ساختمان داده‌ها (۲۲۸۲۲)

مدرس: حسین بومری

[زمستان 99]

جلسه ۱: هفتم　　　　نگارنده: رضا پیشکو

در جلسه قبل مرتبه زمانی معرفی شد و تعدادی مثال از محاسبه مرتبه زمانی حل شد در این جلسه بحث مرتبه زمانی با مثال های بیشتر ادامه پیدا کرد و توضیحاتی هم در مورد سایز ورودی داده شد.

۱ جلسه هفتم

مثال:

```
PUBLIC STATIC VOID PRINT1(int n)
for (i = 0; i < n; i++)
        System.out.println(i);
```

تابع print1 از O(n) است . روش حل به این گونه است که یک مجموعه از عملگر ها را پیدا کنیم که به ازای بقیه عملیات ها این عملگر ها هم تکرار شوند. که در اینجا عملگر مقایسه همراه همه عملیات ها صدا زده می شود و خود مقایسه نیز n+1 بار تکرار مي شود. اگر I را اندازه ورودی در نظر بگیریم. می دانیم I برابر است با log(n). پس مرتبه این تابع نسبت به اندازه ورودی نمایی است و برابر با 2^I است. ⊠

مثال:

```
PUBLIC STATIC VOID PRINT1(int[ ] numbers)
for (i = 0; i <length[numbers]; i++)
        System.out.println(numbers[i]);
```

در این مثال هر کاراکتر که چاپ می شود یک زمانی می برد و مرتبه چاپ شدن هر کاراکتر برابر با log(n) است. پس اگر I را برابر با اندازه ورودی و M را برابر با ماکسیمم عدد در نظر بگیریم؛ I برابر است با nlog(M). که n تعداد اعداد آرایه هست. مانند مثال قبل با در نظر گرفتن عملگر مقایسه متوجه می شویم که تابع از O(n) است. و در نتیجه نسبت به اندازه ورودی از $O(I/log(M))$ است. حالا اگر فرض کنیم که تعداد بیت های عدد ثابت است یعنی M ثابت است؛ مرتبه تابع نسبت به سایز ورودی همان O(I) است. ⊠

از اینجا به بعد فرض شد که تعداد بیت های یک عدد برای عملی مثل پرینت یا جمع کردن ثابت است.

مثال:

PUBLIC STATIC VOID PRINT1(int[ ] numbers)

$n = length[numbers]$
for $(i = 1; i < n; i+ = i)$
    $System.out.println(i);$

در این مثال براي محاسبه مرتبه كافي است محاسبه كنيم i چند بار تغيير مى كند و براى اينكه اين تعداد تغييرات راحت تر ديده شود يک متغير کمکی میسازیم و به صورت زیر تابع را در نظر میگیریم.

PUBLIC STATIC VOID PRINT1(int[ ] numbers)

$n = length[numbers]$
for $(i = 1, j = 1\ i < n; i+ = i, j++)$
    $System.out.println(i);$

در این جالت واضح است که j از ۱ شروع می شود و نهایتا به log(n) می رسد. در نتیجه این تابع از O(log(n)) است. زیرا j در هر لحظه برابر با log(i) است. $\boxtimes$

مثال:

PUBLIC STATIC VOID PRINT1(int[ ] numbers)

$n = length[numbers]$
for $(i = 0, j = 1\ i < n; i+ = j, j++)$
    $System.out.println(i);$

اگر به مقادیر i نگاه کنیم به این صورت هستند : 1, 1+2, 1+2+3, ... , 1+2+...+k که چون $k(k+1)/2$ باید از n کمتر باشد پس k از مرتبه $n^{1/2}$ است. و از طرفی به ازای هر مقدار i یک عملیات انجام می دهیم پس k عملیات انجام می دهیم. در نتیجه مرتبه کل تابع k یا همان log(n) است. $\boxtimes$

مثال: در این مثال تابع بازگشتی توان را بررسی می کنیم.

PUBLIC STATIC INT POW(int n, int p)

$if(p = 0)$
    $return 1$
$else$
    $return\ \ pow(n, p-1) + n$

اگر T(n,p) را مقدار زمان اجرای تابع برای ورودی های n و p باشد. انگاه داربم :

T(n,0) = O(1)

T(n,p) = T(n, p-1) + O(1)

حالا T(n,p) را به این صورت مجاسبه می کنیم:

T(n,p) = T(n, p-1) + O(1)*1 = T(n, p-2) + O(1)*2 = ... = T(n, 0) + O(1)*p = O(1)p + O(1)

در نتیجه :

T(n,p) = O(p)

حالا همان تابع توان را به شکل دیگری می نویسیم.

```
PUBLIC STATIC INT POW(int n, int p)
if(p = 0)
        return1
int  result = pow(int n, int p) * pow(int n, int p)
if(p%2 = 1)
        result+ = n
return  result
```

در این مثال هم رابطه ای که برای T(n,p) داریم به این شکل است :

T(n,p) = 2*T(n,p/2) + 1

T(n,0) = O(1)

همانطور که مشخص است این ربطی مستقل از n است و می توانیم آن را به صورت زیر بنویسیم.

T(p) = 2*T(p/2) + 1

T(0) = O(1)

برای محاسبه T(p) در این حالت داریم :

T(p) = 2*T(p/2) + O(1) = 2*(2*T($p/2^2$) + O(1)) + O(1) = ... = 2*(2*...(2*T($p/2^{log(p)}$))) + O(1)) + O(1)

T(p) = 2*2*...*2*T($p/2^{log(p)} + 2^{log(p)} * O(1) + ... + 2 * O(1) + O(1)$

T(p) = $2^{log(p)} + (2 * 2^{log(p)} - 1) * O(1) = O(2^{log(p)}) = O(p)$

پس فرقی با حالت قبل نکرد اما اگر به صورت زیر تابع را می نوشتیم مرتبه تغییر می کند.

```
PUBLIC STATIC INT POW(int n, int p)
if(p = 0)
        return1
int  half = pow(int n, int p/2)
int  result = half * half
if(p%2 = 1)
        result+ = n
return  result
```

در این حالت رابطه ای که داریم به صورت زیر است:

$$T(p) = T(p/2) + 1$$

$$T(0) = O(1)$$

برای محاسبه $T(p)$ در این حالت داریم :

$$T(p) = T(p/2) + O(1) = T(p/2^2) + 2*O(1) = ... = T(p/2^{log(p)}) + log(p)*O(1)$$

$$T(p) = T(0) + log(p)*O(1)$$

$$T(p) = log(p)$$

در این حالت با حذف شدن ضریب پشت $T(p/2)$ مرتبه زمانی از $(O(p))$ به $log(p)$ تغییر کرد.

⊠

مثال:

```
PUBLIC STATIC INT FACT(int n)
int  result = 0
for (i = 0; i < n; i++)
        result+ = fact(n − 1)
return  result
```

T is thye number of operations

$$T(n) = n*T(n-1) = n*(n-1)*T(n-2) = ... = n*(n-1)*...*1$$

⊠

$$T(n) = O(n!)$$

مثال:

```
PUBLIC STATIC INT FIB(int n)
if(n = 0||n = 1)
    return 1
return fib(n - 1) + fib(n - 2)
```

T is thye number of operations

T(n) = T(n-1) + T(n-2) + O(1)

برای حل معادلات بازگشتی مانند این مثال ابتدا آن را دو قسمت میکنیم. یکی بخش همگن :

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

و یک قسمت غیر همگن :

T(n) = T(n-1) + T(n-2) + H(n)

ابتدا به جل قسمت همگن می پردازیم.

قبل از شروع به حل مثال بالا به این مثال ساده تر توجه کنیم :

T(n) = T(n-1) + T(n-1) = 2*T(n-1), T(0) = 1

در نتیجه :

T(n) = 2*2*T(n-2)

T(n) = 2^n*T(0)

T(n) = 2^n

در این حالت جواب نهایی شد. پس می توان حدس زد که جواب مثال اصلی نیز نمایی باشد. حالا می خواهیم با استقرا ثابت کنیم که جواب رابطه بازگشتی T(n) = c1*T(n-1) + c2*T(n-2) نمایی است.

حدس : T(n) = r^n

$r^n = c_1 * r^{n-1} + c_2 * r^{n-2}$ = lrT(n)

$r^2 - c_1 * r - c_2 = 0$

اگر r_1, r_2 ریشه های چند جمله ای بالا باشند انگاه :

T(n) = $c_1 * r_1^n + c_2 * r_2^n$

و نهایتا نکته ای که می ماند این است که c_1, c_2 را محاسبه کنیم که به جلسه بعد موکول شد.

⊠