مرکز آموزش الکترونیکی دانشگاه علم و صنعت ایران

طراحی و تحلیل الگوریتمها

فصل اول: مقدمه ای بر تحلیل الگوریتمها

(جلسه دوم)

تحلیل الگوریتمها

- برای محاسبه پیچیدگی الگوریتمها، تعداد دفعات تکرار عمل اصلی را مشخص می کنیم
- در الگوریتمها، معمولاً عمل اصلی طوری انتخاب می شود که بیشتر از سایر دستورات تکرار می شود
- چنانچه اجرای یک الگوریتم، فقط به اندازه مسئله بستگی داشته باشد، آنگاه هر سه نماد O، θ، Ω با هم برابرند
- اگر اجرای الگوریتم به شرایطی دیگر به جز اندازه مسئله بستگی داشته باشد (مانند مقدار و نوع پارامترها)، ممکن است مقدار این سه نماد متفاوت باشد

مثال

```
s=0;i=1;
for(i=1;i<=n;i++)
while(i<=n){
  s=s+i;
s=0;i=1;
  i=i+5;
while(i<=n){
}s=s+i;
i=i*5;
}
```

$O=\theta=\Omega=n$

$O=\theta=\Omega=n$

تعداد دفعات اجرای بدنه حلقه $n/5$ می باشد

i: 1 5 25 125 … n

5^0 5^1 5^2 5^3 … 5^x

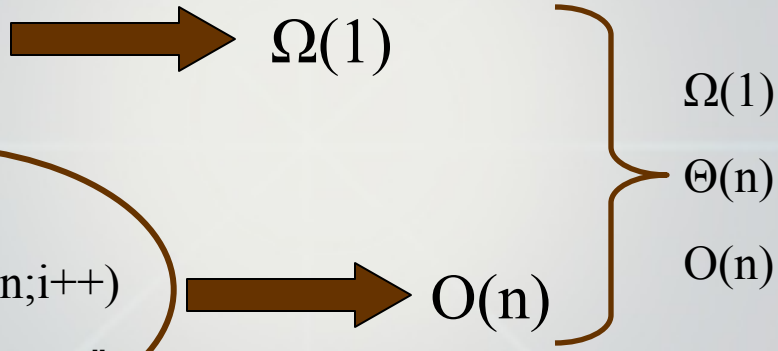
$x=\log n$ تعداد دفعات تکرار

$O=\theta=\Omega=\log n$

مثال

- در الگوریتمهایی مانند الگوریتم زیر بسته به پارامتر مسئله زمان اجرا متفاوت است

```
void  f(int n)
{   if(n%2==0)
        return;
    else
        for(i=1;i<=n;i++)
            cout<<"hello";
}
```



مثال

- فرض کنید که آرایه A دارای n عنصر باشد. بهترین و بدترین زمان اجرای الگوریتم زیر را تعیین کنید

```
void f(int A[],int n)
{  i=0;
   while(A[i]>A[i+1]){
        swap(A[i],A[i+1]);
        i++;
    }
}
```

3,6,2,1,7

بهترین حالت زمانی است که اولین عنصر آرایه از دومین عنصر کوچکتر باشد

7,6,3,2,1

بدترین حالت زمانی است که هر عنصر از عنصر بعدی خود بزرگتر باشد (آرایه صعودی مرتب شده باشد) در این حالت شرط حلقه همیشه درست است تا i به n برسد

مثال(مرتب سازی حبابی (Bubble sort

```
void Bubble_sort(int A[], int n)
{   for(i=0;i<=n-2;i++)
    {   for(j=n-1;j>=i+1;j--)
        {   if(A[j-1]>A[j])
            {   temp=A[j-1];
                A[j-1]=A[j];
                A[j]=temp;
            }
        }
    }
}
```

$$O=\Omega=\theta=n^2$$

اگر آرایه صعودی مرتب شده باشد (بهترین حالت) در این صورت شرط if هیچگاه صحیح نیست و دستورات درون مستطیل هیچگاه اجرا نمی شوند

اگر آرایه نزولی مرتب شده باشد (بدترین حالت) آنگاه در هر تکرار حلقه for دستورات شرط if اجرا می شوند. تعداد دفعات اجرای این دستورات در این حالت دقیقاً $n(n-1)/2$ می باشد

مثال(مرتب سازی درجی (Insertion sort

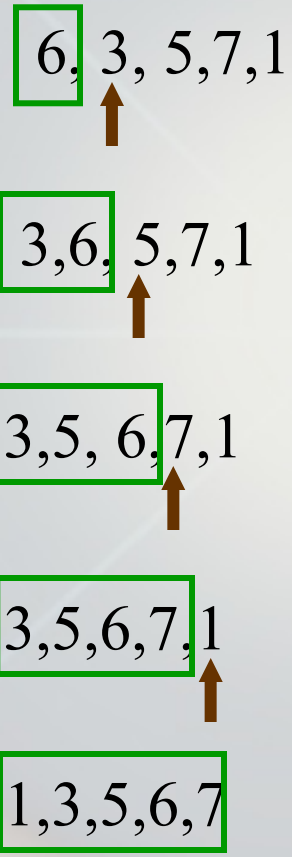
```
void  insertion_sort(int A[], int n)
{   for(i=1;i<n;i++)
    {   j=i;
        while(A[j]<A[j-1] && j>0)
        {   swap(A[j],A[j-1]);
            j=j-1;
        }
    }
}
```

اگر آرایه صعودی مرتب شده باشد، شرط حلقه while همیشه نادرست است و بدنه آن اجرا نمی شود در این حالت $\Omega(n)$

اگر آرایه به صورت نزولی مرتب شده باشد، در هر تکرار حلقه for شرط حلقه while درست بوده و به اندازه i بار تکرار می شود. در این حالت $O(n^2)$

در حالت متوسط، این الگوریتم از $\theta(n^2)$ است

مثالی از مرتب سازی درجی



تمرین: برای آرایه زیر مراحل اجرای الگوریتم مرتب سازی درجی را نشان دهید

8,6,5,4,2,1

مثال

- الگوریتم زیر را حساب کنید order

```
void  f(int n,int m)
{  for(i=1;i<=n;i++)
     cout<<“hello”;
   for(j=1;j<=m;j++)
     cout<<“hello”;
}
```

O(n)

O(m)

O(n+m)

تمرین: اگر m=log n باشد، پیچیدگی تابع فوق چقدر است؟

مثال

* order الگوریتم زیر را حساب کنید

```
void  f(int n,int m)
{  for(i=1;i<=n;i++)
     cout<<“hello”;
   for(j=1;j<=m;j++)
     for(k=1;k<=m;k++)
       cout<<“hello”;
}
```

$O(n)$

$O(m^2)$

$O(n+m^2)$

مثال

- فرض کنید آرایه A دارای n خانه باشد و در هر خانه ای عددی صحیح بزرگتر یا مساوی صفر ذخیره شده به قسمی که مجموع عناصر ذخیره شده m باشد.به عنوان مثال در آرایه مقابل n=5 و m=4 می باشد 1,0,0,2,1
- با مفروضات فوق پیام hello چند بار چاپ می شود؟
- طبق فرض مسئله این مجموع برابر m باشد.

```
for(i=0;i<n;i++)
    for(j=0;j<=A[i];j++)
        cout<<"hello";
```

دفعات اجرای cout

0 ← بار A[0]

1 ← بار A[1]

2 ← بار A[2]

n-1 ← بار A[n-1]

تمرین: نشان دهید order این الگوریتم O(n+m) است

تحلیل الگوریتمهای بازگشتی

- روش تحلیل الگوریتمهای بازگشتی با روش تحلیل الگوریتمهای غیر بازگشتی متفاوت است
- یک روش برای تحلیل الگوریتمهای بازگشتی تحلیل آنها از روی درخت بازگشت است
- روش فوق همیشه عملی نیست

```
int  pow(int a,int n)
{  if (n==0) return 1;
   else return a*pow(a,n-1);
}
```

```
int  pow(int a,int n)
{   if(n==0) return 1;
    else return pow(a,n/2)*pow(a,(n+1)/2);
}
```

یک روش مناسب برای تحلیل الگوریتمهای بازگشتی(۱)

- بر اساس سایز مسئله، تابعی تشکیل می دهیم که سایز مسئله پارامتر آن باشد
- به عنوان مثال، اگر سایز مسئله n باشد تابع مورد نظر T(n) خواهد بود
- فرض می کنیم T(n) زمان اجرای الگوریتم برای مسئله ای با سایز n باشد
- بر اساس فرض فوق و با استفاده از بدنه الگوریتم یک رابطه بازگشتی برای T(n) ارائه داده و آنرا حل می کنیم

یک روش مناسب برای تحلیل الگوریتمهای بازگشتی(۲)

- در دو تابع زیر که برای محاسبه a به توان n نوشته شده اند، روابط بازگشتی عبارتند از:

```
int pow(int a,int n)
{ if (n==0) return 1;
  else return a*pow(a,n-1);
}
```

pow(int a,int n) → T(n)

if (n==0) return 1 → T(0)=1

pow(a,n-1) → T(n-1)

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

$$T(0)=1$$

```
int pow(int a,int n)
{ if(n==0) return 1;
  else return pow(a,n/2)*pow(a,(n+1)/2);
}
```

pow(int a,int n) → T(n)

if(n==0) return 1; → T(0)=1

pow(a,n/2) → T(n/2)

pow(a,(n+1)/2); → T((n+1)/2)

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

$$T(0)=1$$

حل روابط بازگشتی

- پس از تعیین رابطه بازگشتی مورد نظر، برای تحلیل الگوریتم، باید آن رابطه را حل نمود
- برای حل روابط بازگشتی ۳ روش وجود دارد
  - جایگذاری با تکرار (حدس و استقرا)
  - استفاده از معادلات مشخصه
  - استفاده از سریهای مولد

روش جایگذاری با تکرار

- در این روش به جای هر جمله در رابطه بازگشتی مقدار آنرا بر اساس رابطه اولیه قرار می دهیم و این کار را آنقدر ادامه می دهیم تا به کران رابطه برسیم
- این روش برای حل روابط بازگشتی درجه ۱ مناسب است
- رابطه بازگشتی درجه ۱ به رابطه ای گفته می شود که جمله n ام فقط بر اساس یک جمله دیگر بیان شده باشد.هر یک از روابط زیر درجه ۱ می باشند

$T(n)=T(n-1)+1$	$T(n)=T(n-2)+n$	$T(n)=T(n/2)+1$	$T(n)=2T(n/3)+n$
$T(1)=1$	$T(1)=0$	$T(1)=1$	$T(1)=1$

مثال

- رابطه بازگشتی مقابل را حل کنید:
- مقدار T(n-1) با توجه به رابطه مقابل $\begin{cases} T(n)=T(n-1)+1 \\ T(0)=1 \end{cases}$ برابر T(n-2)+1 می باشد پس

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

- مقدار T(n-2) بر اساس رابطه برابر است با T(n-3)+1 پس

$$T(n)=T(n-2)+2=\{T(n-3)+1\}+2=T(n-3)+3$$

- اگر به همین صورت ادامه دهیم:

$$T(n)=T(n-3)+3=T(n-4)+4=\ldots=T(n-(n-1))+(n-1)=T(1)+n-1=T(0)+n=1+n \Rightarrow T(n)=n+1 \Rightarrow T(n)=O(n)$$

مثال

- رابطه بازگشتی مقابل را حل کنید:

$$\begin{cases} T(n)=2T(n/2)+1 \\ T(1)=1 \end{cases}$$

- برای راحتی حل مسئله n را توانی از ۲ در نظر می گیریم(تا n به راحتی بر ۲ قابل تقسیم باشد) مثلاً فرض می کنیم: $n = 2^m$
- با توجه به رابطه فوق T(n/2) برابر است با 2T(n/4)+1 پس:

$$T(n) = 2T(n/2) + 1 = 2\{2T(n/2^2) + 1\} + 1 = 2^2 T(n/2^2) + 2 + 1 =$$
$$2^2\{2T(n/2^3) + 1\} + 2 + 1 = 2^3 T(n/2^3) + 2^2 + 2 + 1 = \ldots =$$
$$2^m T(n/2^m) + 2^{m-1} + 2^{m-2} + \ldots + 2 + 1 = 2^m + 2^{m-1} + \ldots + 2 + 1 =$$
$$2^{m+1} - 1 = 2 \times 2^m - 1 = 2n - 1$$

→ T(n)=2n-1 → T(n)=O(n)

قضیه اصلی برای حل روابط بازگشتی

- روابط بازگشتی که به صورت مقابل باشند را می توان با استفاده از قضیه زیر پیچیدگی آنها را بدست آورد بدون اینکه آنها را حل نمود
- این روابط را روابط تقسیم و غلبه گویند

$$\begin{cases} T(n)=aT(n/b)+C\,n^k & b>1 \text{ و } a>=1 \text{ اعداد مثبت و } a \text{ و } b \text{ و } C \text{ و } k \\ T(1)=c \end{cases}$$

$$\begin{cases} T(n)=\theta(n^{\log_b^a}) & \text{اگر } a > b^k \text{، در اینصورت:} \\ T(n)=\theta(n^k \log n) & \text{اگر } a= b^k \text{ در اینصورت:} \\ T(n)=\theta(n^k) & \text{اگر } a < b^k \text{، در اینصورت:} \end{cases}$$

مثال

- با استفاده از قضیه اصلی پیچیدگی تابع زیر چقدر است؟

$$\begin{cases} T(n)=2T(n/2)+1 \\ T(1)=1 \end{cases} \rightarrow \begin{cases} a=2,\ b=2,\ k=0 \\ a > b^k \end{cases} \rightarrow T(n)=\theta(n)$$

- با استفاده از قضیه اصلی پیچیدگی تابع زیر چقدر است؟

$$\begin{cases} T(n)=2T(n/2)+5n \\ T(1)=3 \end{cases} \rightarrow \begin{cases} a=2,\ b=2,\ c=5,\ k=1 \\ a = b^k \end{cases} \rightarrow T(n)=\theta(n \log n)$$

استفاده از روش معادله مشخصه(۱)

- معمولاً برای حل معادلات درجه ۲ به کار می رود
- شکل کلی یک رابطه بازگشتی درجه ۲ به صورت زیر می باشد

$$T(n) + aT(n-1) + bT(n-2) = c^n f(n)$$

- که در آن a,b,c اعداد حقیقی بوده و f(n) یک چند جمله ای از درجه d می باشد
- برای حل این رابطه ابتدا باید ریشه های آنرا بدست آورد، سپس هر ترکیب خطی این ریشه ها پاسخ رابطه خواهد بود
- ریشه های این روابط دو دسته اند: ریشه های قسمت همگن و ریشه های قسمت ناهمگن

استفاده از روش معادله مشخصه(۲)

- برای بدست آوردن ریشه های قسمت همگن، جملات رابطه بازگشتی را یک طرف و برابر صفر قرار می دهیم
- با فرض اینکه پاسخ T(n) به صورت عددی حقیقی به توان n است (مثلاً r)، به جای آن قرار داده و به یک معادله درجه ۲ می رسند

$$r^n + ar^{n-1} + br^{n-2} = 0 \longrightarrow r^2 + ar + b = 0$$

- ریشه های این معادله درجه ۲ ریشه های قسمت همگن می باشند

استفاده از روش معادله مشخصه(۳)

- سپس ریشه های قسمت ناهمگن را از فرمول زیر محاسبه می کنیم

$$(r-c)^{d+1}=0$$

- معادله فوق دارای d+1 ریشه مضاعف می باشد
- ترکیب خطی ریشه های قسمت همگن و ناهمگن جواب مسئله است
- چنانچه ریشه ای تکراری باشد، به ازای هر تکرار یک ضریب n در ترکیب خطی به آن اضافه می شود
- برای محاسبه ضرایب ثابت در ترکیب خطی از مقادیر کران در رابطه بازگشتی استفاده کرده و با تشکیل دستگاههای معادله و مجهول آنها را محاسبه می کنیم

مثال

- رابطه بازگشتی مقابل را حل کنید:

$T(n)-3T(n-1)+2T(n-2)=n+3$

$T(1)=T(2)=1$

- ابتدا ریشه های قسمت همگن:

$T(n)-3T(n-1)+2T(n-2)=0 \rightarrow r^n - 3r^{n-1} + 2r^{n-2} = 0$

$\rightarrow r^2 - 3r + 2 = 0 \rightarrow \begin{cases} r_1 = 1 \\ r_2 = 2 \end{cases}$

- قسمت ناهمگن را می توان به صورت مقابل نوشت $1^n(n+1)$
- درنتیجه c=1 و d=1، درنتیجه ریشه های قسمت ناهمگن برابر است با $r_3 = r_4 = 1$

مثال(ادامه)

- ترکیب خطی این ریشه ها پاسخ مسئله است
- چون ۳ ریشه تکراری برابر با ۱ داریم، در ترکیب خطی مورد نظر به ازهی هر کدام باید یک ضریب n اضافه گردد
- پاسخ به صورت زیر خواهد بود

$$T(n) = c_1 2^n + c_2 1^n + c_3 n 1^n + c_4 n^2 1^n$$

- که در آن ضرایب c_1, c_2, c_3, c_4، مقادیر ثابتی هستند که با تشکیل یک دستگاه معادله می توان آنها را محاسبه کرد
- چون هدف تعیین پیچیدگی الگوریتمها می باشد، نیازی به محاسبه آنها نیست

$$T(n)=O(2^n)$$

مثال

- با توجه به رابطه بازگشتی مقابل،تعیین کنید که T(n)=O(?)

$T(n)-5T(n-1)+6T(n-2)=6$

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

- چنانچه با روش توضیح داده شده ریشه های قسمت همگن را بدست آوریم عبارتند از 2 و 3
- ریشه قسمت غیر همگن نیز برابر ۱ است(چون f(n) برابر 6 است و در نتیجه درجه آن صفر است)
- جواب:

$T(n) = c_1 2^n + c_2 3^n + c_3 1^n \Rightarrow T(n)=O(3^n)$

استفاده از تغییر متغیر برای حل روابط بازگشتی

- در مواردی که تابع بازگشتی پیچیده تر باشد، روشهای فوق قادر به حل آن نخواهد بود
- در برخی موارد می توان با تغییر متغیر، شکل رابطه بازگشتی را به گونه ای تغییر داد که با استفاده از روشهای گفته شده بتوان آنرا حل نمود
- نحوه تغییر متغیر در هر مسئله ای متفاوت است و نیاز به تمرین دارد

مثال

- در رابطه مقابل مشخص کنید T(n)=O(?)

$T(n)=T(n/2)+\log n$

$T(1)=1$

فرض کنید $n=2^m$

$\Rightarrow T(2^m)=T(2^{m-1})+\log 2^m \Rightarrow T(2^m)=F(m)$

$\Rightarrow F(m)=F(m-1)+m \Rightarrow F(m)=F(m-2)+(m-1)+m$

$\Rightarrow F(m)=F(0)+1+2+\ldots+(m-1)+m$

$T(1)=1$

$F(0)=1$

$\Rightarrow F(m)=1+(m(m+1)/2)$

$\Rightarrow F(m)=O(m^2) \Rightarrow T(n)=F(m)=O(m^2)$ $m=\log n$ $\Rightarrow T(n)=O((\log n)^2)$

مثال

- در رابطه مقابل مشخص کنید T(n)=O(?)

$$T(n) = 2T(\sqrt{n}) + \log n, \quad n = 2^m$$

T(1)=1

T(n)=T(2^m)=F(m) , m=log n, $\sqrt{n} = \sqrt{2^m} = 2^{\frac{m}{2}}$

→ F(m)=2F(m/2)+m → F(m)=O(m log m)

→ T(n)=F(m)=O(m log m)=O((log n)*(log log n))

مثال

- در رابطه مقابل مشخص کنید T(n)=O(?)

$$T(n)=\sqrt{n}T(\sqrt{n})+n,\quad n=2^m$$

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

$$\Rightarrow \frac{T(n)}{n}=F(n) \Rightarrow F(n)=F(\sqrt{n})+1$$

$$\Rightarrow F(n)=F(2^m)=S(m) \Rightarrow S(m)=S(m/2)+1$$

$$\Rightarrow S(m)=O(\log m) \Rightarrow F(n)=S(m)=O(\log m)=O(\log\log n)$$

$$\Rightarrow \frac{T(n)}{n}=F(n) \Rightarrow T(n)=nF(n)=O(n \log\log n)$$

مثال

- رابطه بازگشتی مقابل را حل کنید

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

$$T(n)=T(n-1)+T(n-2)+\ldots+T(1)$$
$$T(n-1)=T(n-2)+T(n-3)+\ldots+T(1)$$
$$T(n)-T(n-1)=T(n-1)$$

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

تمرین: رابطه فوق را با روش جایگذاری با تکرار حل کنید