ساختمان داده ها والگوریتم ها

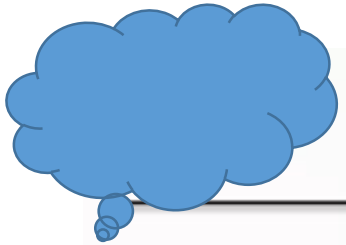
فصل اول

روش های تحلیل الگوریتم

تعریف الگوریتم

- مجموعه ای متناهی از دستور العمل ها که با دنبال کردن آن ها هدف خاصی برآورده می شود. همچنین دارای خصوصیات زیر باشند:

- ✓ **ورودی:** حداقل صفر ورودی
- ✓ **خروجی:** حداقل یک خروجی
- ✓ **قطعیت:** هر دستورالعمل باید واضح و بدون ابهام باشد.
- ✓ **محدودیت (پایان پذیر بودن):** الگوریتم پس از طی مراحلی مشخص باید اتمام پذیرد.
- ✓ **کارایی (انجام پذیر بودن):** هر دستورالعمل باید به گونه ای باشد که شخص بتواند آنرا روی کاغذ به طور دستی اجرا کند.



تفاوت الگوریتم و برنامه

- یک برنامه تمام خصوصیات یک الگوریتم را به جز شرط "پایان پذیر بودن" شامل می شود.
- به عنوان نمونه سیستم عامل برنامه ای است که دائما در یک حلقه انتظاراست تا برنامه بعدی وارد شود.
- سیستم عامل در صورتی پایان می یابد که یا سیستم از کار بیوفتد و یا دچار خرابی شود.

تحلیل برنامه ها

- حافظه
- زمان اجرا
  - ۱- نوع پردازنده
  - ۲- کامپایلر
  - ۳- اندازه ورودی
  - ۴- پیچیدگی الگوریتم

تحلیل برنامه ها

انواع مختلف دستورات:

۱- عبارات توضیحی(comment): تعداد گام های اجرا صفر است.

۲- عبارات تعیین نوع: شامل عباراتی مثل معرفی متغیر ها و ثابت ها و... می باشد. تعداد گام های اجرا صفر است.

۳- دستورات انتساب: تعداد گام های اجرا یک است.

۴- عبارات تکرار(for,while): شمارش گام را فقط برای قسمت کنترل این عبارات در نظر میگیریم.

۵- عبارت if-else: بیشینه زمان اجرای if یا else

```
If{
.
.
}
else{
.
.
}
```

۶- عبارت نام تابع: تعداد گام های اجرا صفر است.

۷- عبارت return: تعداد گام های اجرا یک است.

تحلیل برنامه ها

مثال:

1	int sum (int n){	⟶	0
2	int i;	⟶	0
3	int s=0;	⟶	1
4	for (i=0 ; i<=n ; i++)	⟶	n+2
5	s+=i;	⟶	n+1(1)
6	return s;	⟶	1
}			
			2n+3

تحلیل برنامه ها

مثال:

1	int test (int n){	⟶	0
2	int i, j, a=5, b=10, c=a+b;	⟶	3
3	for (i=1 ; i<=n ; i++)	⟶	n+1
4	for (j=1 ; i<n ; j++){	⟶	n(n)
5	a+=1;	⟶	(n)(n-1)(1)
6	b+=2;	⟶	(n)(n-1)(1)
7	c+=a+b;}	⟶	(n)(n-1)(1)
8	return c;	⟶	1
}			$4n^2 - 2n + 5$

حالت های ورودی

۱- **بهترین حالت**(Best case): ورودی که باعث می شود الگوریتم کم ترین زمان اجرا را داشته باشد.

۲- **بدترین حالت**(Worst case): ورودی که باعث می شود الگوریتم حداکثر زمان اجرا را داشته باشد.

۳- **حالت میانگین**(Average case): در این حالت فرض بر این است که همه حالت های مختلف داده های ورودی، احتمال برابر دارند و میانگین پیچیدگی زمان الگوریتم در همه این حالت ها محاسبه می شود. $T_{AVG} = \sum_{i=1}^{m} p_i * t_i$

p_i = احتمال رخداد دسته i ام t_i = زمان مورد نیاز دسته i ام

تحلیل پیچیدگی زمانی مرتب سازی درجی

```
Insertion sort ( A contain n numbers )

For   k=2   to   n   do

        key ← A[k]

        i ← k

        while  i > 1  and  A[i-1] > key   do

                A[i] ← A[i-1]

                i ← i-1

        A[i] ← key
```

مثال:

K=2	8	**2**	4	9	3	6	3
K=3	2	8	**4**	9	3	6	3
K=4	2	4	8	**9**	3	6	3
K=5	2	4	8	9	**3**	6	3
k=6	2	3	4	8	9	**6**	3
K=7	2	3	4	6	8	9	**3**
K=8	2	3	3	4	6	8	9

تحلیل پیچیدگی زمانی مرتب سازی درجی

مثال:

```
Insertion sort ( A contain n numbers )

For   k=2   to   n   do                          S1
        key ← A[k]                               S2
        i ← k                                    S3
        while  i > 1  and  A[i-1] > key  do       S4
              A[i] ← A[i-1]                      S5
              i ← i-1                            S6
        A[i] ← key                               S7
```

$$T(n) = nS_1 + (n-1)(S_2+S_3) + S_4 \sum_{k=2}^{n} t_k + (S_5+S_6)$$

$$\sum_{k=2}^{n}(t_k-1) + (n-1)S_7$$

$$T(n) = An + B + C \sum_{k=2}^{n} t_k$$

بهترین حالت: $an + b$

بدترین حالت: $a'n^2 + b'n + c'$

حالت میانگین: ؟؟؟

تحلیل پیچیدگی زمانی مرتب سازی درجی

```
Insertion sort ( A contain n numbers )

For  k=2  to  n  do                                  S1
        key ← A[k]                                   S2
        i ← k                                        S3
        while  i > 1  and  A[i-1] > key  do          S4
                A[i] ← A[i-1]                        S5
                i ← i-1                              S6
        A[i] ← key                                   S7
```

	زمان اجرا
α_1	$T(\alpha_1)$
α_2	$T(\alpha_2)$
α_3	$T(\alpha_3)$
α_4	$T(\alpha_4)$
.	.
.	.
.	.
$\alpha_{n!}$	$T(\alpha_{n!})$

$$\text{میانگین} = \frac{\sum_{i=1}^{n!} T(\alpha_i)}{n!}$$

تحلیل پیچیدگی زمانی مرتب سازی درجی

★ مجموع هزینه مصرفی برای مرتب سازی یک جایگشت مثل α_i با α_{n-i} یک عدد ثابت است.

$\alpha_1 , \overline{\alpha_1}$

$\alpha_2 , \overline{\alpha_2}$

$\alpha_3 , \overline{\alpha_3}$

.

.

.

$\alpha_{\frac{n!}{2}} , \overline{\alpha_{\frac{n!}{2}}}$

$$\longrightarrow \quad \frac{\sum_{i=1}^{\frac{n!}{2}} (T(\alpha_i)+T(\overline{\alpha_2}))}{n!} = \frac{\frac{n!}{2} * C}{n!} = \frac{C}{2}$$

تحلیل پیچیدگی زمانی مرتب سازی درجی

8 2 4 9 3 6 3

3 6 3 9 4 2 8

★ هزینه برای هر عدد برابر تعداد اعداد بزرگتر از آن عدد است.

★ پس برای کوچکترین عدد n-1 می شود، برای عدد بعد آن n-2 و به همین ترتیب تا آخر که برای بزرگترین عدد صفر می شود.

$$1 + 2 + \dots + n\text{-}2 + n\text{-}1 = \frac{(n-1)(n)}{2} = \frac{n^2 - n}{2} = C$$

$$\frac{\frac{n!}{2} * C}{n!} = \frac{\frac{n!}{2} * \frac{n^2 - n}{2}}{n!} = \frac{n^2 - n}{4} = \text{حالت میانگین}$$

بی اهمیت بودن ضرایب

الگوریتم	T(n)	حداکثر اندازه ورودی که در 1000 ثانیه حل شود	حداکثر اندازه با پردازنده 10 برابر سریع تر	نسبت	نسبت برای پردازنده 1000 برابر سریعتر
A1	100n	10	100	10	1000
A2	$5n^2$	14	45	3.2	131.94
A3	$\frac{n^3}{2}$	12	27	2.3	125.99
A4	2^n	10	13	1.3	2

A4: n=100 ➔ 2^{100}s ➔ $(2^{10})^{10}$s ➔ 10^{30}s ➔ 10^{20} سال

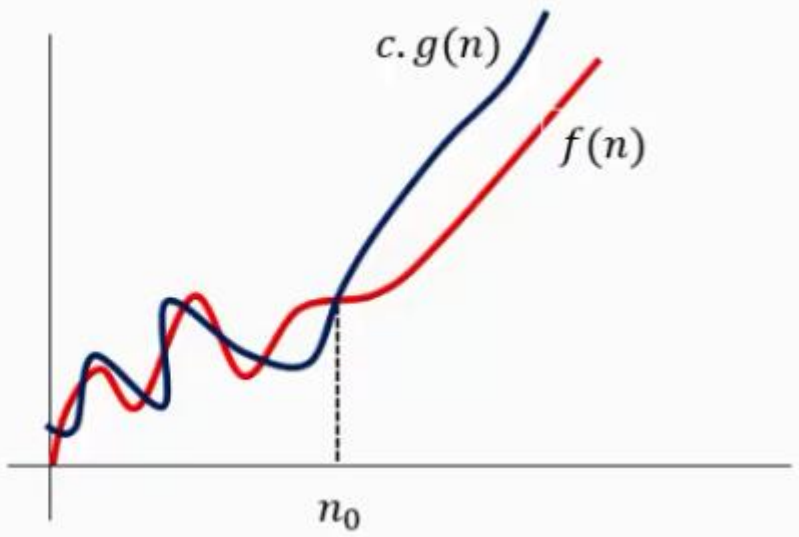
A1: n=100 ➔ 10000s ➔ حدود ۳ ساعت

نماد های مجانبی

o	O	θ	Ω	ω
<	≤	=	≥	>

تعریف O بزرگ:

$F(n) = O(g(n)) \quad \text{---> } \{\exists\ c > 0, n_0 > 0 \mid \forall\ n \geq n_0\ ;\ f(n) \leq c.g(n)\}$



نماد های مجانبی

مثال: $f(n) = 2n^2 - 3n + 4$, $g(n) = n^3$ $f(n) \stackrel{?}{=} O(g(n))$

$2n^2 - 3n + 4 \le c.\, n^3 \xrightarrow{C=2} 2n^2 - 3n + 4 \le 2n^3$

$2n^2 \le 2n^3$

$-3n+4<0 \quad : \quad n>1$

$2n^2-3n+4 \le 2n^3 \quad : n>1$

$c=2 , n_0=2: \forall\, n \ge 2 : 2n^2-3n+4 \le 2n^3$

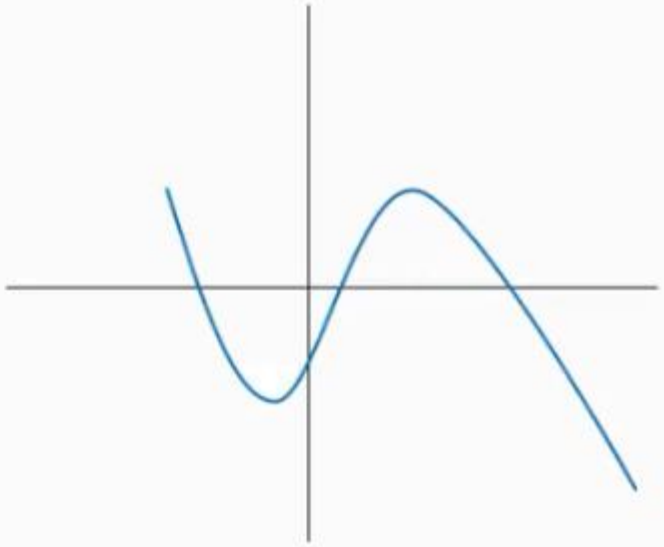
نماد های مجانبی

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

$\{ \nexists\ c, n_0 \mid n \geq n_0 , c.(2n^2 - 3n + 4) \leq n^3 \}$

$-n^3 + 2cn^2 - 3cn + 4 \geq 0$

$n = n_0 + c$



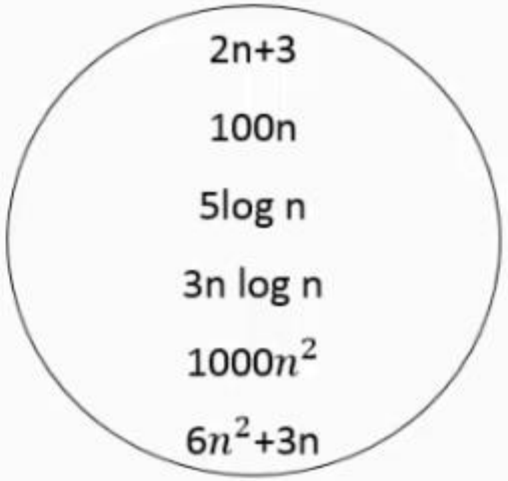
نماد های مجانبی

مثال:

$F(n) = 10^9 n^2$ $\quad\quad$ $g(n) = n^2$

$c = 10^9$, $n = 1$ ----------> $F(n) = O(g(n))$

★به طور کلی می توان نشان داد $O(f(n))$ شامل همه توابعی است که نرخ رشد آن ها کمتر یا مساوی $f(n)$ است.



$O(n^2)$

نماد های مجانبی

تعریف Ω:

$F(n) = \Omega(g(n)) \quad \text{---> } \{\exists\ c > 0, n_0 > 0 \mid \forall\ n \geq n_0\ ;\ f(n) \geq c.g(n)\}$

مثال:

$g(n) = 2n^2 - 3n + 4$ $\qquad$ $f(n) = n^3$

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

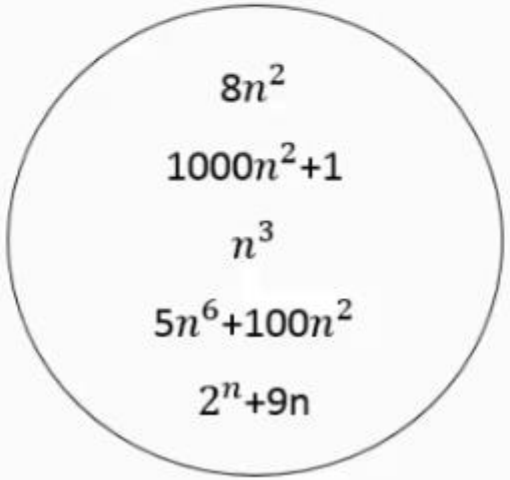
$n^3 \geq 2cn^2 - 3cn + 4c \text{ ----------> } n^3 \geq 2n^2 - 3n + 4 \quad : n \geq 2$

$c = 1$

نماد های مجانبی

$F(n) = 10^9 n^2$ $\quad g(n) = n^2 \quad \text{-------} > \quad g(n) = \Omega(f(n))$

★به طور کلی می توان نشان داد $\Omega(f(n))$ شامل همه توابعی است که نرخ رشد آن ها بیشتر یا مساوی $f(n)$ است.



$\Omega(n^2)$

نماد های مجانبی

تعریف θ :

$F(n) = \theta(g(n)) \quad \text{---> } \{\exists\ c_1, c_2 > 0, n_0 > 0 \mid \forall\ n \geq n_0\ ;\ c_1.g(n) \leq f(n) \leq c_2.g(n)\}$

$F(n) = \theta(g(n)) \quad \text{<----> } F(n) = O(g(n)) \text{ , } F(n) = \Omega(g(n))$

مثال:

$f(n) = 3n + 3 \qquad g(n) = 2n$

$c_1 = 1 \text{ , } c_2 = 2 \quad \text{--------> } 2n \leq 3n+3 \leq 4n \quad \xrightarrow{n_0 = 3} \quad F(n) = \theta(g(n))$

★ θ دقیق ترین بیان برای نرخ رشد یک تابع است.

نماد های مجانبی

قضیه: اگر $f(n) = a_k n^k + \dots + a_1 n^1 + a_0$ باشد، می توان نشان داد که $F(n) = \Theta(n^k)$.

$F(n) = O(n^k)$

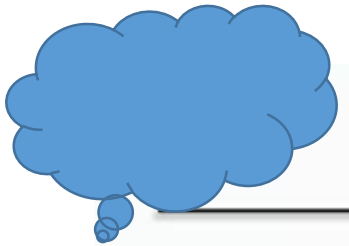
$\exists\ c, n_0 > 0\ ;\ a_k n^k + \dots + a_1 n^1 + a_0 \leq c\, n^k$

$$c = \sum_{i=0}^{k} |a_i| \ , \ n = 1$$

$F(n) = \Omega(n^k)$

$\exists\ c, n_0 > 0\ ;\ a_k n^k + \dots + a_1 n^1 + a_0 \geq c\, n^k$

$$c = \min(a_k, a_{k-1}, \dots, a_1, a_0) \ , \ n = 1$$



نماد های مجانبی

نرخ رشد برخی از توابع:

$$\log^* n < \log n < \sqrt{n} < \sqrt{n}\log n < n < n\log n \approx \log(n!) < n^2 < n^3 < (\log n)! <$$
$$2^n < 3^n < (\log n)^n < n! < n^n$$

توابع بازگشتی

تعریف: به هر تابع یا زیر برنامه ای که بتواند داخل بدنه اش، خودش را دوباره فراخوانی کند، تابع یا زیربرنامه بازگشتی می گویند. مثلا:

$n! = n * (n-1)!$

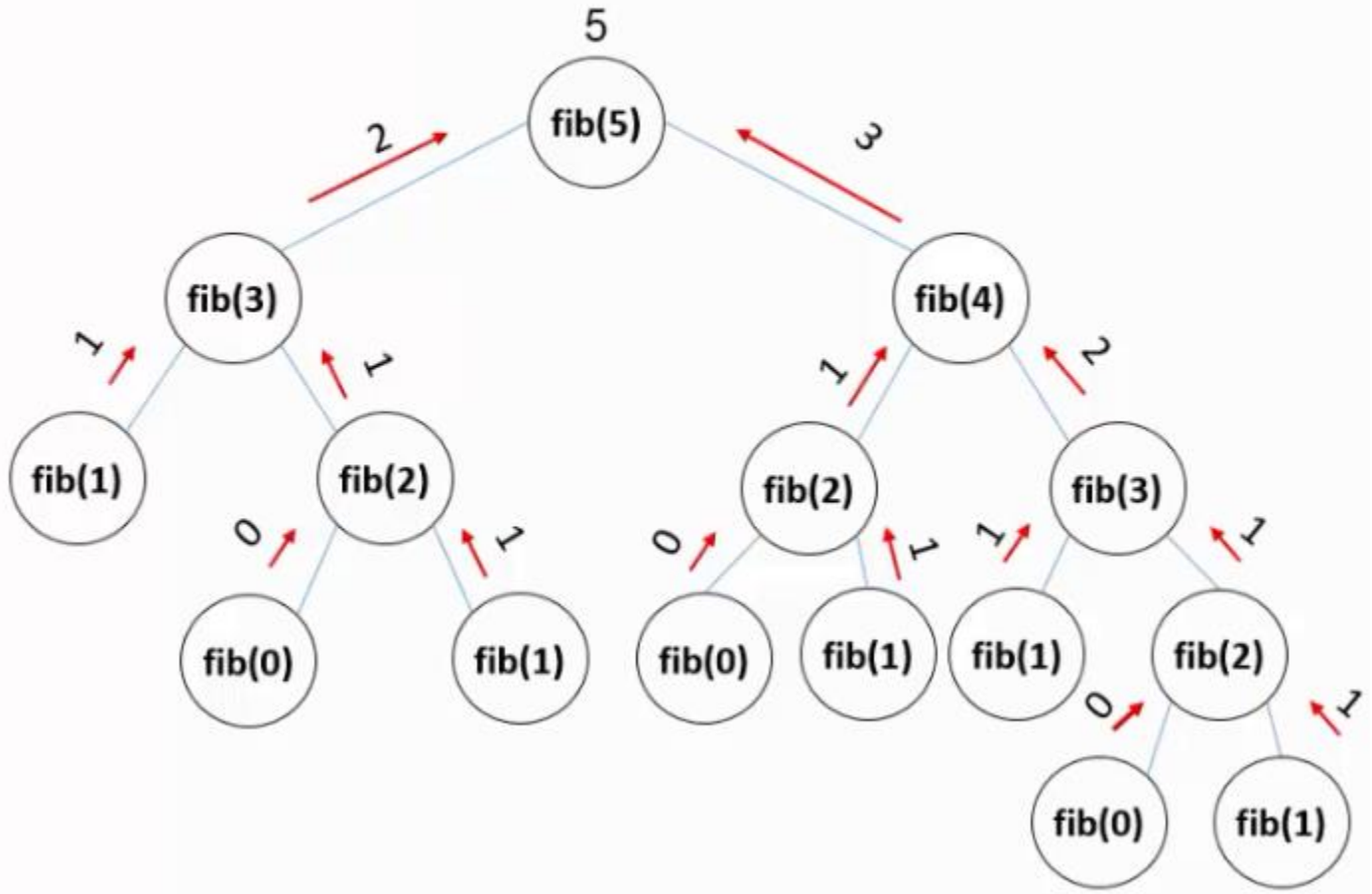
```
Int fact( int x ){
if(x>1)
        return x * fact(x-1);
else
        return 1;
}
```

$x^n = x * x^{n-1}$

```
Int pow( int x, int n ){
if(n>=1)
        return x * pow(x,n-1);
else
        return 1;
}
```

تابع بازگشتی سری فیبوناچی

```
int fib (int n)
{
If(n<=1)
return n;
Else
return (fib(n-1) + fib(n-2));
}
```



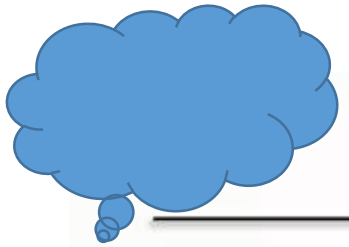
محاسبه مرتبه زمانی توابع بازگشتی

هر تابع بازگشتی دارای ۳ ویژگی زیر می باشد:

۱- شرایط آغاز: مقداری از اندازه مسئله که تابع برای اولین بار با آن فراخوانده می شود.

۲- شرط توقف: مقداری از اندازه مسئله که وقتی اندازه مسئله به آن مقدار برسد فراخوانی مجدد تابع را متوقف می کنیم.

۳- قسمت کاهش مسئله: قسمتی که در آن تابع مجددا فراخوانی می شود تا به شرط توقف برسیم.



محاسبه مرتبه زمانی توابع بازگشتی

❖ همیشه زمانی که اندازه مسئله به شرط توقف برسد الگوریتم زمان ثابتی را صرف می کند.(ثابت c)

❖ زمانی که هنوز اندازه مسئله به شرط توقف نرسیده باشد فراخوانی مجدد صورت می گیرد که به این معنی است که زمان $T(n)$ براساس زمان فراخوانی مجدد اما با اندازه مسئله کاهش یافته به صورت یک رابطه بازگشتی به دست می آید.(در این قسمت، باز هم ثابت c نیز به آن اضافه می شود)

محاسبه مرتبه زمانی توابع بازگشتی

```
Int fact( int x ){
if(x>1)
        return x * fact(x-1);
else
        return 1;
}
```

$$T(n) = T(n-1)+c \qquad n>c$$

$$T(n) = c \qquad n<=c$$

```
int fib (int n)
{
If(n<=1)
return n;
Else
return (fib(n-1) + fib(n-2));
}
```

$$T(n) = T(n-1)+T(n-2)+c \qquad n>c$$

$$T(n) = c \qquad n<=c$$

حل معادلات بازگشتی

روش جایگذاری

مثال:

```
int test (int n)
{
if(n<=c)
        return 1;
else
        return (test(n-1) + test(n-1));
}
```

$$T(n) = 2T(n-1) + c \qquad n>c$$

$$T(n) = c \qquad n<=c$$

حل معادلات بازگشتی

مثال:

$$T(n) = 2*T(n-1) + c = 2*[2*T(n-2)+c]+c = 4T(n-2)+3c = 4*[2*T(n-3)+c]+3c$$

$$= 8T(n-3)+7c = 8*[2*T(n-4)+c]+7c = 16T(n-4)+15c = \ldots.$$

$$T(n) = 2^k T(n-k)+ (2^k-1)\, c \xrightarrow[k = n - c]{\text{شرط توقف : } n-k=c} T(n) = 2^{n-c} T(n-n+c)+ (2^{n-c}-1)\, c$$

$$= 2^{n-c} T(c)+ (2^{n-c})\, c - c \overset{T(c)=c}{=} c(2^{n-c+1} -1) \text{ -------> } T(n) = O(2^n)$$

حل معادلات بازگشتی

مثال:

```
int test2 (int n)
{
if(n<=c)
        return 1;
else
        return (test(n/2) + test(n/2));
}
```

$$T(n) = 2T(\frac{n}{2}) + c \qquad n>c$$

$$T(n) = c \qquad n<=c$$

حل معادلات بازگشتی

مثال:

$$T(n) = 2*T(\frac{n}{2}) + c = 2*[2*T(\frac{n}{4})+c]+c = 4T(\frac{n}{4})+3c = 4*[2*T(\frac{n}{8})+c]+3c$$

$$= 8T(\frac{n}{8})+7c = 8*[2*T(\frac{n}{16})+c]+7c = 16T(\frac{n}{16})+15c = \dots.$$

$$T(n) = 2^k T(\frac{n}{2^k})+ (2^k-1)\ c \xrightarrow[k = log\ \frac{n}{c}]{\frac{n}{2^k} = c : \text{شرط توقف}} T(n) = 2^{\log \frac{n}{c}}\ T(\frac{n}{2^{\log \frac{n}{c}}})+ (2^{\log \frac{n}{c}} -1)\ c$$

$$= \frac{n}{c}\ T(c)+ (\frac{n}{c})\ c - c \overset{T(c)=c}{=} 2n - c \dashrightarrow T(n) = O(n)$$