می‌بینیم که این کار زمان بر است که برای بدست آوردن پیچیدگی، به ازای سایزهای متفاوت، شکل رسم کنیم و در نهایت تشخیص دهیم که از چه مرتبه‌ای است پس نیاز به روشی داریم که نیاز به آزمایش کردن نباشد.

## روشهای تحلیل الگوریتم

الگوریتم ← ترتیبی
← بازگشتی

### تحلیل الگوریتم ترتیبی:

فرض: اجرای هر دستور به زمان ثابتی نیاز دارد، بنابراین اگر تعداد ثابتی دستور ساده پشت سرهم باشند در مجموع زمانی ثابت می‌گیرند ($O(1)$)

```
int x,y;
cin>>x>>y;
int sum=x+y;
cout<<sum;
```

$O(1)$

در تحلیل الگوریتمهای ترتیبی به دستوراتی توجه می‌کنیم که تعداد دفعات اجرای آنها وابسته به ورودی هستند

* اگر تعدادی دستور داخل حلقه باشند زمان اجرا، حاصلضرب تعداد تکرار در زمان اجرای دستورات خواهد بود.

* اگر برنامه شامل ساختار if و else باشد که هرکدام زمانهای $T_1$ و $T_2$ را داشته باشند $T_1(n) = O(f_1(n))$ و $T_2(n) = O(f_2(n))$

در این صورت $T(n) = O(\max\{f_1(n), f_2(n)\})$

* زمان کل برنام برابر حاصل جمع تک برنام هاست .

مثال :

```
int f(int ar[], int n)
{  int s=0;   ⟶ O(1)
   s+=7;      ⟶ O(1)
   return s;  ⟶ O(1)
}
```

⇒ دارای مرتبه زمانی O(1) است

$O(1)+O(1)+O(1) = c_1+c_2+c_3 = C \Rightarrow O(1)$

مثال :

```
int sum(int ar[], int n)
{  int s=0;   ⟶ O(1)
   for(int i=1; i<=n; i++)
      s+=ar[i]; ⟶ O(1)
   retur s;  ⟶ O(1)
}
```

این دستور وابستگی به n ندارد

حلقه for چند بار تکرار می شود ؟ n بار

رشد بیشتری دارد

$O(1) + n \times O(1) + O(1) \Rightarrow O(n)$ ✓

تعداد تکرار حلقه

مرتبه زمانی دستور داخل حلقه

تابع sum دارای مرتبه زمانی (مرتبه پیچیدگی) $O(n)$ است

مثال: برنامه زیر شامل سه تکه برنامه مستقل از هم است

```
int f( ) {
  int i, j=1, n;
  cin>>n;  ⟶ O(1)

  for(i=1; i<=n; i++)
    {
      ---   O(1)
    }

  while( i<n)
    {
      ---   O(1)
    }

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

}
```

$O(1)+n\times O(1)+n\times O(1)+n^2\times O(1) = O(1)+2n\times O(1)+n^2\times O(1)$

با بیشترین رشد

$\Rightarrow O(n^2)$

پیچیدگی زمانی این برنامه $O(n^2)$ است

مثال:

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

$\Rightarrow O(mn)$

$\longrightarrow O(1)$

روش دوم:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} 1 = \sum_{i=1}^{m} n = n\sum_{i=1}^{m} 1 = n \times m$$

$\Rightarrow O(mn)$

مثال:

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

$\longrightarrow O(1)$

* حلقه for داخلی براساس i نوشته شده

روش اول: اجرای دستی انجام میدهیم

if $n=10 \Rightarrow$ می خواهیم ببینیم که حلقه داخلی چند بار تکرار می شود

| i | تعداد تکرار for دوم (دستور داخل) |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| ⋮ | ⋮ |
| 10 | 10 |
| ⋮ | ⋮ |
| n | n |

$\Rightarrow 1+2+3+\cdots+n$

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

$O(n^2)$

روش دوم:

$$\sum_{i=1}^{n}\sum_{j=1}^{i} 1 = \sum_{i=1}^{n} i = 1+2+3+\cdots+n$$

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

نکته:

$$\sum_{i=1}^{n} 1 = \overbrace{1+1+\cdots+1}^{n} = n$$

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

$$\sum_{i=1}^{n} i^p = 1^p + 2^p + \cdots + n^p = \Theta(n^{p+1})$$ حتی اگر p اعشاری باشد

$$\sum_{i=1}^{n} i^{\frac{1}{2}} = \sqrt{1} + \sqrt{2} + \cdots + \sqrt{n} = \Theta(n^{\frac{1}{2}+1}) = \Theta(n^{\frac{3}{2}})$$

مثال:

```
for(i=1; i<=n^2, i++)
  for(j=i; j<=i+n; j++)
    for(k=1; k<=j, k++)
      cout<<k;        ⟶ O(1)
```

$$\sum_{i=1}^{n^2} \sum_{j=i}^{i+n} \sum_{k=1}^{j} 1 = \sum_{i=1}^{n^2} \sum_{j=i}^{i+n} j =$$

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

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

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

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

$$\underbrace{(n+1)\frac{n^2(n^2+1)}{2}}_{\Theta(n^5)} + \underbrace{\frac{n(n+1)}{2}\times n^2}_{\Theta(n^4)} =$$

$\Longrightarrow$ این حلقه مرتبه زمانی $\Theta(n^5)$ است

مثال:

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

در این مثال نمی‌توانیم از سری استفاده کنیم.

| $i$ | تعداد تکرار حلقه دوم (دستور داخل) |
|---|---|
| 1 | $n$ |
| 2 | $n/2$ |
| 3 | $n/3$ |
| ⋮ | |
| $n$ | $n/n$ |

$$\Rightarrow n + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n} =$$

$$n\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right) =$$

$$n\log n$$

$$\sum_{i=1}^{n} \frac{1}{i} \simeq \int_{i=1}^{n} \frac{1}{i} di = \ln i \Big]_1^n = \underbrace{\ln n}_{\log n} - \ln 1^{\,0}$$

نکات:

1) `for (i=a ; i<=b ; i=i+k)`
`s;`

تعداد اجرای این دستور s : $\frac{b-a+1}{k}$

مثال:

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

تعداد اجرای این دستور s : $\frac{n-3+1}{2} = \frac{n-2}{2} = O(n)$

2) `for (i=a; i<=b ; i=i*k)`
`s;`

تعداد اجرای دستور s : $\log_k^b - \log_k^a + 1$

مثال:

`for (i=1; i<=n ; i=i*2)`
`s;`

تعداد اجرای s : $\log n - \log 1 + 1 = \log n + 1$
$= O(\log n)$

اثبات:

| n | تعداد تکرار |
|---|---|
| 1 | 1 ⟶ $\log_2^1 + 1$ |
| 2 | 2 ⟶ $\log_2^2 + 1$ |
| 4 | 3 ⟶ $\log_2^4 + 1$ |
| 8 | 4 ⟶ $\log_2^8 + 1$ |
| 16 | 5 ⟶ $\log_2^{16} + 1$ |
| : | : |
| 22 : n | $\log_2 n + 1$ |

مرتبه را اینطوری تعداد تکرار مشخص می‌کنیم:

$\Longrightarrow O(\log n)$

3) `for(i=n; i>=1; i=i/2)`

`S;`

مثل مثال قبل ثابت می‌شود که از مرتبه $O(\log n)$ است.

مثال:

```
for(i=2; i<=n, i=i+4)
  for(j=n; j>3, j=j-2)
    S;
```

تعداد تکرار for اولی: $\frac{n-2+1}{4} = \frac{n-1}{4}$

تعداد تکرار for دوم: $\frac{n-4+1}{2} = \frac{n-3}{2}$

$\longrightarrow \frac{n-1}{4} \times \frac{n-3}{2}$

$= \frac{n^2-4n+3}{8}$

تعداد اجرای دستور S $\Leftarrow O(n^2)$

مثال:

```
for(i=1; i<=n; i=i*2)
  for(j=1; j<=n, j++)
    S;
```

تعداد تکرار حلقه اولی: $\log n + 1$

تعداد تکرار حلقه دومی: $n$

$\longrightarrow O(n \log n)$

مثال:

```
for(i=1; i<=n, i=i*2)
for(j=1; j<=i; j++)
    s;
```

در این مثال نمی توانیم استفاده کنیم
بخاطر گام حرکت حلقه اولی.
همچنین حلقه داخلی به حلقه بیرونی وابسته است.

| i | تعداد تکرار s |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 4 | 4 |
| ⋮ | |
| n | n |

$$\Rightarrow 1+2+4+8+\cdots+n$$

$$= 2^0+2^1+2^2+2^3+\cdots+2^{\log_2 n}$$

$$= \frac{2^{\log_2 n+1}-1}{2-1} = 2^{\log_2 n+1}-1$$

$$= 2^{\log n}\cdot 2^1 - 1 = 2n-1$$

$$\Rightarrow O(n)$$

روش دوم: می توانیم از تغییر متغیر $i = 2^t$ استفاده کنیم $0 \le t \le \log n$

در این حالت چون گام حرکت 1 می شود می توان از $\sum$ استفاده کرد.

$$i = 2^t \longrightarrow \text{if } i=1 \longrightarrow t=0$$

$$\text{if } i=n \longrightarrow n = 2^t \Rightarrow \log n = \underbrace{\log 2^t}_{t}$$

$$\longrightarrow t = \log n$$

حال می نویسیم:

```
for(t=0; t<=logn, t=t+1)
for(j=1; j<=2^t, j++)
        S;
```

$$\sum_{t=0}^{\log n} \sum_{j=1}^{2^t} 1 = \sum_{t=0}^{\log n} 2^t = 2^0 + 2^1 + 2^2 + \cdots + 2^{\log n}$$

در صفحه قبل ثابت کردیم که این رشته $O(n)$ است.

مثال

```
x=0;
for(int i=2; i<=n; i*=i)
        x+=sin(i);
```

| n | تعداد تکرار | رشته‌ای از تعداد تکرار مشخص می‌کنیم. |
|---|---|---|
| 2 | 1 | |
| 4 | 2 | می‌بینیم که: |
| 16 | 3 | $\log(\log 4)+1 = 1+1 = 2$ |
| 256 | 4 | $\log(\log 16)+1 = 2+1 = 3$ |
| ⋮ | ⋮ | $\log(\log 256)+1 = 3+1 = 4$ |
| n | ? | ⋮ |
| | | $\log(\log n)+1$ |

$$\Rightarrow O(\log(\log n))$$

```
for(i=1; i<n; i*=2) → O(logn)
for(k=1; k<n; k*=2) → O(logn)
        S;
```

$O(\log n * \log n)$

```
while(j>=1) → logn
    for(i=1; i<=n; i++) → n
        j=[j/2]
```

$O(n \log n)$

```
for(i=1; i<=n; i++) → O(n)
{
  for(j=1; j<=m, j++){ O(m)
      x=x+1
  j=1;
  while(j<=n)
    { x=x+1;          } → O(log3 n)
      j=3*j;
    }
}
```

$\Rightarrow O((m+\log n) * n)$

26