)=n$

الگوریتم بازگشتی

0	1	2	3	4	5	6	7
2	10	5	13	20	1	42	17

min=2 max=1 | min=5 max=13 | min=1 max=20 | min=17 max=42

min=2 max=13 | min=1 max=42

min=1 max=42

(A,0,7,min,max)
m=3
(A,0,2,min1,max1)
m=1
(A,0,0,min1,max1)
A[0]=2=min1=max1

(A,0,7,min,max)
m=3
(A,3,7,min2,max2)
m=5
(A,5,7,min2,max2)
m=6
(A,6,7,min2,max2)
max2=A[6]=42
min2=A[7]=17

```
D&C-min-max(A,low,high,min,max)
{
if (low==high)
max=min=A[low];
else
  If (low==high-1)
    {
     if (A[low]>=A[high]
     max=A[low];min=A[high];
     else
     max=A[high];min=A[low];
     }
  else
  {
   m=⌊(low+high)/2⌋

   D&C-min-max(A,low,m-1,min1,max1);
   D&C-min-max(A,m,high,min2,max2);
   If(min1<min2) min=min1;
   else
   min=min2;
   if(max1>max2) max=max1;
   else max=max2;
  }
}
```

$$\begin{cases} n=2^k \\ T(n)=2T(\frac{n}{2})+2 \\ T(1)=0 \\ T(2)=1 \end{cases} \qquad T(n)=\frac{3n}{2}-2$$

ضرب چند جمله ایها:

میخواهیم در چند جمله ای به نامهای A(n),B(n) را که از درجه n-1 هستند در هم ضرب کنیم ،سه الگوریتم متفاوت وجود دارد که یکی از روشها از قوانین تقسیم و حل استفاده نمیکند.

1-الگوریتم غیر تقسیم و حل:

$$\begin{cases} A(n)=a_0+a_1x+a_2x^2+\ldots\ldots+a_{n-1}x^{n-1} \\ B(n)=b_0+b_1x+b_2x^2+\ldots\ldots+b_{n-1}x^{n-1} \end{cases}$$

آنگاه چند جمله ای A(n) را میتوان بصورت آرایه P[0…..n-1] (بطول n) و چند جمله ای B(n) را میتوان بصورت q[0….n-1] (بطول n) در نظر گرفت و نتیجه ضرب این دو چند جمله ای آرایه C[0……..2n-2] (بطول 2n-1) خواهد بود که مقادیر آرایه C ضرایب چند جمله ای هستند.

$$C(n)=c_0+c_1x+c_2x^2+\ldots.+c_{2n-2}x^{2n-2} \quad \begin{cases} C_0=a_0b_0 \\ C_1=a_0b_1+a_1b_0 \\ C_2=a_0b_2+a_1b_1+a_2b_0 \end{cases} \longrightarrow \boxed{C_k=\sum_{i=0}^{k} a_i\ b_{k-i}}$$

این الگوریتم دو حلقه for تو در تو خواهد بود .با بررسی الگوریتم مربوط به این روش مشخص میشود که برای ضرب چند جمله ایهای a_n , b_n دو حلقه for مستقل بطول n وجود دارد که میتوان نتیجه گرفت مرتبه زمانی این روش برابر $\theta(n^2)$ خواهد بود.

2 روش تقسیم و حل

در روش تقسیم و حل چند جمله ایهای از درجه n هر کدام به دو بخش تقسیم میشود ،سپس زیر بخشهای این دو چند جمله ای در یکدیگر ضرب میشوند اگر A(x),B(x) دو چند جمله ای به شکل زیر باشند.

$$\overbrace{A_L(x)}^{\text{نیمه سمت چپ چند جمله ای}} \qquad \overbrace{\qquad\qquad}^{\text{نیمه سمت راست چند جمله ای}}$$

$$A(x)=a_0+a_1x+a_2x^2+\ldots\ldots+a_{\frac{n}{2}-1}x^{\frac{n}{2}-1}+a_{\frac{n}{2}}x^{\frac{n}{2}}+a_{\frac{n}{2}+1}x^{\frac{n}{2}+1}+\ldots..+a_{n-1}x^{n-1}$$

$$x^{\frac{n}{2}}\underbrace{\left(a_{\frac{n}{2}}+a_{\frac{n}{2}+1}x+a_{\frac{n}{2}+2}x^2+\ldots..+a_{n-1}x^{\frac{n}{2}-1}\right)}_{A_R(x)}$$

$$\overbrace{2+5x-4x^2+7x^5}$$

$$a_0=2,a_1=5,a_2=-4,a_3=a_4=0,a_5=7$$

$$\overbrace{\qquad\qquad}^{B_L(x)}$$

$$B(x)=b_0+b_1x+b_2x^2+\ldots\ldots+b_{\frac{n}{2}-1}x^{\frac{n}{2}-1}+b_{\frac{n}{2}}x^{\frac{n}{2}}+b_{\frac{n}{2}+1}x^{\frac{n}{2}+1}+\ldots..+b_{n-1}x^{n-1}$$

$$x^{\frac{n}{2}}\underbrace{\left(b_{\frac{n}{2}}+b_{\frac{n}{2}+1}x+b_{\frac{n}{2}+2}x^2+\ldots..+b_{n-1}x^{\frac{n}{2}-1}\right)}_{B_R(x)}$$

$$A(x)=5+2x-10x^2+9x^3+x^4-8x^5 \longrightarrow \underbrace{5+2x-10x^2}_{A_L(x)}+x^3\underbrace{(9+x-8x^2)}_{A_R(x)}$$

$$A(x)=A_L(x)+A_R(x)x^{\frac{n}{2}}$$

$$B(x)=B_L(x)+B_R(x)x^{\frac{n}{2}}$$

$$A(x).B(x)=\underbrace{A_L(x).B_L(x)}_{A}+\underbrace{\left(A_L(x).B_R(x)+A_R(x).B_L(x)\right)}_{*}x^{\frac{n}{2}}+\underbrace{A_R(x).B_R(x)}_{B}.x^n$$

$$\boxed{\begin{array}{c} T(n)=4T(\frac{n}{2})+\theta(n) \\ O(n^2) \end{array}}$$

$$Z=(A_L(x)+A_R(x)).(B_L(x)+B_R(x))=\underbrace{A_L(x).B_L(x)}_{A}+\underbrace{A_L(x).B_R(x)+A_R(x).B_L(x)}_{*}+\underbrace{A_R(x).B_R(x)}_{B}$$

*=Z-A-B

$$\boxed{\begin{array}{c} T(n)=3T(\frac{n}{2})+\theta(n) \\ \theta(n^{Log_2^3}) \end{array}}$$

نکته:مرتبه زمانی دو چند جمله ای از درجه n برابر $\theta(n)$ است و مرتبه زمانی محاسبه مجموع دو ماتریس $n\times n$ برابر $\theta(n^2)$ است.

3 محاسبه a^n

برای محاسبه a^n بصورت خطی میتوان از یک حلقه for بطول n وبشکل زیر استفاده نمود.میتوان اینگونه ادعا نمود که مرتبه زمانی این روش خطی برابر $\theta(n)$ است.

محاسبه a^n به روش خطی:

```
P=1;
For(i=1;i<=n;i++)
P*=a;
```

$\rightarrow \boxed{O(\theta n)}$

محاسبه a^n به روش تقسیم وحل

$$a^n \begin{cases} a & n=1 \\ (a^{\frac{n}{2}})^2 & n=2k \\ a.a^{n-1} & n=2k+1 \end{cases}$$

با توجه به اینکه محاسبه a^n در این فرمول وابستگی مستقیم به محاسبه $a^{\frac{n}{2}}$ دارد مرتبه زمانی آن کمتر خواهد بود.تابع بازگشتی بدین صورت میباشد.

```
Int f(it n)
{
If (n==1) return a;
Else if (n%2==0) return (pow(f(n/2),2);
Else return (a*f(n-1);
}
```

$\rightarrow \boxed{\theta(\log_2 n)}$

مثال: آرایه A بطول 2k مفروض است تعداد مقایسه های لازم برای یافتن کمترین و بیشترین عدد در بین آرایه را محاسبه کنید؟(از بهترین فرمول در مرتبه زمانی استفاده شود)

$$T(n)=\frac{3n}{2}-2 \xrightarrow{n=2k} \frac{3(2k)}{2}-2=3k-2$$

4 الگوریتم مرتب سازی ادغامی (Merge Sort)

اگر یک آرایه nتایی از اعداد داشته باشیم آنگاه در مرتب سازی ادغامی این آرایه به دو بخش برابر تقسیم میشود که هر بخش جداگانه مرتب خواهد شد و حاصل این دو بخش مرتب شده به گونه ای ادغام میشود که آرایه نهایی نیز مرتب شده باشد.در مرتب سازی ادغامی تقسیم آرایه تا زمانی ادامه پیدا میکند که هر بخش شامل یک عضو از آرایه باشد ،آنگاه دو زیر آرایه تک عنصری با یکدیگر ادغام شده و زیر آرایه های سطح بالاتر خود را می سازند.

$$T(n)=2T(\frac{n}{2})+\theta(n)$$

$$\theta(n \log n)$$

الگوریتم Merge-Sort به شکل زیر است.

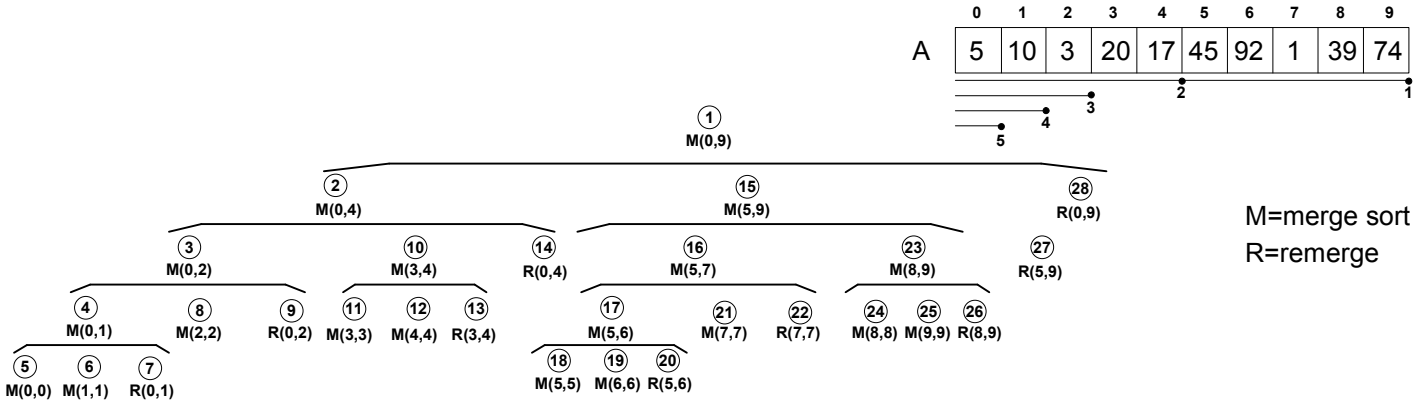
1-اگر تعداد عناصر آرایه بحد مناسبی کم باشد نیازی به مرتب سازی ادغامی نیست (یعنی یک عدد)
2-در غیر اینصورت اندیس عنصر میانی را پیدا کن
3-نیمه سمت راست و چپ آرایه را بطور جداگانه و به روش مرتب سازی ادغامی مرتب کن
4-دو نیمه مرتب شده آرایه را به گونه ای ادغام کن که آرایه نهایی مرتب باشد.

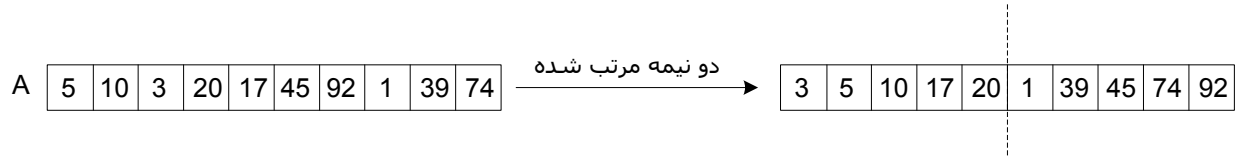
(زمان لازم برای اجرای تابع Remerge،این تابع در بدترین حالت n-1 مقایسه انجام میدهد.)

```
Merge sort(A,low,high)
{
If (low==high) return 0;
Else
m=⌊(low+high)/2⌋
Merge sort(A,low,m)
Merge sort(A,m+1,high)
Remerge(A,low,high,m)
```

m برای این ذکر میشود که اشاره گر برای مقایسه از m جلوتر نرود.

مثال: آرایه آرایه A بشکل زیر وجود دارد درخت بازگشتی و ترتیب فراخوانیها را به روش مرتب سازی ادغامی بدست آورید؟



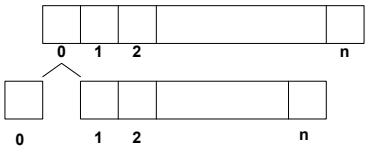


آرایه مرتب شده

1	3	5	10	17	20	39	45	74	92

در این روش از یک آرایه کمکی برای درج نهایی مرتب سازی استفاده میگردد.

مثال: اگر در الگوریتم merge-sort ترتیب شکستن آرایه بشکل زیر باشد orderزمانی آن چقدر است؟



$T(n)=\underbrace{T(1)}_{0}+T(n-1)+\theta(n)$

$T(1)+T(n-1)+\theta(n)$
$T(1)+T(n-2)+\theta(n)$
$T(1)+T(n-3)+\theta(n)$
$T(1)+T(1)+\theta(n)$

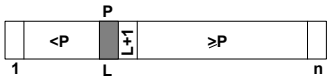
به تعداد n بار $\theta(n)$ تکرار میشود که اگر $\theta(n)=n$ در نظر بگیریم مرتبه زمانی برابر n^2 میشود.

$O(n^2)$

$$T(n)=T(1)+T(n-1)+\theta(n)=T(1)+T(n-2)+\theta(n)=\ldots\ldots=\underbrace{T(1)}_{0}+\underbrace{T(1)}_{0}+\theta(n)$$

4 الگوریتم مرتب سازی سریع (Quick Sort)

مبنای این مرتب سازی تعیین یک عدد میباشد(محور Pivot)بشکلی که در هر بار اجرای حلقه مرتب سازی مکان صحیح محمر انجام میشود.بعبارت دیگردر هر بار اجرای حلقه مرتب سازی ،عناصر قبل از محور ،کوچکتر از عدد و عناصر بعد از محور بزرگتر مساوی محور است.



الگوریتم مرتب سازی سریع شامل سه بخش است.

1-الگوریتم مربوط به پیدا کردن مکان محور در هر بار اجرای حلقه (تابع Partition که Pivot را پیدا میکند)

2- فراخوانی تابع بازگشتی Quick Sort برای عناصر کوچکتر از محور

3- فراخوانی تابع بازگشتی Quick Sort برای عناصر بزرگتر مساوی محور

Pseudocode Partition

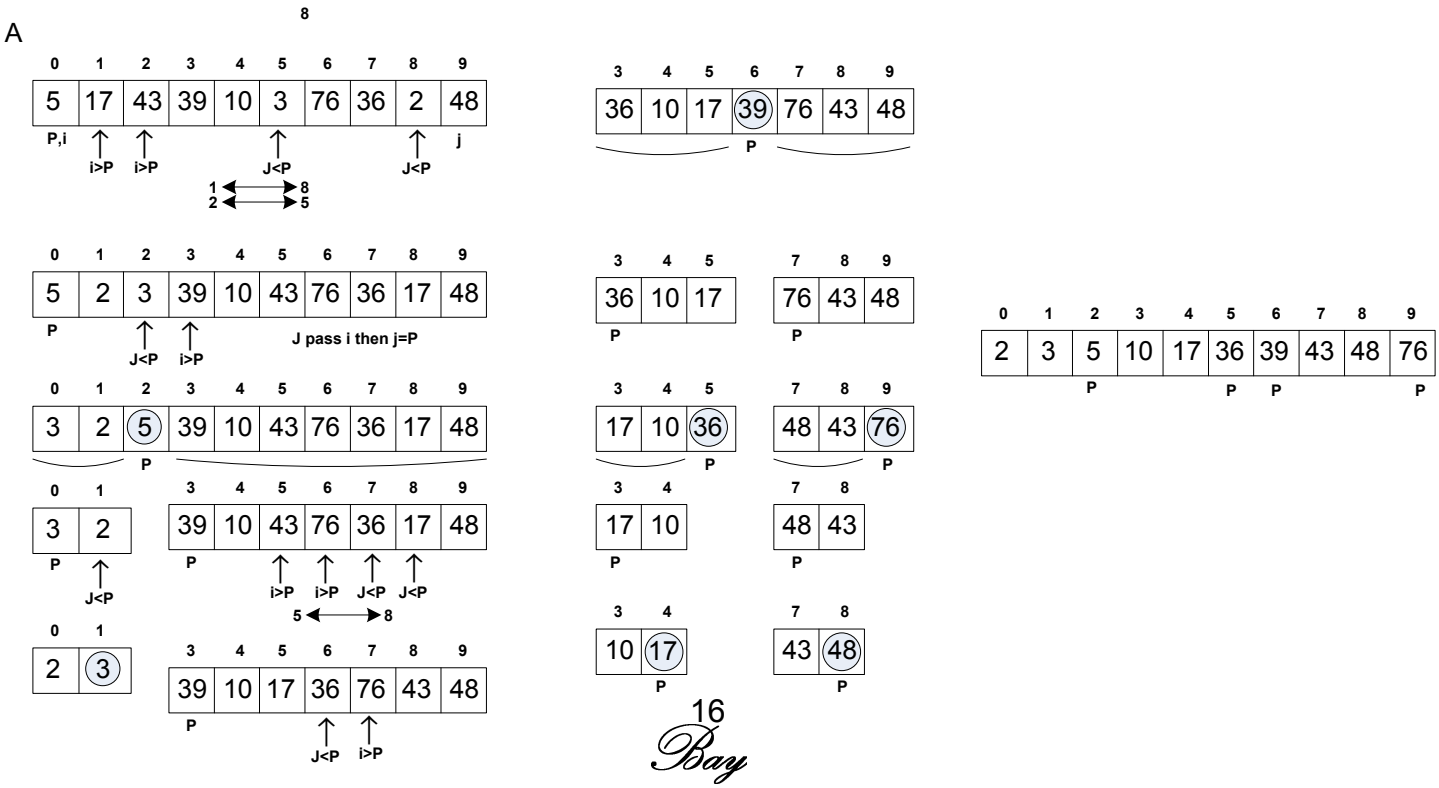
```
Partition (A,low,high)
{
P=A[low],i=low,j=high+1;
   Repeat
   {
   Repeat i++; until A[i]>=p;
   Repeat j-- ;until A[j]<p;
   If(i<j) Swap (A[i],A[j]);
   }
   Until (i>=j);
Swap (A[low],A[j]);
Return j;
}
```

Pseudocode Quick Sort

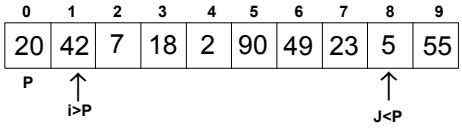
```
Quick sort (A,low,high)
{
If(low<high)
   {
   p=partition(A,low,high)
   Quick sort(A,low,p-1);
   Quick sort(A,p+1,high);
   }
}
```

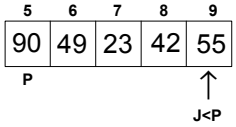
مثال: الگوریتم مرتب سازی سریع را تا سه مرحله بر روی آرایه داده شده اعمال نمائید؟

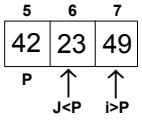
نبسمت j حرکت میکند تا از P بزرگتر شود و بلعکس



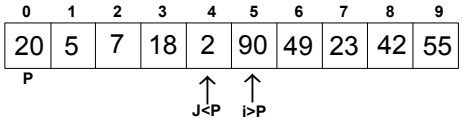
مثال: الگوریتم مرتب سازی سریع را تا شش مرحله بر روی آرایه داده شده اعمال نمائید؟

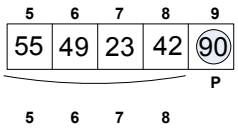


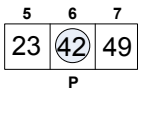




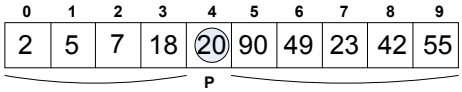
در مرحله اول عدد 20 انتخاب میشود(اولین P) تا مکان دقیق آن مشخص شود،برای مشخص شدن مکان دقیق عدد 20 خانه اندیس 1 با 8 جابجا میشود.

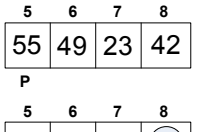






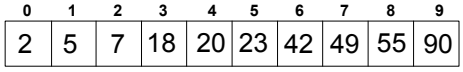
در مرحله دوم اجرای پارتیشن عدد 2 بعنوان محور انتخاب میشود تا مکان دقیق آن مشخص شود-برا ی تعیین مکان دقیق عدد 2 با توجه به اینکه جای 2 درست میباشد هیچگونه جابجایی انجام نمیشود.

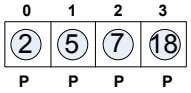


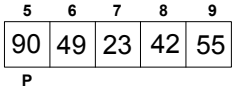


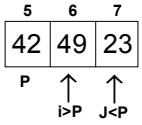
در مرحله سوم ،چهارم و پنجم نیز وقتی تابع پارتیشن اجرا میشود با توجه به sort بودن هیچگونه جابجایی صورت نمیگیرد.

در مرحله ششم اجرای این تابع مکان دقیق عدد 90مشخص میشود که برای تعیین این عدد یک جابجایی انجام میشود (خانه اندیس 5 با9)



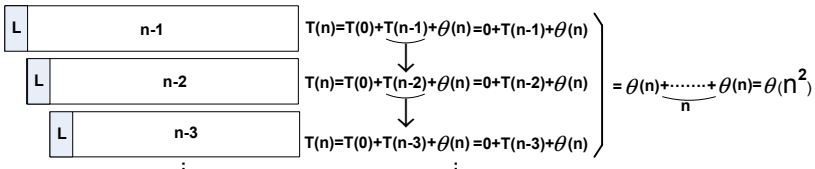






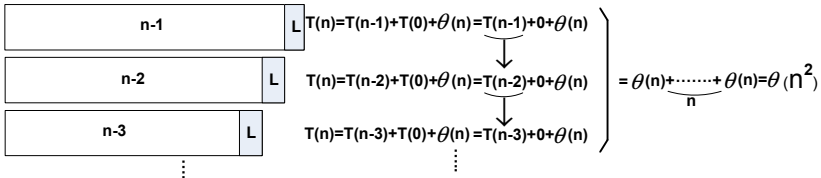
مثال: اگر درحالت کلی محور آرایه اندیس ،L فرض شود زمان اجرای الگوریتم quick-sort را بدست آورید.؟

حالت اول: اگر در هر مرحله مکان دقیق محور، خانه اول آن زیر آرایه شود. (L=1)



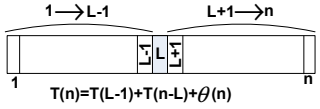
یعنی اگر آرایه صعودی باشد و بخواهیم صعودی مرتب کنیم یا نزولی باشد و بخواهیم نزولی مرتب کنیم.

حالت دوم: اگر در هر مرحله مکان دقیق محور ،خانه آخر آن زیر آرایه شود. (L= n)



یعنی اگر آرایه صعودی باشد و بخواهیم نزولی مرتب کنیم یا نزولی باشد و بخواهیم صعودی مرتب کنیم.

حالت سوم:اگر در هر مرحله مکان دقیق محور، خانه وسط آن زیر آرایه شود. ($L=\frac{n}{2}$)



1⟶L-1
(L-1)-1+1=L-1

L+1⟶n
n-(L+1)+1=n-L

1+ابتدا - انتها

فرمول کلی مرتبه زمانی الگوریتم
جستوی سریع ،در حالت متوسط

$$T(n)=\sum_{x=1}^{n}\left[T(x-1)+T(n-x)+\theta(n)\right]=n\log n$$

$T(n)=T(\frac{n}{2})+T(\frac{n}{2}+1)+\theta(n)$

$T(n)=T(\lfloor\frac{n}{2}\rfloor)+T(\lceil\frac{n}{2}+1\rceil)+\theta(n)$

ازآنجائیکه در محاسبه مرتبه زمانی ثابتها و حد پایین و بالا حذف میشوند به رابطه زیر می رسیم.

$$T(n)=2T(\frac{n}{2})+\theta(n)$$

$$\theta(n\log n)$$

در quick-sort بهترین حالت حالت سوم است و بدترین حالت ،حالت اول و دوم

5 روش حریصانه Greedy

در این روش حل مسئله همواره سعی میشود که برای حل یک مسئله ،پر ارزشترین و در عین حال مطلوبترین انتخاب را استفاده کرد.بنابراین میتوان نتیجه گرفت روش حریصانه در مسائل بهینه سازی کاربرد فراوان دارد.

در مسائلی که از طریق الگوریتم حریصانه حل میشود ممکن است شرایط به گونه ای باشد که روند حل مسئله به بن بست برسد و یا پاسخ نهایی بهینه نباشد.در الگوریتم حریصانه مجموعا سه تابع استفاده میشود که به شرح زیر است.

1-مجموعه C :مجموعه ای از کاندیدها میباشند که یک جواب از مسئله و در صورت وجود زیر مجموعه ای از آن محسوب میشود.

2-مجموعه S:مجموعه جواب است که در ابتدا تهی است و با پیشرفت حل مسئله و در صورت وجود جواب و همگرا بودن مسئله این مجموعه تکمیل میشود.

2-مجموعه S:مجموعه جواب است که در ابتدا تهی است و با پیشرفت حل مسئله و در صورت وجود جواب و همگرا بودن مسئله این مجموعه تکمیل میشود.

3- تابع select() :ورودی این تابع مجموعه C وخروجی آن بهترین عضو مجموعه C است که با شرط الگوریتم مطابقت داشته باشد.پس از تعیین یک عضو از مجموعه C تابع select آن عضو را از مجموعه C حذف میکند.

4-تابع Feasible() :پس از انتخاب عنصر x از مجموعه C ،حاصل S U{x} بعنوان ورودی به تابع feasible داده میشود این تابع بررسی میکند که آیا x میتواند به S اضافه شود یا خیر.اگر پاسخ True باشد x به S اضافه میشود.

5-تابع Solution() :این تابع مجموعه S را بعنوان ورودی دریافت میکند و بررسی میکند که آیا S یک پاسخ کامل ازمسئله است یا خیر.خروجی این تابع True یا False است.(در صورت true تابع به اتمام میرسد)

```
Function Greedy (c:set):set
s ⟵ ϕ
While (c≠ϕ and not(solution() ))
{
  x ⟵ Select(c) ;
  c ⟵ c\{x};
  If (feasible (sU{x})
  s ⟵ sU{x};
}
If (solution (s))
return s;
else
Print ("there are no solution")
```

(c:set):set مجموعه C را میگیرد و جواب را بصورت یک مجموعه میدهد.

c≠ϕ and not(solution() آیا کاندیداها خالی شدند و آیا به جواب رسیده است.

x ⟵ Select(c) ; انتخاب بهترین عضو

c ⟵ c\{x}; بهترین انتخاب از کاندیداها حذف میشود.

If (feasible (sU{x}) اگر مجموعه بدون مشکل بود(توجیه پذیر است)

If (solution (s)) آیا راه حل پیدا شده است.

الگوریتم اول:الگوریتم هافمن Huffman Code:

یکی از روشهای سریع برای فشرده سازی فایلهای متنی روش هافمن است در این روش فایل ورودی کارکتر به کاراکتر خوانده میشودو فراوانی هر کاراکتر محاسبه می شود و متناظر با هر یک از کاراکترهای موجود در الگوریتم هافمن ، یک برگ برای درخت هافمن در نظر گرفته میشود در این الگوریتم عمل زیر آنقدر انجام میشود تا به ریشه درخت برسیم.

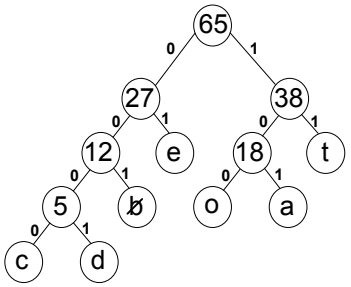
از بین برگهای موجود که دارای برچسب فراوانی کاراکترهای خود هستند دو برگ مینیمم را انتخاب میکنیم و با یکدیگر ادغام میکنیم(گره کوچکتر سمت چپ و گره بزرگتر سمت راست) و گره جدیدی ایجاد میشود که برچسب آن معادل مجموع فراوانیهای دو برگ ادغام شده است.

پس از ایجاد درخت دودویی هافمن (هر گره دو فرزند دارد) یالهای سمت راست دارای وزن 1 و یالهای سمت چپ دارای وزن 0خواهند بود.

سپس برای محاسبه کد هر کاراکتر دنباله 0،1 را از ریشه تا برگ مربوط به آن کارکتر بدست می آوریم.

مثال: با توجه به جدول کاراکتر و فراوانیهای داده شده به موارد زیر پاسخ دهید؟

char	fi	code	n
a	10	101	3
c	2	0000	4
e	15	01	2
↵	7	001	3
t	20	11	2
d	3	0001	4
o	8	100	3



101110010000=at↵c
a t ↵ c

تعداد بیتهای کل قبل از روش هافمن $(\sum fi)*8=65*8=520$

$\sum (fi*(8-n))=10*5+2*4+....+8*5=355$
تعداد بیتهای صرفه جویی شده در روش هافمن

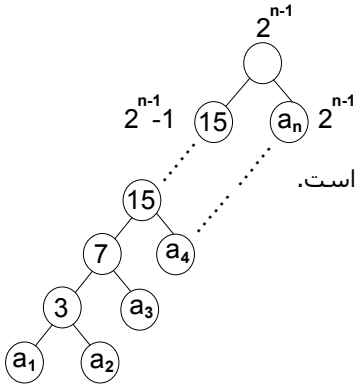
=520-355=165 تعداد بیتهای کل بعد از روش هافمن

مثال: در متنی n کاراکتر وجود دارد $(a1,a2,\ldots,an)$ که فراوانی هر a_i از رابطه 2^{i-1} بدست می آید.

الف)درخت هافمن مربوط به این متن را مشخص کنید؟

ب)طولانیترین کد هافمن مربوط به کدام کاراکتر و چیست؟

a_i	fi
a_1	1
a_2	2
a_3	4
a_4	8
⋮	⋮
a_{n-1}	2^{n-2}
a_n	2^{n-1}



$2^{n-1}-1+2^{n-1}=2(2^{n-1})-1=2^{n-1}-1$

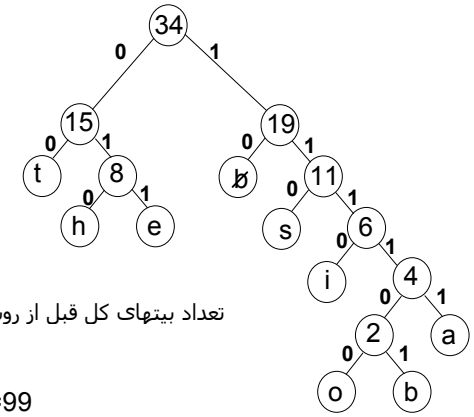
طولانیترین کد هافمن در بدترین حالت n-1 میباشد ودر این مسئله a_1 و a_2 است.

همیشه تعداد یالها یک واحد کمتر از تعداد گره ها است.

تمرین: با توجه به متن زیر حجم فایل فشرده شده چند بیت خواهد بود اگر فشرده سازی از روش کد هافمن انجام شود؟

this␣is␣the␣set␣that␣has␣to␣be␣set

char	fi	code	n
t	7	00	2
h	4	010	3
i	2	1110	4
␣	8	10	2
s	5	110	3
e	4	011	3
a	2	11111	5
b	1	111101	6
o	1	111100	6



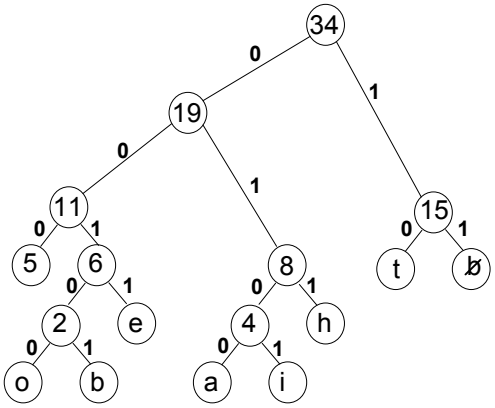
تعداد بیتهای کل قبل از روش هافمن $(\sum fi)*8=34*8=272$

$\sum (fi*(n)= 7*2+4*3+2*4+8*2+5*3+4*3+2*5+1*6+1*6=99$

تعداد بیتهای کل بعد از روش هافمن

char	fi	code	n
t	7	10	2
h	4	011	3
i	2	0101	4
␣	8	11	2
s	5	000	3
e	4	0011	4
a	2	0100	4
b	1	00100	5
o	1	00101	5

حدالمکان سعی میکنیم اعداد را از جدول انتخاب کنیم،اولویت استفاده از اعداد جدول میباشد



```
Procdure prim(v,E,T)
T_∅
Tv_{1}
While |T| < n-1 do
Select e= (u,v) with minimum cost from E such that u∈Tv and v∉Tv
if there is no any such edge then
exit
endif
add e to T
add v to Tv
repeat
if |T|<n-1 then write ('no spanning tree') endif
end.
```

مرتبه زمانی $O(|V^2|)$

الگوریتم یافتن درخت پوشای مینیمم : (MST) Minimum Spanning Tree

گراف ساده: میگوییم G(V,E) یک گراف ساده است اگر V مجموعه متناهی از رئوس و E یک مجموعه از زیر مجموعه های دو عضوی V باشد.
یال E=Edge, راس یا گره V=Vector

درخت: گرافی متصل همبند که فاقد دور است ،در یک درخت این رابطه برقرار است. |V|=|E|+1

گراف وزن دار: هرگاه یال در یک گراف برچسب در نظر بگیریم آنگاه گراف وزن دار بدست می آید.

درخت پوشا: درخت پوشای گرافی مانند G شامل همه رئوس G و V-1 یال از گراف G است ،این درخت لزوما همبند است.

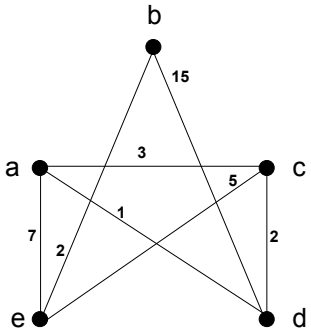
درخت پوشا مینیمم: درخت پوشایی است که مجموع وزن یالهای آن کمینه (مینیمم)باشد.به این درخت اصطلاحا MST گفته میشود.

الگوریتم Prim:

طرز کار این الگوریتم بدین شکل است که MST را در هرمرحله با ایجاد یک درخت همبند که دارای حد اقل مجموع وزن است ایجاد میکند.در این الگوریتم ابتدا یک راس دلخواه مانند A بعنوان راس مبدا انتخاب میشود ،سپس بین یالهایی که از A خارج شده است یک یال با کمترین وزن انتخاب میشود و گره مقصد آن مانند B به مجموعه گره های بازدید شده اضافه میشود و همچنین یال e که گره A را به گره B متصل کرده است به مجموعه جواب اضافه میشود.

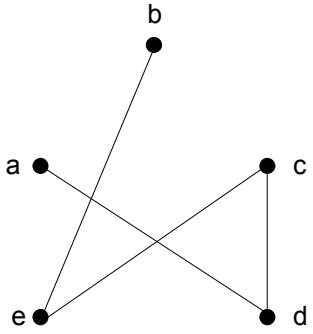
در مرحله بعد یالی که کمترین وزن را بین یالهای خروجی B ویالهای بازدید نشده A را دارد انتخاب میشود و در صورتیکه اضافه شدن این یال به درخت ،دور ایجاد نکند این یال به مجموعه جواب اضافه میشود ،این عمل تا جایی ادامه پیدا میکند که همه راسهای گراف بازدید شود.

مثال: درخت پوشای مینیمم برای گراف G به شکل زیر را طبق الگوریتم Prim بدست آورید (شروع با راس b)؟

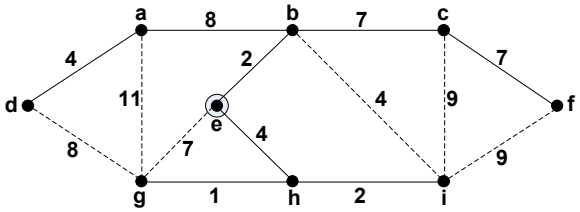


رئوس بازدید شده	یال انتخابی	راس بازدید شده جدید	ایجاد دور؟	مجموع وزن
{b}	(b,e)	{e}	F	2
{b,e}	(e,c)	{c}	F	7
{b,c,e}	(c,d)	{d}	F	9
{b,c,d,e}	(a,d)	{a}	F	10
{a,b,c,d,e}	———	———	———	(10)

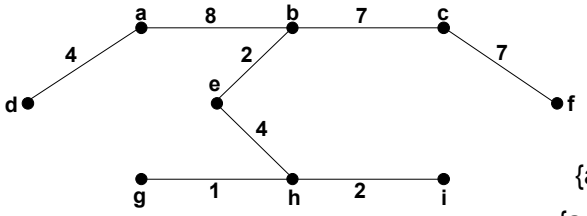
MST



تمرین: در گراف روبرو لبه های پررنگ درخت پوشای حداقل را با استفاده از الگوریتم Prim نشان میدهد.گره ای که با دایره مشخص شده است نقطه شروع الگوریتم است ،در اینحالت آخرین گره ای که به درخت پوشا متصل شده دارای چه وزنی است.؟



رئوس بازدید شده	یال انتخابی	راس بازدید شده جدید	ایجاد دور؟	مجموع وزن
{e}	(e,b)	{b}	F	2
{e,b}	(e,h)	{h}	F	6
{e,b,h}	(g,h)	{g}	F	7
{e,b,g,h}	(i,h)	{i}	F	9
{e,b,g,h,i}	(b,c)	{c}	F	16
{e,b,c,g,h,i}	(c,f)	{f}	F	23
{e,b,c,f,g,h,i}	(a,b)	{a}	F	31
{a,e,b,c,f,g,h,i}	(a,d) 4	{d}	F	35
{a,e,b,c,d,f,g,h,i}				MST



تمرین: اگر Gگراف بی جهت ، M یک MST باشد آنگاه کدام گزاره درست است.

الف) در M مسیر بین هر جفت راس V1,V2 کوتاهترین مسیر وجود دارد.

ب)همه مسیرهای موجود در M کوتاهترین مسیر هستند.

الف و ب یکی هستند و جواب هر دو غلط هستند. چون (b,i)=4 میتوانست باشد و لی در MST وزن آن 8 است .
در درخت MST اگرچه مجموع وزن یالهای حاصل مینیمم است ولی مسیر بین هر دو راس در این درخت نسبت به گراف اولیه ممکن است مینیمم نباشد.

تمرین:* مدل خاصی از الگوریتم مرتب سازی سریع بنام مرتب سازی سریع تصادفی Randomaize quick Sort وجود دارد که هر بار محور را در بازه مربوطه تصادفی انتخاب میکند ،باچه احتمالی مرتبه زمانی این الگوریتم مرتب سازی برابر$\theta(n^2)$است.

الگوریتم کروسکال: Kruskal

طرز کار این الگوریتم بدین صورت است که ابتدا همه یالهای موجود در گراف بصورت صعودی مرتب میشوند و به ترتیب به MST اضافه میشوند تا جائیکه دور در درخت MST ایجاد نشود ،برای ساخت این درخت ،الگوریتم کروسکال جنگلهای متمایز را به یکدیگر متصل میکند.

نکته: مرتبه زمانی الگوریتم کروسکال برابر O(|El logl El) میباشد. همه الگوریتمهای مرتب سازی بصورت متوسط مرتبه زمانی آنها θ(n logn) میباشد.

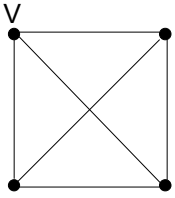
الگوریتم کروسکال بطور کلی از دو بخش تشکیل شده است.

1-مرتب سازی صعودی یالهای گراف $\theta(n\log n)=\theta(E\log E)$

$\theta(n\log n)$ ←

2-انتخاب n-1 یال با کمترین وزن که دور در درخت ایجاد نکند $\theta(n)=\theta(|V|)$

قانون گراف:



گراف پر:یعنی از هر راس به همه راسهای دیگر یک یال برقرار باشد. $E=\frac{V(V-1)}{2}=6$

تعداد یالهای یک گراف همیشه در این معادله صدق میکند. $|V|-1 \leqslant |E| \leqslant \frac{|V|(|V|-1)}{2}$

(گراف خلوت: $|V|-1$ ، درخت پوشا مینیمم)

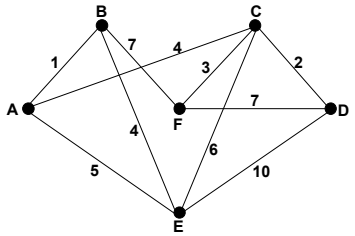
Prim	$O(V^2)$
Kruskal	$O(E\log E)$

گراف خلوت $|E| \simeq |V|$ →
- Prim $O(V^2)$
- Kruskal $O(V\log V)$

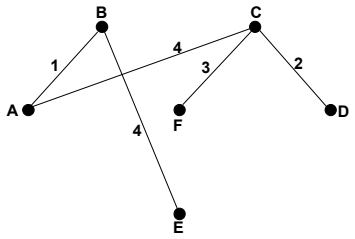
گراف پر $|E| \simeq |V|$ →
- Prim $O(V^2)$
- Kruskal $O(V^2\log V^2)$

در حالت گراف خلوت الگوریتم Kruskal بهتر است و درحالت گراف پر الگوریتم Primبهتر است.

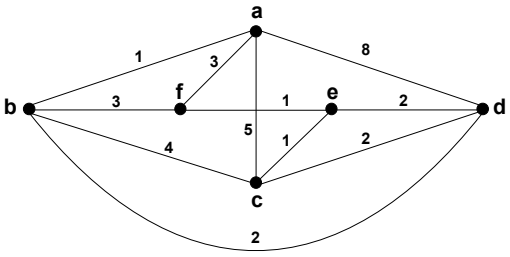
تمرین: وزن مینیمم درخت MST مربوط به گراف داده شده را محاسبه کنید؟(به روش کروسکال)



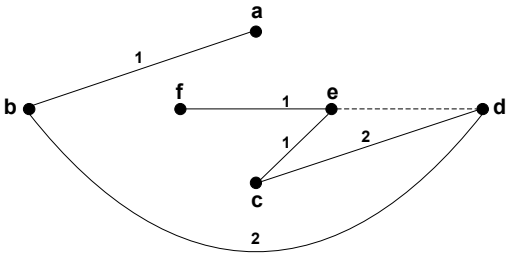
رئوس بازدید شده	یال انتخابی	n	رائوس بازدید شده جدید	ایجاد دور؟	مجموع وزن
—	(A,B)	1	{A,B}	F	1
{A,B}	(C,D)	2	{C,D}	F	3
{A,B,C,D}	(C,F)	3	{F}	F	6
{A,B,C,D,F}	(A,C)	4	—	F	10
{A,B,C,D,F}	(B,E)	5	{E}	F	14
{A,B,C,D,E,F}					MST



تمرین: وزن مینیمم درخت MST مربوط به گراف داده شده را محاسبه کنید؟(به روش کروسکال)



رئوس بازدید شده	یال انتخابی	n	رائوس بازدید شده جدید	ایجاد دور؟	مجموع وزن
—	(a,b)	1	{a,b}	F	1
{a,b}	(c,e)	2	{c,e}	F	2
{a,b,c,e}	(e,f)	3	{f}	F	3
{a,b,c,e,f}	(c,d)	4	{d}	F	5
{a,b,c,d,e,f}	(e,d)	—	—	T	—
{a,b,c,d,e,f}	(b,d)	5	—	F	7
					MST



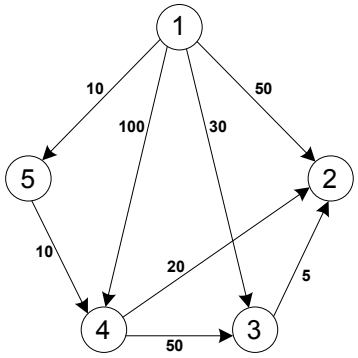
الگوریتم دایجسترا: Dijekstra

(الگوریتم محاسبه کوتاهترین مسیر از گره مبداء به همه گره ها)

فرض کنیم میخواهیم در یک گراف جهت دار که همه یالهای آن با اعداد نامنفی برچسب خورده اند،فاصله یک گره مشخص را که گره مبداء نامیده میشود در بهترین حالت با سایر گره ها محاسبه کنیم. پیاد سازی این الگوریتم با استفاده از ماتریس مجاورت انجام میشود که درایه های ماتریس برچسبهای مربوط به یالهای گراف هستند.

```
Procedure Dijekstra (L1 [1...n,1...n]:array [2...n]
Array D[2...n]
{initialization}
C←{2,3,….,n}
For i=2 to n Do
D[i]=L1[1,i]
{Greedy loop}
Repeat n- 2 times
{
V←some elements of C that minimizing D[V]
C←C\{V}
S← S U{V}
For each W∈ C  Do
D[W]←min{d[w],d[v]+L [V,W]}
}
Return D;
```

تمرین: الگوریتم دایجسترا را برای گراف زیر پیاده سازی کنید؟



	1	2	3	4	5
1	0	50	30	100	10
2	∞	0	∞	∞	∞
3	∞	5	0	∞	∞
4	∞	20	0	0	∞
5	∞	∞	∞	10	0

انتخاب V :راسی از C انتخاب میشود که مقدار آن در C مینیمم است.

نقش واسط دارد

C ← [2,3,..n] L1[1,i]

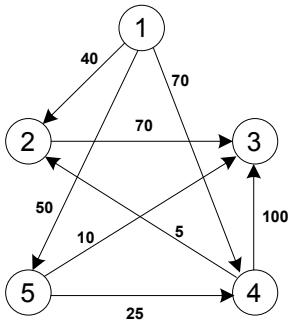
Step	V	C	D[2...n]
1		{2,3,4,5}	[50,30,100,10]
2	5	{2,3,4}	[50,30,20,10] *
3	4	{2,3}	[50,30,20,10]
4	3	{2}	[35,30,20,10] *
5	2	{}	[35,30,20,10]

$D[2]=\min\{D[2],D[5]+L_1[5,2]\}$

از 1 به 2 50 10 ∞ ∞ 50

جواب: [35,30,20,10] کوتاهترین فاصله از گره 1 به 5،4،3،2

تمرین: الگوریتم دایجسترا را برای گراف زیر پیاده سازی کنید؟و تعیین کنید که چگونه میتوان علاوه بر وزن کوتاهترین مسیر ،گره های کوتاهترین مسیر را مشخص کرد؟(یعنی یک خط به شبه کد اضافه شود تا شماره واسط (V) یا واسطها را محاسبه نماید.)



	1	2	3	4	5
1	0	40	∞	70	50
2	∞	0	70	∞	∞
3	∞	∞	0	∞	∞
4	∞	5	100	0	∞
5	∞	∞	10	25	0

Step	V	C	D[2...n]
1		{2,3,4,5}	[40,∞,70,50]
2	2	{3,4,5}	[40,110,70,50] *
3	5	{3,4}	[40,60,70,50] *
4	4	{3}	[40,60,70,50]
5	3	{}	[40,60,70,50]

جواب: [40,60,70,50]

الگوریتم کوله پشتی غیر صفر و یک: None 0..1 Knap Sack

تعدادی شی و یک کوله پشتی موجود است ،هر شی i دارای وزن W_i و ارزش P_i است.ظرفیت کوله پشتی برابر M است ،میخواهیم اشیاء را به گونه ای انتخاب کنیم که اولا از ظرفیت کوله پشتی بیشتر نشود و دوما ارزش موجود در کوله پشتی حداکثر شود .راه حل این مسئله بدین صورت است که چون وزن شی در انتخاب شی تاثیر منفی و ارزش شی در انتخاب آن تاثیر مثبت دارد ،بهترین حالت اینست که اشیاء را به ترتیب نزولی و بر اساس نسبت $\frac{P_i}{W_i}$ انتخاب نمود.

ابتدا اشیاء بر اساس $\frac{P_i}{W_i}$ و بصورت نزولی مرتب میشوند.سپس این اشیاء بطور کامل و به ترتیب به کوله پشتی اضافه میشوند تا جائیکه با انتخاب یک شی مجموع وزن از کوله پشتی بیشتر شود ،آنگاه کسری از آخرین شی به کوله پشتی اضافه میشود و اجراء به پایان میرسد.

```
Knap Sack (P,W,M,n,X)
{
PW-Sort (P<W,n);          → تابع مرتب سازی نزولی بر اساس نسبت ارزش به وزن
For (i=0 ;i<n;i++)
X[i]=0;                   → کل آرایه x را صفر میکند
rc=M                      → مقدار دهی اولیه در rc ریخته میشود.
For (i=0;i<n;i++)
{
If (W[i]>rc) Break ;
X[i]=1;                   → یعنی x کامل انتخاب شده است.
rc-=W{i};
}
If (i<=n-1)

X[i]= rc / W{i}
}
```

P آرایه ارزش اشیاء

W آرایه وزن اشیاء

M ظرفیت کوله پشتی

n تعداد اشیاء

X آرایه خروجی

rc ظرفیت باقیمانده remaining content

مثال: با توجه به جدول اشیاء داده شده مراحل اجرای الگوریتم کوله پشتی صفر و یک را تعیین کنید؟(ظرفیت کوله M=50)

در ابتدا سه ستون اول را داریم.

Object	P	W	$\frac{P}{W}$	PW-Sort
1	20	5	4	1
2	30	15	2	3
3	5	5	1	4
4	50	1	50	0
5	50	20	2.5	2
6	30	50	0.6	5

W [1,5,20,15,5,50] P [50,20,50,30,5,30] X [0,0,0,0,0,0]

1) W_0 =1 , rc=50 ⟶ { x[0]=1 ,rc=49}
2) W_1 =5 rc=49 ⟶ { x[1]=1 ,rc=44}
3) W_2 =20 rc=44 ⟶ { x[2]=1 ,rc=24}
4) W_3 =15 rc=24 ⟶ { x[3]=1 ,rc=9}
5) W_4 =5 rc=9 ⟶ { x[4]=1 ,rc=4}
6) W_5 =50 rc=4 ⟶ { x[5]=$\frac{rc}{w[i]}=\frac{4}{50}$=0.08,rc=0}

X[i] نشان دهنده اینست که چه کسری از W انتخاب میشود. X[i]=0 یعنی انتخاب نمیشود و X[i]=1 یعنی کامل انتخاب میشود.

جواب X [1,1,1,1,1,0.08]

$$P_{Total} = \sum_{i=0}^{n-1} x[i].p[i]$$

ارزش کوله P=50+20+50+30+5+30*0.08=157.4

مثال: با توجه به جدول اشیاء داده شده مراحل اجرای الگوریتم کوله پشتی صفر و یک را تعیین کنید؟(ظرفیت کوله M=50)

در ابتدا سه ستون اول را داریم.

Object	P	W	$\frac{P}{W}$	PW-Sort
1	20	10	2	3
2	30	15	2	2
3	5	5	1	4
4	50	1	50	0
5	40	20	2	1
6	30	50	0.6	5

W [1,20,15,10,5,50] P [50,40,30,20,5,30] X [0,0,0,0,0,0]

1) W_0 =1 , rc=50 ⟶ { x[0]=1 ,rc=49}
2) W_1 =20 rc=49 ⟶ { x[1]=1 ,rc=29}
3) W_2 =15 rc=29 ⟶ { x[2]=1 ,rc=14}
4) W_3 =10 rc=14 ⟶ { x[3]=1 ,rc=4}
5) W_4 =5 rc=4 ⟶ { x[4]=$\frac{rc}{w[i]}=\frac{4}{5}$=0.8,rc=0}

اگر P/W ها یکسان بود در مرحله اول بر اساس P بیشتر مرتب میکنیم و اگر P ها نیز مساوی بود بر اساس W کمتر PW-Sort را مرتب سازی میکنیم.

جواب X [1,1,1,1,0.8,0]

ارزش کوله P=50+40+30+20+5*0.8+0=144

مثال: کوله پشتی با وزن 100 کیلوگرم و شش شی به شرح زیر وجود دارد ارزش نهایی در کوله پشتی را محاسبه کنید؟

Object	P	W	$\frac{P}{W}$	PW-Sort
1	30	20	1.5	3
2	5	30	0.16	5
3	50	10	5	0
4	40	20	2	2
5	25	5	5	1
6	10	20	0.5	4

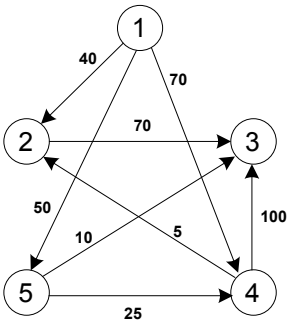
W [10,5,20,20,20,30] P [50,25,40,30,10,5] X [0,0,0,0,0,0]

1) W_0 =10 , rc=100 ⟶ { x[0]=1 ,rc=90}
2) W_1 =5 , rc=90 ⟶ { x[1]=1 ,rc=85}
3) W_2 =20 , rc=85 ⟶ { x[2]=1 ,rc=65}
4) W_3 =20 , rc=65 ⟶ { x[3]=1 ,rc=45}
5) W_4 =20 , rc=45 ⟶ { x[4]=1 ,rc=25}
6) W_5 =30 , rc=25 ⟶ { x[5]=$\frac{rc}{w[i]}=\frac{25}{30}$=0.83,rc=0}

جواب X [1,1,1,1,1,0.83]

ارزش کوله P=50+25+40+30+10+5*0.83=159.15

تمرین: الگوریتم دایجسترا را برای گراف زیر پیاده سازی کنید؟و تعیین کنید که چگونه میتوان علاوه بر وزن کوتاهترین مسیر ،گره های کوتاهترین مسیر را مشخص کرد؟(یعنی یک خط به شبه کد اضافه شود تا شماره واسط (V) یا واسطها را محاسبه نماید.)



	1	2	3	4	5
1	0	40	∞	70	50
2	∞	0	70	∞	∞
3	∞	∞	0	∞	∞
4	∞	5	100	0	∞
5	∞	∞	10	25	0

Step	V	C	D[2...n]
1		{2,3,4,5}	[40,∞,70,50]
2	2	{3,4,5}	[40,110*,70,50]
3	5	{3,4}	[40,60*,70,50]
4	4	{3}	[40,60,70,50]
5	3	{}	[40,60,70,50]

جواب: [40,60,70,50]

Procedure Dijekstra (L_1 [1...n,1...n]:array [2...n]
Array D[2...n]
{initialization}
C⟵ {2,3,….,n}
For i=2 to n Do
D[i]=L [1,i]
{Greedy loop}
Repeat n- 2 times
{
V⟵ some elements of C that minimizing D[V]
C⟵ C\{V}
S⟵ S U{V}
For each W C Do
D[W]⟵ min{d[w],d[v]+L [V,W]}

pick the vertex, V, in D[V] with the shortest path to S
add V to F
F in First level is the empty array

}
Return D;

مثال: کوله پشتی با وزن 100 کیلوگرم و شش شی به شرح زیر وجود دارد ارزش نهایی در کوله پشتی را محاسبه کنید؟

Object	P	W	$\frac{P}{W}$	PW-Sort
1	30	20	1.5	3
2	5	30	0.16	5
3	50	10	5	0
4	40	20	2	2
5	25	5	5	1
6	10	20	0.5	4

W [10,5,20,20,20,30] P [50,25,40,30,10,5]

1) W_0 =10 , rc=100 ⟶ { x[0]=1 ,rc=90}
2) W_1 =5 , rc=90 ⟶ { x[1]=1 ,rc=85}
3) W_2 =20 , rc=85 ⟶ { x[2]=1 ,rc=65}
4) W_3 =20 , rc=65 ⟶ { x[3]=1 ,rc=45}
5) W_4 =20 , rc=45 ⟶ { x[4]=1 ,rc=25}
6) W_5 =30 , rc=25 ⟶ { x[5]=$\frac{rc}{w[i]}=\frac{25}{30}$=0.83,rc=0}

جواب X [1,1,1,1,1,0.83]

ارزش کوله P=50+25+40+30+10+5*0.83=159.15

برنامه نویسی پویا: Dynamic Programming

در روش D&P مسائل بصورت از پایین به بالا حل میشود (Bottem-Up) تفاوت روش DP و روش تقسیم و حل در اینست که در روش تقسیم و حل یک مسئله به زیر مسائل کوچکتر تقسیم میشود و پس از حل این زیر مسائل پاسخ ها ادغام شده تا پاسخ نهایی حاصل شود.ولی در روش DP از ابتدا به بررسی زیر مسائل میپردازیم و با ترکیب پاسخ آنها به مسئله اصلی میرسیم.

اگر برای بدست آوردن یک مسئله نیاز به حل یک زیر مسوله به تعداد دفعات مشخصی باشیم کارائی الگوریتم تقسیم و حل کاهش پیدا میکند ،راه حل پاسخ به این مسائل به این صورت است که ابتدا پاسخ زیر مسائل را در حافظه موقت نگهداری کنیم و از پاسخهای ذخیره شده در سطوح بالاتر استفاده کنیم.

نکته:اصل اساسی موجود در حل مسائل DP اصل بهینگی است ،این اصل بیانگر این مطلب است که برای بدست آوردن پاسخ بینه یک مسئله لزوما نتایج حاصل از زیر مسائل باید بهینه باشد.

در این فصل به به بررسی الگوریتمهای زیر میپردازیم.

1-ضرب زنجیره ای ماتریسها

2-محاسبه $\binom{n}{k}$ ترکیب K از n

3-کوله پشتی صفر ویک

4-یافتن زیر رشته مشترک

5-الگوریتم فروشنده دوره گر

الگوریتم ضرب زنجیره ای ماتریسها:

فرض کنید ماتریس M_1 تا M_n موجود است و قرار است $M_1M_2....M_n$ محاسبه شود.

نکته قابل توجه در ضرب ماتریسها اینست که در ضرب زنجیره ای ماتریسها حداقل تعداد ضرب حائز اهمیت است.در ضرب دو ماتریس دو ویژه گی باید رعایت شود.

$$M_1.M_2.M_3=\begin{cases}(M_1.M_2)M_3\\ M_1(M_2.M_3)\end{cases}$$

1 - ضرب ماتریسها شرکت پذیر است.

2 - دو ماتریسی که قرار است در هم ضرب شوند باید ضرب پذیر باشند.بعبارت دیگر $A_{i\times j}, B_{j\times k}$ ضرب پذیر هستند زیرا تعداد ستون A و تعداد سطر B با هم برابرند و حاصل ضرب این دو ماتریس معادل $C_{i\times k}$ خواهد بود. که هزینه ضربی(تعداد ضرب انجام شده) $i\times j\times k$ دارد.

نکته:تعداد حالات ممکن برای ضرب n ماتریس برابر nامین عدد کاتالان است که از رابطه زیر بدست می آید.

$$\begin{cases}T(n)=\sum_{i=1}^{n-1}T(i).T(n-i)\\ T(1)=1\end{cases}$$

مثال: محاسبه تعداد حالات پرانتزگذاری برای چهار ماتریس بعنوان مثال $A_{2\times10}\ B_{10\times5}\ C_{5\times6}\ D_{6\times20}$

$$T(n)=\sum_{i=1}^{n-1}T(i).T(n-i)\longrightarrow\begin{cases}T(1)=1\\ T(2)=\{T(1).T(1)=1\\ T(3)=\begin{cases}T(1).T(2)=1\\ +\\ T(2).T(1)=1\end{cases}=2\\ T(4)=\begin{cases}T(1).T(3)=2\\ +\\ T(2).T(2)=1\\ +\\ T(3).T(1)=2\end{cases}=5\end{cases}$$

یعنی وقتی 4 ماتریس قابل ضرب داشته باشیم به تعداد 5 حالت پرانتزگذاری داریم

((AB)C)D ((AB)(CD))

(A(BC))D A((BC)D) A(B(CD))

فرض میکنیم ماتریس M_i دارای ابعاد $(i=1....n)$ $r_{i-1}\times r_i$ و فرض میکنیم M_{ij}حداقل تعداد ضرب برای محاسبه $M_i\times M_{i+1}\times\cdots M_j$ است،در شرایط خاص اگر $i=j\longrightarrow M_{ij}=0$ (در حقیقت مساوی M_{ii} میشود که مساوی صفر میباشد.)

بطور کلی M_{ij} از این رابطه استفاده میشود و با توجه به متغییر بودن K مقادیر M_{ij}مختلف خواهد بود.

$M_i=r_{i-1}\times r_i\ ,\ M_{i+1}=r_i\times r_{i+1}\ ,\ M_{k+1}=r_k\times r_{k+1}$

$M_{ik}=M_i\times M_{i+1}\times\cdots\times M_{k-1}\times M_k$ (هزینه ضرب ماتریسهای M_{ik})

$M_{(k+1)j}=M_{k+1}\times M_{k+2}\times\cdots\times M_{j-1}\times M_j$ (هزینه ضرب ماتریسهای $M_{(k+1)j}$)

$$M_{ij}=\min(m_{ik}+m_{(k+1)j}+r_{i-1}\times r_k\times r_j)\quad i\le k\le j-1$$

$r_{i-1}\times r_k\times r_j=$(هزینه ضرب ماتریسهای M_{ik} ,$M_{(k+1)j}$)

مثال: چهار ماتریس با ابعاد داده شده را در نظر بگیرید،حداقل هزینه ضرب بهمراه بهترین حالت پرانتزگذاری در ضرب زنجره ای این 4 ماتریس را مشخص کنید (به روش DP)

$$\overbrace{A_{10\times2}}^{M_1}\ \overbrace{B_{2\times25}}^{M_2}\ \overbrace{C_{25\times3}}^{M_3}\ \overbrace{D_{3\times4}}^{M_4}$$

$$M_{ij}=\min(m_{ik}+m_{(k+1)j}+r_{i-1}\times r_k\times r_j)$$
$$i\leqslant k\leqslant j-1$$

$$\begin{cases} M_{ii}=M_{11}=M_{22}=M_{33}=M_{44}=0 \end{cases}$$

$1\leqslant k\leqslant 2-1$,k=1: $M_{12}=\min(m_{11}+m_{22}+10*2*25)=500$ (AB)

$2\leqslant k\leqslant 3-1$,k=2: $M_{23}=\min(m_{22}+m_{33}+2*25*3)=150$ * (BC)

$3\leqslant k\leqslant 4-1$,k=3: $M_{34}=\min(m_{33}+m_{44}+25*3*4)=300$ (CD)

$1\leqslant k\leqslant 3-1$:

- ,k=1: $\min(m_{11}+m_{23}+10*2*3)=210$ * A(BC)
- ,k=2: $\min(m_{12}+m_{33}+10*25*3)=1250$ (AB)C

$M_{13}=$

$2\leqslant k\leqslant 4-1$:

$M_{24}=$

- ,k=2: $\min(m_{22}+m_{34}+2*25*4)=500$ B(CD)
- ,k=3: $\min(m_{23}+m_{44}+2*3*4)=174$ * (BC)D

	1	2	3	4	
1	0	500	210	254	
2		0	154	174	
3			0	300	
4				0	

$1\leqslant k\leqslant 4-1$:

$M_{14}=$

- ,k=1: $\min(m_{11}+m_{24}+10*2*4)=254$ * A((BC)D) ← بهترین حالت پرانتزگذاری با حداقل هزینه
- ,k=2: $\min(m_{12}+m_{34}+10*25*4)=1800$ (AB)(CD)
- ,k=3: $\min(m_{13}+m_{44}+10*3*4)=330$ ((ABC)D)

الگوریتم محاسبه K شی از n شی: $\binom{n}{k}$

فرمول کلی محاسبه تعداد حالات انتخاب K مولفه از n مولفه بصورت زیر است.

$$\binom{n}{k}=\frac{n!}{k!(n-k)!} \longrightarrow \binom{5}{2}=10$$

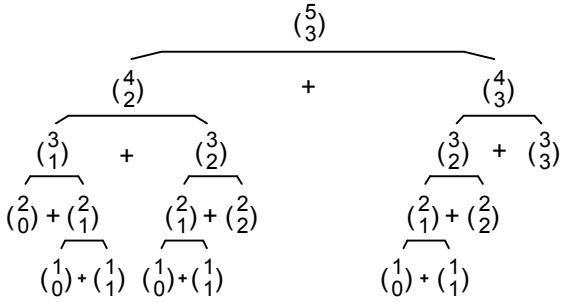
در روش تقسیم و حل برای محاسبه K حالت از n حالت از فرمول بازگشتی زیر استفاده میشود.

$$\binom{n}{k}=\begin{cases} 1 & \text{If } n=k \text{ or } k=0 \\ \binom{n-1}{k-1}+\binom{n-1}{k} & \text{Else} \end{cases}$$

در اینحالت درخت را ادامه نمیدهیم $\binom{1}{0}$ $\binom{3}{3}$

مرتبه زمانی $O(2^k)$

در سئوالاتی که فراخوانی بازگشتی آن با هم اشتراک دارند بهتر است از الگوریتم DP استفده شود.



بعنوان مثال محاسبه ترکیب $\binom{5}{3}$ در روش تقسیم و حل در قالب درخت بازگشتی زیر نمایش داده شده است ،در این درخت بازگشتی بعضی از فراخوانیها بصورت تکراری محاسبه شده اند.که همین امر باعث افزایش مرتبه زمانی حل این مسئله تا حد نهایی شده است.
با استفاده از روش DP مرتبه زمانی تا حد چند جمله ای کاهش می یابد.

الگوریتم ترکیب به روش DP

```
Function f(int n,int k)
{
Int b[n][k];
Int I,j;
For (j=0;j<min(i,k);j++)
{
If (j==i || j==0)
B[i][j]=1
Else
B[i][j]=B[i-1][j-1]+B[i-1][j];
}
Return B;
}
```

n=5,k=3 $\binom{5}{3}$

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

2=عدد بالا+عدد سمت چپ

$\binom{5}{3}$

i	0	1	2		k-1	k	k+1	k+2		n
تعداد دفعات حلقه j	1	2	3		k	k+1	k+1	k+1		k+1

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

θ(nk)

مرتبه زمانی

محاسبه زمان اجرا ترکیب $\binom{5}{3}$ به روش DP

الگوریتم تعیین زیر رشته مشترک: Longest Common Sub String (LCS)

اگر $X=x_1 x_2 x.....x_n$ یک رشته باشد آنگاه $x_i x_j x_k.....x_m$ یک زیر رشته از X است اگر $i<j<k< ....n$ باشد.

مثال یک زیر رشته از X X=ABCCBADACD ⟶ ACCAD

هدف از حل این مسئله محاسبه طولانیترین زیر رشته مشترک بین دو رشته X,Y است.

فرض میکنیم $X=x_1 x_2.....x_n$ و $Y=y_1 y_2.....y_m$ دو رشته ورودی هستند.

آنگاه هدف اصلی محاسبه C_{nm} است که بیانگر طولانیترین زیر رشته مشترک بین رشته n کاراکتری X و m کاراکتری Y است.

میتوان اینگونه ادعا کرد که $C_{j0}=0, C_{i0}=0$ برابر صفر است.در حالت کلی مقدار C_{ij} از رابطه زیر بدست می آید.

$$C_{ij}=C[i][j]=\begin{cases}0 & \text{If } i=0 \text{ or } j=0\\ C[i-1][j-1]+1 & \text{If } X_i=Y_j\\ \max\begin{cases}C[i-1][j]\\ C[i][j-1]\end{cases} & \text{If } X_i \neq Y\end{cases}$$

	j-1	j
i-1	C[i-1][j-1]	C[i-1][j]
i	C[i][j-1]	C[i][j]

مثال: اگر X=ABACBBA,Y=BCACBA باشد آنگاه LCS(x,y) را محاسبه کنید؟

برای محاسبه به یک ماتریس $(|X|+1)(|Y|+1)$ نیاز داریم.

فضای اشغالی

		B	C	A	C	B	A
	0	1	2	3	4	5	6
0	←0	←0	←0	←0	←0	←0	←0
A 1	↑0	←0	←0	1	←1	←1	1
B 2	↑0	1	←1	←1	←1	2	←2
A 3	↑0	↑1	←1	2	←2	←2	3
C 4	↑0	↑1	2	←2	3	←3	←3
B 5	↑0	1	↑2	←2	↑3	4	←4
B 6	↑0	1	↑2	←2	↑3	4	←4
A 7	↑0	↑1	↑2	3	←3	↑4	(5)

در صورت مساوی بودن دو کاراکتر در سطر و ستون از فلش مورب استفاده میکنیم و یک واحد به خانه اشاره شده در فلش مورب اضافه میکنیم.و نتیجه را در خانه سطر و ستون قرار میدهیم.

در صورت نامساوی بودن دو کاراکتر سطر و ستون ماکزیمم خانه بالایی و خانه سمت چپ را محاسبه میکنیم و نتیجه را در آن خانه قرار میدهیم و جهت فلش بسمت عدد بزرگتر میباشد ،در صورتیکه هر دو خانه اشاره شده با هم برابر باشند فلش را به سمت چپ نماد گذاری میکنیم.

LCS=5

رشته مشترک= BACBA

الگوریتم کوله پشتی صفر و یک:

روش حریصانه که در برنامه نویسی پویا بکار گرفته شده است اگر قرار باشد در مسئله کوله پشتی صفر و یک نیز استفاده شود جواب بهینه تولید نمیکند ،با استفاده از روش DP و بکارگیری یک ماتریس کمکی که در آن هر سطر برای یک شی و هر ستون متناظر با یکی از وزنهای 0 تا M است،مسئله به حالت بهینه تبدیل میشود در این روش ماتریس کمکی را به شکل V[0.....n,0.....M] در نظر میگیریم.

یک درایه از ماتریس V به شکل V_{ij} حداکثر ارزشی است که میتوان با توجه به وزن اشیاء در کوله پشتی قرار داد و از آنجائیکه $0<= j<= M$ قابل تغییر است و اشیاء قابل انتخاب همان اشیائی هستند که از 1 تا n شماره گذاری شده اند جواب نهایی مسئله در درایه V[n.m] میباشد.

برای محاسبه V_{ij} از فرمول پویای زیر استفاده میکنیم.

$$V[i,j]=\max(V[i-1,j], V[i-1,j-W_i]+P_i)$$

$$V[i,j]=\max\left(V[\,i-1,j\,],V[\,i-1,j-W_i\,]+P_i\,\right)$$

i ⟵ تعداد اشیاء بکاررفته در پر کردن کوله

j ⟵ حداکثر وزن کوله

V[i- j] ⟵ بیشترین ارزش برای پر کردن کوله ای به وزن j توسط i شی اول

V[i-1,j] ⟵ به این مفهوم است که کوله پشتی با i-1 شی قبل پر شود و شی i ام انتخاب نشود.

$V[\,i-1,j-w_i]+p_i$ ⟵ به این مفهوم که برای پر کردن کوله از شی i ام استفاده میشود،بنابراین بقیه وزن (ظرفیت) کوله که معادل $j-w_i$ است توسط i-1 شی قبل پر میشود و همچنین ارزش شی j ام که p_i است به نتیجه اضافه میشود.

در پر کردن جدول پویای کوله پشتی صفر و یک رابطه زیر برقرار است.

1) $\forall_i$ $V[\,i\,,0\,]=0$

2) $\forall_j$ $j\geq 0$ $V[\,0\,,j\,]=0$

3) $\forall_{i,j}$ $j\leq 0$ $V[\,i\,,j\,]=-\infty$

مثال: برای پر کردن کوله ای با ظرفیت M=11 و اشیائی با مشخصات زیر بیشترین ارزش بدست آمده از پر کردن کوله چیست؟

$Object_i$	W_i	P_i
1	1	1
2	2	6
3	5	18
4	6	22
5	7	28

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1 شی اول	0	1	1	1	1	1	1	1	1	1	1	1
2 شی اول تا دوم	0	1	6	7	7	7	7	7	7	7	7	7
3 شی اول تا سوم	0	1	6	7	7	18	19	24	25	25	25	25
4 شی اول تا چهارم	0	1	6	7	7	18	22	24	28	29	29	40
5 شی اول تا پنجم	0	1	6	7	7	18	22	28	29	34	35	(40)

باید جمع W_i تا مجموعه شی مورد نظر بیشترین P_i را داشته باشد.

$\sum P_i$

V[1,1]=max (V[0,1],V[0,1-1]+1=1

V[2,2]=max (V[1,2],V[1,2-2]+6=6