۲- تحلیل الگوریتمها

در این بخش به تعریف نمادهای مجانبی مورد نیاز در تحلیل الگوریتمها و بیان چند قضیه قضیه در مورد آن میپردازیم.

۲-۱- نمادهای مجانبی

تعریف: $f(n)\in O(g(n))$ اگر و فقط اگر ثابت c و ثابت $n_\circ$ وجود داشته باشند که برای همه مقادیر $n \ge n_\circ$، داشته باشیم.

$$\forall n \ge n_0 : |f(n)| \le c|g(n)|$$

تعریف: $f(n)\in \Omega[g(n)]$ اگر و فقط اگر ثابت c و ثابت صحیح $n_\circ$ وجود داشته باشد که:

$$\forall n \ge n_0 : |f(n)| \ge c|g(n)|$$

در این صورت کران پایین تعداد اعمال لازم جهت یک الگوریتم به $c|g(n)|$ محدود گردیده است. و بدین صورت خوانده می شود: $f(n)$ اُمگای بزرگ $g(n)$ است.

تعریف: $f(n)\in \theta[g(n)]$ اگر و فقط اگر ثابتهای c_1 و c_2 و ثابت صحیح $n_\circ$ وجود داشته باشد به گونه ای که برای همه مقادیر $n \ge n_\circ$ داشته باشیم.

$$\forall n \ge n_0 : c_1|g(n)| \le |f(n)| \le c_2|g(n)|$$

قضیه (۱):

$$if\ \ f(n) = a_m n^m + \cdots + a_1 n + a_0\ \ then\ \ f(n) = \begin{cases} O(n^m) \\ \Omega(n^m) \\ \Theta(n^m) \end{cases}$$

قضیه (۲):

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

قضیه (۳):

$$f(n)=\Theta(g(n)) \Leftrightarrow \begin{cases} f(n)=O(g(n)) \\ f(n)=\Omega(g(n)) \end{cases}$$

تعریف: $f(n)\in o(g(n))$ اگر و فقط اگر ثابت c و ثابت $n_\circ$ وجود داشته باشند که برای همه مقادیر $n \geq n_\circ$، داشته باشیم:

$$\forall n \geq n_0 :| f(n)|<c.| g(n)|$$

نماد o را میتوان به گونه دیگری نیز تعریف کرد:

$$f(n)=o(g(n)) \Leftrightarrow \lim_{n\to\infty}\frac{f(n)}{g(n)}=0$$

مثال:

$$\frac{n}{\log n}=o(n)$$

تعریف: $f(n)\in \omega(g(n))$ اگر و فقط اگر ثابت c و ثابت $n_\circ$ وجود داشته باشند که برای همه مقادیر $n \geq n_\circ$، داشته باشیم:

$$\forall n \geq n_0 :| f(n)|>c.| g(n)|$$

نماد ω را میتوان به گونه دیگری نیز تعریف کرد:

$$f(n)=\omega(g(n)) \Leftrightarrow \lim_{n\to\infty}\frac{f(n)}{g(n)}=\infty$$

نکته: مقایسه نمادهای مجانبی دو تابع f و g را با دو عدد حقیقی a و b میتوان بصورت زیر بیان کرد:

$$\begin{aligned} f(n)=\mathrm{O}(g(n)) &\approx a\leq b \\ f(n)=o(g(n)) &\approx a<b \\ f(n)=\Omega(g(n)) &\approx a\geq b \\ f(n)=\omega(g(n)) &\approx a>b \\ f(n)=\Theta(g(n)) &\approx a=b \end{aligned}$$

نکته:

$$2^n \neq \mathrm{O}(2^n)$$

۲-۲- تحلیل حالت متوسط الگوریتم

تحلیل حالت متوسط یک الگوریتم دارای اهمیت بیشتری نسبت بدترین حالت و بهترین حالت الگوریتم میباشد. اگر تعداد تکرار یک دستورالعمل در یک الگوریتم با n ورودی را در حالت متوسط با A(n) نشان دهیم آنگاه یک رابطه ساده جهت بدست آوردن حالت متوسط بصورت زیر میباشد:

$$A(n)=\sum_{i\in S}P(i).F(i)$$

که در آن S مجموعه حالات ممکنه، P(i) احتمال رخ دادن حالت i ام و F(i) تعداد تکرار دستور مورد نظر در حالت i ام میباشد.

مثال: متوسط تعداد تکرار دستورالعمل (×) را در برنامه زیر بدست آورید:

```
Procedure   max (A,n,j)
    j← 1
   for  i←2   to  n   do
          if   A(i) > A(j)    then
                j←i          (*)
          endif
    repeat
end.
```

تعداد تکرار دستور× در یک آرایه n تایی را بطور متوسط برابر با A_n در نظر میگیریم و لذا:

$$0\le A_n<n-1$$

تعداد تکرار دستور (*) در این زیربرنامه در صورتی صفر است که بزرگترین عنصر در ابتدای آرایه باشد.

$$\begin{cases}A_1=0\\ A_n=\dfrac{n-1}{n}A_n+\dfrac{1}{n}(1+A_{n-1})\end{cases}$$

$$\Rightarrow A_n=\frac{1}{n}+A_{n-1}=\frac{1}{n}+\frac{1}{n-1}+\cdots+\frac{1}{2}+A_1=\underbrace{\frac{1}{n}+\frac{1}{n-1}+\cdots+\frac{1}{2}+\frac{1}{1}}_{\substack{=H_n\\ =\ln n+\gamma+\Theta(\frac{1}{n})\\ \gamma=0.5772}}-1=\Theta(\log n)$$

تعداد تکرار دستور $A(i)>A(j)$

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

پس همیشه n-1 بار تکرار می شود.

مثال: متوسط تعداد تکرار دستورالعمل(×) را در برنامه زیر بدست آورید؟

```
Function Linear_search(A, n, x)
   for   i←1   to   n   do
      if   A(i)=x    then       (*)
            return i
      endif
   repeat
end.
```

$$\text{متوسط تکرار} = \frac{1}{n}\times 1+\frac{1}{n}\times 2+\frac{1}{n}\times 3+\cdots+\frac{1}{n}\times n=\frac{1}{n}\times n=\frac{1}{n}(1+2+3+\cdots+n)$$

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

نکته: در الگوریتم قبلی در هر صورت حلقه تا n اجرا می شد ولی در این مثال هرگاه که شرط برقرار باشد از حلقه خارج می شویم.

مثال: میانگین زمان اجرای الگوریتم QuickSort را بدست آورید:

$$T(n)=\begin{cases} 0 & ;n=1 \\ (n-1)+\dfrac{2}{n}\displaystyle\sum_{i=1}^{n-1}T(i) & ;n>1 \end{cases}$$

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

$$\Rightarrow \begin{cases} nT(n)=n(n-1)+2\displaystyle\sum_{i=1}^{n-1}T(i) \\ (n+1)T(n+1)=n(n+1)+2\displaystyle\sum_{i=1}^{n}T(i) \end{cases} *(-1)+$$

$$\Rightarrow (n+1)T(n+1)-nT(n)=2n+2T(n)$$

$$\Rightarrow (n+1)T(n+1)=2n+(n+2)T(n)$$

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

$$\overbrace{\Rightarrow}^{n\geq 1\Rightarrow \frac{2n}{n+1}\leq 2} \quad T(n+1)\leq 2+\frac{n+2}{n+1}T(n)$$

$$\Rightarrow T(n)\leq 2+\frac{n+1}{n}T(n-1)\Rightarrow T(n)\leq 2+\frac{n+1}{n}[2+\frac{n}{n-1}T(n-2)]$$

$$\Rightarrow T(n)\leq 2\frac{n+1}{n+1}+2\frac{n+1}{n}+\frac{n+1}{n-1}T(n-2)$$

$$\Rightarrow T(n)\leq 2(n+1)(\frac{1}{n+1}+\frac{1}{n})+\frac{n+1}{n-1}[2+\frac{n-1}{n-2}T(n-3)]$$

$$\Rightarrow T(n)\leq 2(n+1)(\frac{1}{n+1}+\frac{1}{n}+\frac{1}{n-1})+\frac{n+1}{n-2}T(n-3)$$

$$\Rightarrow T(n)\leq 2(n+1)\underbrace{(\frac{1}{n+1}+\frac{1}{n}+\frac{1}{n-1}+\cdots+\frac{1}{3})}_{H_{n+1}-\frac{1}{2}-1}+\frac{n+1}{2}\underbrace{T(1)}_{0}$$

$$\Rightarrow T(n)=\mathrm{O}(n.\log n)$$

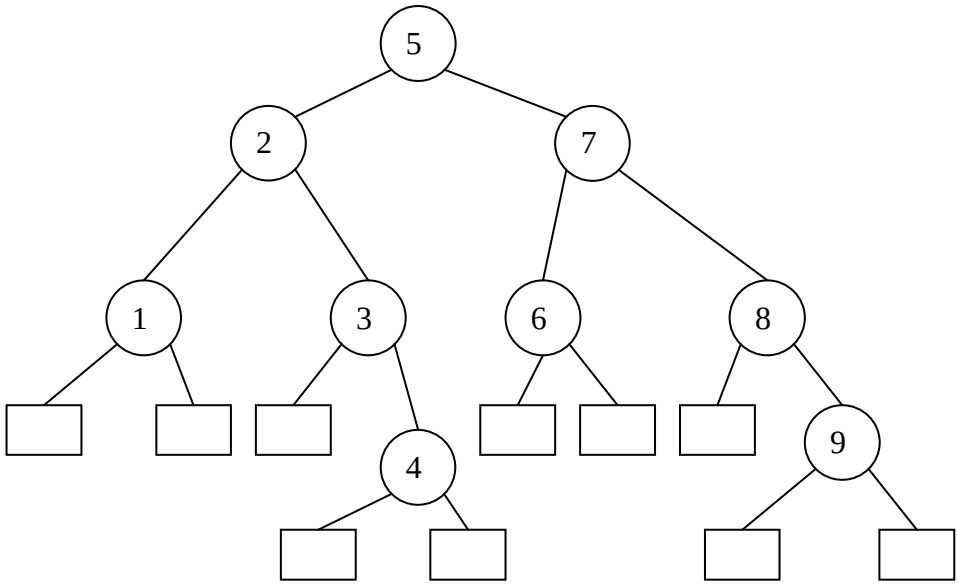
نتیجه: بدترین حالت میانگین زمان اجرای الگوریتم QuickSort برابر $O(n. \log n)$ میباشد.

مثال: فرض کنید یک آرایه مرتب (صعودی) و کلید x داده شده باشند. متوسط زمان اجرای الگوریتم BinarySearch را محاسبه کنید.

```
Function BinarySearch( A, n, x)
   low ← 1
   high←n
   while low≤high do
      mid←⌊(low+high)/2⌋
      if x=A(mid) then
         return mid
      elsif x>A(mid) then
         low←mid+1
      else
         high←mid-1
      endif
   repeat
   return 0
end.
```

در این الگوریتم اگر x داخل آرایه موجود باشد تعدادی مقایسه با عناصر آرایه صورت میگیرد و نهایتاً جستجو موفق خواهد بود و اگر x داخل آرایه موجود نباشد، تعدادی مقایسه با عناصر آرایه صورت میگیرد و نهایتاً جستجو ناموفق خواهد بود. مقایسه ها بصورت جستجو در یک درخت جستجوی دودویی انجام میشوند برای نمونه به آرایه زیر و درخت حاصل از جستجوی آن دقت کنید:

A	1	2	3	4	5	6	7	8	9
مقادیر	6	10	16	19	21	42	66	81	93
تعداد مقایسات در حالت موفق	3	2	3	4	1	3	2	3	4



در درخت بالا گره های خارجی با مستطیل و گره های داخلی با دایره نشان داده شده اند.درخت بالا را درخت تصمیم دودویی مینامند. جستجوهای موفق به گره های داخلی و جستجوهای ناموفق به گره های خارجی ختم میشوند.

نکته: اگر تعداد گره های خارجی را با n_0 نشان داده و تعداد گره های داخلی را با n نشان دهیم آنگاه $n_0=n+1$

میانگین جستجوهای موفق=(3+2+3+4+1+3+2+3+4)/9=25/9=2.77

نکته: فرض کنید احتمال موجود بودن x داخل عناصر مختلف آرایه و همچنین در صورت موجود نبودن x در آرایه احتمال قرار داشتن x بین عناصر آرایه مساوی است(به این معنی که احتمال قرار داشتن x در مجموعه [5,9] و [11,15] و ... مساوی است(کلاسها متساوی الاحتمالند).

A	1	2	3	4	5		6	7	8	9
تعداد مقایسات در حالت ناموفق	3	3	3	4	4	3	3	3	4	4

میانگین جستجوهای ناموفق=(3+3+3+4+4+3+3+3+4+4)/10=37/10=3.7

قضیه: اگر $n\in[2^{k-1},2^k)$ باشد، آنگاه الگوریتم جستجوی دودویی مستلزم حداکثر k مقایسه برای جستجوی موفق و k-1 یا k مقایسه برای جستجوی ناموفق است.

نتیجه: جستجوی ناموفق دارای همیشه دارای مرتبه زمانی $\Theta(\log n)$ است. جستجوی موفق در بهترین وضعیت $\Omega(1)$ و در بدترین وضعیت $O(\log n)$ میباشد.

حال متوسط تعداد مقایسات در جستجوی موفق را محاسبه میکنیم:

اگر v یک گره داخلی در درخت تصمیم باشد و سطح آن را با Level(v) نشان دهیم، آنگاه تعریف میکنیم:

$$d(v)=level(v)-1$$

$$E=\sum_{v\in External\ \ Nodes} d(v)$$

$$I=\sum_{v\in Internal\ \ Nodes} d(v)$$

قضیه: E=I+2n

راهنمایی اثبات: با استقرا اثبات کنید.

حال اگر میانگین تعداد مقایسات در جستجوی موفق در یک درخت تصمیم با n گره داخلی را با S(n) و میانگین مقایسات در جستجوی ناموفق را با U(n) نشان دهیم، آنگاه:

$$S(n)=\frac{\sum_{v\in Internal\ \ Nodes}[d(v)+1]}{n}=\frac{I+n}{n}$$

$$U(n)=\frac{\sum_{v\in External\ \ Nodes} d(v)}{n+1}=\frac{E}{n+1}$$

حال با استناد به قضیه بالا و دو رابطه به دست آمده میتوان نتیجه گرفت که:

$$\Rightarrow S(n)=\frac{E-2n+n}{n}=\frac{(n+1)U(n)-n}{n}=(1+\frac{1}{n})U(n)-1$$

و چون قبلاً دیدیم که $U(n)=\Theta(\log n)$ در نتیجه:

$$S(n)=\Theta(\log n)$$

۲-۳- روابط بازگشتی

در این قسمت به طور مختصر از حل روابط بازگشتی سخن خواهیم گفت. روابط بازگشتی را میتوان از روی درجـه آن دسته بندی کرد:

۲-۳-۱-روابط بازگشتی درجه ۱

این روابط به سادگی از طریق روش جایگذاری قابل حل میباشند. به این معنی که با باز کردن رابطه بصورت تـو در تو میتوان رابطه را حل نمود.

مثال:

$$T(n)=\begin{cases}2T(n-1)+1 & ;n>0\\ 1 & ;n=0\end{cases}$$

$$\begin{aligned}\Rightarrow T(n)&=2T(n-1)+1\\ &=2[2T(n-2)+1]+1=4T(n-2)+2+1=2^2T(n-2)+2^1+2^0\\ &=2^iT(n-i)+2^{i-1}+\cdots+2^1+2^0\\ &=2^nT(0)+2^{n-1}+\cdots+2^0=\sum_{i=0}^{n}2^i=2^{n+1}-1=\mathrm{O}(2^n)\end{aligned}$$

۲-۳-۲-روابط بازگشتی درجه ۲ (همگن)

این روابط به روش محسبه معادله مشخصه قابل حل هستند.

مثال: تعداد فراخوانی های تابع مقابل را بدست آورید؟

```
function   f(n)
    if   n<2    then
          return     n
    else
          return    f(n-1)+f(n-2)
    end if
end.
```

A_n= تعداد فراخوانی های تابع f با ورودی n

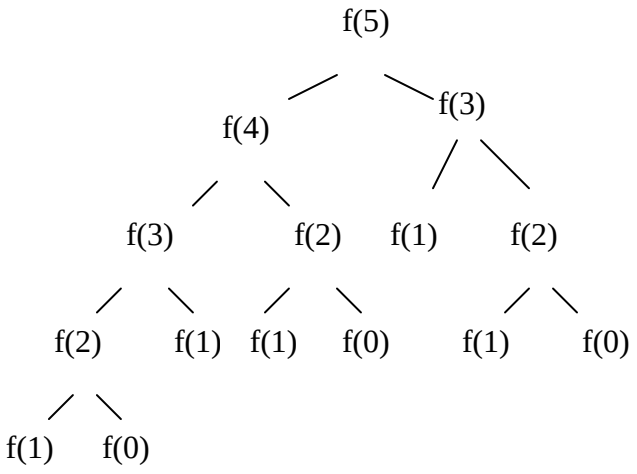
$$\begin{cases}A_n=A_{n-1}+A_{n-2}+1\\ A_0=1\\ A_1=1\end{cases}$$

راه حل:

$$A_n = A_{n-1} + A_{n-2} + 1$$
$$A_n + 1 = A_{n-1} + A_{n-2} + 1 + 1$$
$$*B_n = A_n + 1$$
$$* \Rightarrow B_n = B_{n-1} + B_{n-2}$$
$$B_0 = 2 \qquad B_1 = 2$$
$$B_n = r^n \Rightarrow r^n = r^{n-1} + r^{n-2}$$
$$r^2 - r - 1 = 0$$
$$r_1 = \frac{1+\sqrt{5}}{2} \qquad r_2 = \frac{1-\sqrt{5}}{2}$$
$$B_n = C_1 r_1^n + C_2 r_2^n$$
$$\Rightarrow \quad B_0 = C_1 r_1^0 + C_2 r_2^0 = C_1 + C_2 = 2 \quad if \quad n=0 \quad ①$$
$$\Rightarrow \quad B_1 = C_1 r_1^1 + C_2 r_2^1 = C_1\left(\frac{1+\sqrt{5}}{2}\right) + C_2\left(\frac{1-\sqrt{5}}{2}\right) = 2 \quad if \quad n=1 \quad ②$$
$$①و② \quad \Rightarrow \quad C_1 = \frac{1+\sqrt{5}}{\sqrt{5}} \qquad C_2 = \frac{1-\sqrt{5}}{\sqrt{5}}$$
$$B_n = \frac{2}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \left(\frac{1-\sqrt{5}}{2}\right)^{n+1}\right]$$
$$A_n = B_n - 1 \cong (1/6)^n$$

روش حل پروسیجر قبلی تقسیم و حل میباشد و یک درخت را به شکل زیر پویش می کند این الگـوریتم مقـادیری را چندین بار بصورت تکراری محاسبه می کند و به همین علت مرتبه زمانی الگوریتم نمائی می باشد.

مثال درخت فراخوانی f(5) :



این مسئله را به وسیله یک حلقه برگ‌ها به سمت ریشه با حفظ مقادیر هم می‌توان حل کرد که این روش پـایین بـه بالا (bottom-up) است در این روش حالت‌های تکراری حذف می‌شود. (روش برنامه سازی پویا)

۲-٤- قضیه اصلی (Master Theorem)

در فصل بعدی با یکی از روشهای حل مسئله به نام روش تقسیم و حل آشنا خواهیم شد که در تحلیل مرتبه زمـانی الگوریتمهای طراحی شده به آن روش به قضیه‌ای به نام قضیه اصلی نیاز خواهیم داشت و لذا در این قسمت به بیان آن خواهیم پرداخت.

قضیه اصلی:

$$T(n) = aT(\frac{n}{b}) + Cn^k \quad if$$

که در آن c,a,b و k اعداد ثابت و b>1 است آنگاه:

$$T(n) = \begin{cases} \theta\left(n^{\log_b^a}\right) & a > b^k \\ \theta\left(n^k \log n\right) & a = b^k \\ \theta\left(n^k\right) & a < b^k \end{cases}$$

مثال:

زمان اجرای Binary search را به کمک قضیه اصلی به دست آورید.

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

$$\begin{cases} a = 1 \\ b = 2 & a = b^k \Rightarrow 1 = 2^0 \\ k = 0 & \theta\left(n^k \log n\right) = \theta(\log n) \end{cases}$$

نکته: در قضیه اصلی مقدار c در زمان نهایی تأثیری ندارد.

مثال: زمان اجرای مثال‌های زیر را به کمک قضیه اصلی محاسبه کنید؟

$$T(n) = T(\frac{3n}{2}) + n^2 \Rightarrow$$

$$\begin{cases} a = 1 \\ b = \frac{2}{3} & a < b^k \\ k = 2 & \theta\left(n^k\right) = \theta\left(n^2\right) \end{cases}$$

مثال:

$$T(n) = 2T\left(\sqrt{n}\right) + \log n$$

$$n = 2^m \Rightarrow m = \log_2^n \quad ①$$

$$T\left(2^m\right) = T\left(2^{\frac{m}{2}}\right) + m$$

$$T\left(2^m\right) = s(m)$$

$$s(m) = 2s\left(\frac{m}{2}\right) + m$$

$$\theta(m \log m), \quad ① \quad \Rightarrow \theta(\log n . \log\log n)$$

مثال:

$$T(n) = 3T\left(\frac{n}{4}\right) + n\log n$$

$$\begin{cases} a = 3 \\ b = 4 \\ k = 1/x \end{cases} \qquad n\log n = n^{1/x} \qquad n < n\log n < n^2$$

$$\theta\left(n^{1/x}\right) = \theta\left(n\log n\right)$$