۶۳- با توجه به رابطه بازگشتی زیر، $T(n)$ از چه مرتبه‌ای است؟

$$\begin{cases} T(n^3) = T((n-1)^3) + \theta(n) \\ T(1) = 1 \end{cases}$$

$= T((n-2)^3) + \theta(n-1) + \theta(n) = \cdots = 1 + \theta(2) + \cdots + \theta(n)$
$= 1 + 2 + \cdots + n$
$= \theta(n^2)$

۱) ۱ ۲) $n^{\frac{1}{3}}$

۳) $n^{\frac{2}{3}}$ ۴) n

$T(n^3) = \theta(n^2)$

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

قضیه: اگر $2^{k-1} \le n < 2^k$ آنگاه جستجوی دودویی برای جستجوی موفق مستلزم حداکثر k مقایسه و برای جستجوی ناموفق دقیقا k یا k-1 مقایسه است.

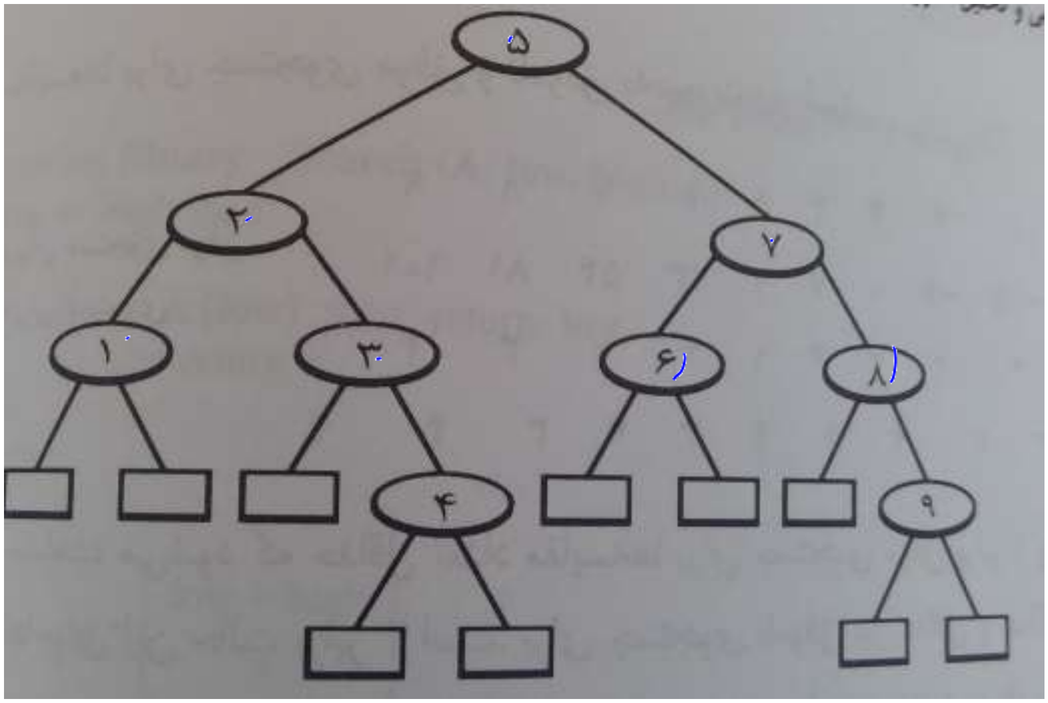
اثبات: اگر درخت دودویی روش را بنویسیم در جستجوی موفق الگوریتم در گرهی داخلی و در جستجوی ناموفق در گرهی خارجی متوقف می شود. چون $2^{k-1} \le n < 2^k$ پس گره های داخلی در سطح 1 تا k قرار دارند ولی گره های خارجی در سطح k یا k+1. پس تعداد مقایسه برای ختم به گره در سطح i برابر i است ولی برای ختم به گره خارجی در سطح i برابر i-1 است.

به بیان دیگر: در جستجوی نا موفق تعداد مقایسه برابراست با $\lfloor \log n \rfloor$ یا $\lfloor \log n \rfloor + 1$

در جستجوی موفق حداکثر مقایسه برابر است با $\lfloor \log n \rfloor + 1$

مثال: درخت دودویی برای آرایه

i	۱	۲	۳	۴	۵	۶	۷	۸	۹
$A[i]$	−۱۵	−۶	۰	۷	۹	۲۳	۵۴	۸۱	۱۰۱



تعداد مقایسه ها برای جستجو موفق و ناموفق:

i	۱	۲	۳	۴	۵	۶	۷	۸	۹		
$A[i]$	−۱۵	−۶	۰	۷	۹	۲۳	۵۴	۸۱	۱۰۱		
	۳	۲	۳	۴	۱	۳	۲	۳	۴		موفق
	۳	۳	۳	۴	۴	۳	۳	۳	۴	۴	ناموفق

- میانگین تعداد جستجوی موفق $S(n)$ برابر است با مجموع تعداد گره ها در هر سطح ضربدر شماره سطح تقسیم بر تعداد گره های موفق.
- میانگین تعداد جستجوی ناموفق $U(n)$ برابر است با مجموع تعداد گره ها در هر سطح ضربدر شماره سطح والد تقسیم بر تعداد گره های ناموفق.

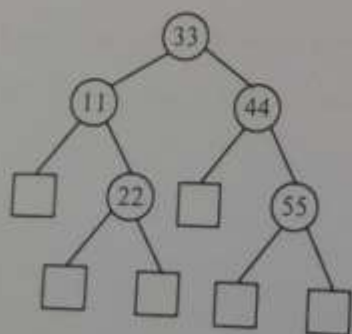
مثال : فرض کنید ارایه‌ای ۵ خانه‌ای و مرتب داریم و می‌خواهیم میانگین تعداد مقایسه‌ها را برای جستجوی موفق و نیز میانگین تعداد مقایسه‌ها را برای جستجوی ناموفق به‌دست آوریم :

1	2	3	4	5
11	22	33	44	55

برای یافتن عدد 11 به ۲ مقایسه، عدد 22 به سه مقایسه، عدد 33 به یک مقایسه و عدد 44 به دو مقایسه و عدد 55 به سه مقایسه نیاز است. پس :

$$\text{میانگین تعداد مقایسه‌ها برای جستجوی موفق} = S(n) = \frac{2+3+1+2+3}{5} = \frac{11}{5}$$

این روش شبیه یافتن اعداد در درخت جستجوی (BST) زیر است که متوسط سطح هر گره داخلی را به‌دست می‌آوریم:



$$\text{میانگین تعداد مقایسه‌ها برای جستجوی موفق} = S(n) = \frac{1+2+2+3+3}{5} = \frac{11}{5}$$

همچنین رابطه زیر بین میانگین تعداد جستجوی موفق و نا موفق ثابت شده است:

$$S(n) = \left(1 + \frac{1}{n}\right) U(n) - 1$$

مرتبهٔ زمانی / حالت‌ها	بهترین حالت	حالت میانگین	بدترین حالت
جستجوی موفق	$\theta(۱)$	$\theta(\log n)$	$\theta(\log n)$
جستجوی ناموفق	$\theta(\log n)$	$\theta(\log n)$	$\theta(\log n)$

یافتن بزرگترین و کوچکترین عنصر آرایه

هدف یافتن بزرگترین و کوچکترین عنصر یک آرایه n عضوی است.

الگوریتم معمولی:

```
Algorithm Simple _ Max _ Min (A, n, max, min)
    max = min = A[1]
    for i = ۲ to n do
    {
        if (A[i] > max) then max = A[i]
        if (A[i] < min) then min = A[i]
    }
```

پیچیدگی زمانی در حالات بهترین، بدترین و میانگین 2(n-1) مقایسه است.

چون مقایسه $A[i] < min$ فقط زمانی لازم است که $A[i] > max$ نباشد، پس با اصلاحی جزیی زیر می توان الگوریتم قبل را بهبود داد:

```
if (A[i] > max) then max = A[i]
    else if (A[i] < min) then min= A[i]
```

برای نسخه بهبود یافته:

بهترین حالت: زمانی است که آرایه بصورت صعودی مرتب باشد و تعداد مقایسه n-1 است.

بدترین حالات: زمانی است که آرایه بصورت نزولی مرتب باشد و تعداد مقایسه 2(n-1) است.

حالت میانگین: در این حالت $A[i] > max$ در نصف کل حالات درست است. پس تعداد مقایسه ها $\frac{3}{2}n - 1$ است.

الگوریتم تقسیم و غلبه

```
Algorithm Find _ Max _ Min (low, high, fmax, fmin)
if  low = high  then  fmax = fmin = A[low]
    else  if  high  =  low  +١  then
     {
        if A[low]  <  A[high] then
        {
            fmax = A[high] ;  fmin = A[low]
        }
         else
        {
            fmax = A[low] ; fmin = A[high]
        }
     }
else
  {
          mid = ⌊ (low + high) / ٢ ⌋

          Find _ Max _ Min (low,  mid,  lmax,  lmin)
          Find _ Max _ min ( mid + ١ , high,  rmax,  rmin)
          if   lmax  >  rmax   then   fmax = lmax
               else  fmax = rmax
          if   lmin  <  rmin   then   fmin = lmin
               else   fmin = rmin
  }
```

$$T(n) = T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + T\left(\left\lceil\frac{n}{2}\right\rceil\right) + 2, T(1) = 0, T(2) = 1$$

اگر n توانی از 2 باشد داریم:

$$T(n) = 2T(\frac{n}{2}) + 2, T(1) = 0, T(2) = 1$$

$a=۲ \quad b=۲ \quad n^{\log_b^a}=n$

حال از قضیه اصلی نتیجه می شود: تحت ا قضیه اصلی

$$\underline{T(n) = \theta(n)}$$

$\underline{n=۲^k}$

جواب دقیق آن به روش جایگذاری نیز $T(n)$=$\frac{3n}{2}-2$ است:

$$T(n)=۲T(\frac{n}{۲})+۲=۲\left(۲T(\frac{n}{۴})+۲\right)+۲=۲^۲T(\frac{n}{۲^۲})+۲^۲+۲^۱$$

$$=۲^۲\left(۲T(\frac{n}{۲^۳})+۲\right)+۲^۲+۲^۱=۲^۳T(\frac{n}{۲^۳})+۲^۳+۲^۲+۲^۱$$

$$=\dots=۲^{k-1}\underline{T(\frac{n}{۲^{k-1}})}+\underbrace{۲^{k-1}+۲^{k-2}+\dots+۲^۱+۲^۰}_{\frac{۲^k-1}{۲-1}=۲^k-1}-۲^۰$$

$$=۲^{k-1}+۲^k-1-1=$$

$$\frac{n}{۲}+n-۲=\frac{۳}{۲}n-۲$$

بنابراین تعداد مقایسات در این حالت دقیقا $\frac{3n}{2}-2$ است. این تعداد در مقایسه با 2(n-1) مقایسه در الگوریتم قبل در حالات بهترین و میانگین حدود 25 درصد صرفه جویی است.

اما از نقاط ضعف الگوریتم فوق، استفاده از پشته برای متغیر های high,low,min,max, lmin, lmax, rmin, rmax و آدرس بازگشت است . برای آرایه n عضوی، $\lfloor\log n\rfloor$+1 سطح فراخوانی خواهیم داشت.

نکته: برای رابطه بازگشتی

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

دیدیم برای حالتی که n توانی از 2 باشد

$$T(n) = \log n + 1 = \theta(\log n)$$

اما در حالت کلی روند اثبات حالت قبل قابل استفاده نیست. چون برای هر n داده شده می توان آن را بین دو توان از 2 محدود کرد می توان همان نتیجه قبل را گرفت به شرطی که تابع T صعودی باشد. این موضوع را با استقرا ثابت می کنیم یعنی برای هر $k \leq n$

$$T(k) \leq T(n).$$

برای n=2 داریم (پایه استقرا)

$$T(1) = 1$$
$$T(2) = T\left(\left\lfloor\frac{2}{2}\right\rfloor\right) + 1 = T(1) + 1 = 1 + 1 = 2.$$

پس

$$T(1) \leq T(2).$$

فرض استقرا: برای هر $m \leq n$ حکم درست است، برای n+1 ثابت می کنیم.

پس برای هر $m \leq n$ اگر $k < m$

$$T(k) \leq T(m).$$

کافیست نشان دهیم

$$T(n) \leq T(n + 1).$$

برای هر $n \geq 1$ داریم

$$\left\lfloor \frac{n}{2} \right\rfloor \leq \left\lfloor \frac{n+1}{2} \right\rfloor \leq n.$$

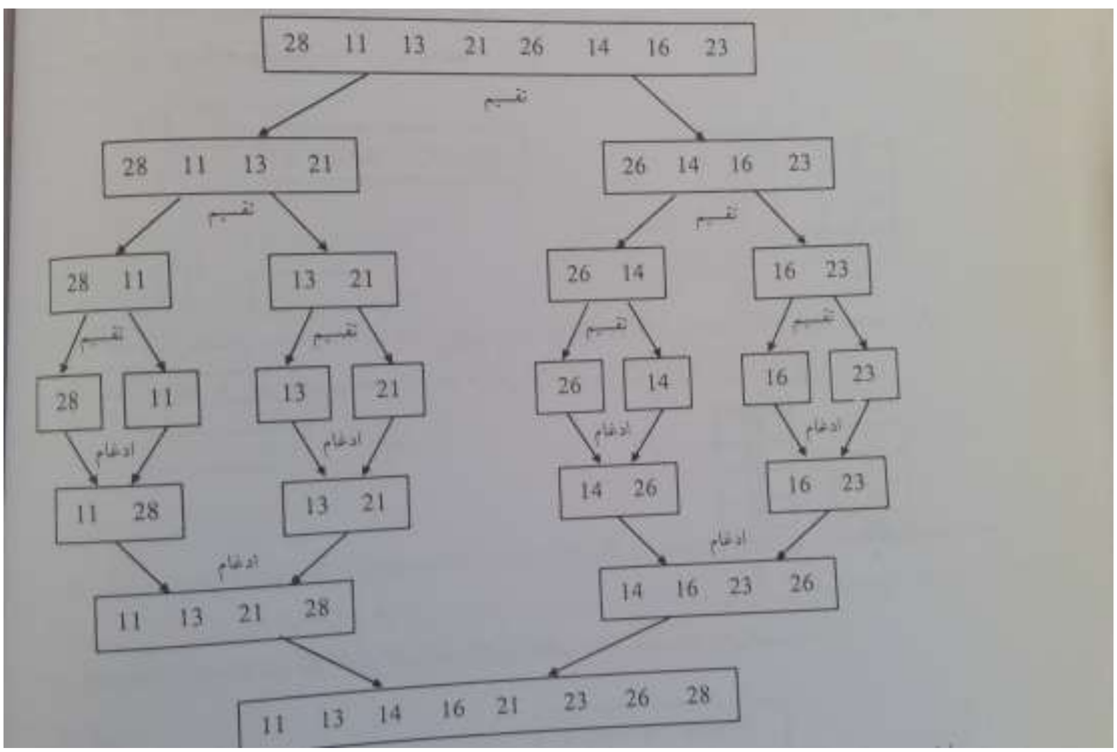
و طبق فرض استقرا

$$T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \leq T\left(\left\lfloor \frac{n+1}{2} \right\rfloor\right).$$

در نتیجه

$$T(n) = \boldsymbol{T}\left(\left\lfloor \frac{\boldsymbol{n}}{\boldsymbol{2}} \right\rfloor\right) + 1 \leq \boldsymbol{T}\left(\left\lfloor \frac{\boldsymbol{n+1}}{\boldsymbol{2}} \right\rfloor\right) + 1 = T(n+1),$$

مرتب سازی ادغامی



```
void mergesort (int n, int S [ ])
{
   int p = ⌊n / 2⌋  ,  m = n − p;          // m و p تقریباً برابر نصف عناصر آرایه S یعنی n/2 هستند
   int A[1 .. p] , B[1 .. m];
   if (n > 1)  {
      Copy S[1]  through S[p]  to A[1]  through A[p];
      Copy S[p+1] through S[n]  to B[1] through B[m];
      mergesort (p,A);
      mergesort (m, B);
      merge (p,m,A,B, S);
   }
}
```

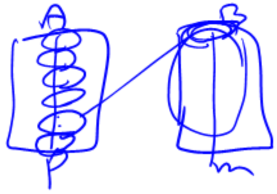
الگوریتم ادغام داخل آن نیز بصورت زیر است:

```
void merge (int p, int m, int A[ ], int B[ ], int S[ ])
{
  int i,j,k;
  i = j = k = 1;
  while (i <= p  && j <= m) {
     if (A[i] < B[j]) {    S[k] = A[i], i++; }
     else {S[k] = B[j];  j++}
     k ++;
  }
                        // اگر عناصر A تمام شد، مابقی عناصر B را در انتهای S قرار بده

  if (i > p)
        while (j <= m)
        {   S[k] = B[j]; k++; j++;}    // اگر عناصر B تمام شد، مابقی عناصر A را در انتهای S قرار بده

  else if (j>m)
      while (i <= p)
        {   S[k] = A[i] ; k ++; i++;}
}
```

عمل اصلی را مقایسه A[i] ,B[j] در نظر می گیریم.



$p-1$
m

بهترین حالت الگوریتم ادغام: این حالت زمانی اتفاق می افتد که در آرایه با طول کمتر همه عناصر از اولین عنصر آرایه با طول بزرگتر کوچکتر باشند. پس تعداد مقایسات $\min(p, m)$ است.

بدترین حالت الگوریتم ادغام: این حالت زمانی اتفاق می افتد که همه عناصر آرایه اول بجز آخرین عنصر از عناصر آرایه دیگر کوچکتر باشند و عنصر آخر از همه آنها بزرگتر باشد. پس تعداد مقایسات $p + m - 1$ است.

پیچیدگی الگوریتم مرتب سازی ادغامی:

عمل اصلی همان مقایسه در الگوریتم ادغام است. در بدترین حالت داریم

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

اگر n توانی از 2 باشد داریم

$$p = \left\lfloor \frac{n}{2} \right\rfloor = \frac{n}{2}, \qquad m = n - p = \frac{n}{2}$$

پس

$$T(n) = 2T\left(\frac{n}{2}\right) + n - 1, T(1) = 0$$

$n^{\log_2 2} = n$

قسمت دوم

از قضیه اصلی نتیجه می شود

$$T(n) = \theta(n \log n)$$

جواب دقیق رابطه فوق نیز بصورت زیر است:

$$T(n) = n \log n - (n - 1).$$

بطور مشابه در بهترین حالت نیز $\theta(n\log n)$ است.

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

اگر n توانی از 2 نباشد داریم

$$T(n) = T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + T(\left\lceil\frac{n}{2}\right\rceil) + n - 1, T(1) = 0$$

در این حالت نیز

$$T(n) = \theta(n\ log\ n)$$

همچنین در حالات بهترین و میانگین نیز این الگوریتم از مرتبه $\theta(n\ log\ n)$ است.

نکاتی در خصوص حافظه مصرفی: در الگوریتم ادغامی در هر مرحله نیاز به حافظه کمکی به اندازه n داریم. چون تعداد مراحل شکستن آرایه $log\ n$ است پس مرتبه فضای مورد استفاده هم $\theta(n\ log\ n)$ است (**تمرین**: الگوریتم را طوری اصلاح کنید که مصرف حافظه آن $\theta(n)$ باشد).

سوال: اگر تقسیم آرایه بصورت k وn-k باشد در خصوص پیچیدگی چه می توان گفت؟

~~For definiteness~~, let us suppose that the $T(1) = 1$ and that if the split is $k + (n-k)$ then the running time is $T(k) + T(n-k) + n$. The recursion for the worst-case running time is therefore

$$T(n) = \begin{cases} \max_{1 \leq k \leq n-1} T(k) + T(n-k) + n & \text{if } n > 1 \\ 1 & \text{if } n = 1. \end{cases}$$

In order to calculate the worst-case complexity, we need both an upper bound and a lower bound. You already gave the lower bound:

$$T(n) \geq T(1) + T(n-1) + n = \underbrace{T(n-1)}_{k=1} + n + 1. = \cdots$$

Iterating this easily yields $T(n) = \Omega(n^2)$.

In the other direction, let us prove that $T(n) \leq n^2$ for all n. The base case is clear. For the inductive case,

$$T(k) + T(n-k) + n \leq k^2 + (n-k)^2 + n \leq 1 + (n-1)^2 + n = n^2 - n + 2 \leq n^2,$$

since $n \geq 2$. It follows that $T(n) = \Theta(n^2)$.

$1 \leq k \leq n-$

min $T(n)$?

min $k^2 + (n-k)^2$

$2k - 2(n-k) = 0 \Rightarrow k = \frac{n}{2}$

حالت میانگین:

Let us now proceed to the average case running time (i.e., the expected running time), which satisfies the recurrence

$$S(n) = \begin{cases} \frac{1}{n-1} \sum_{1 \le k \le n-1} [S(k) + S(n-k)] + n & \text{if } n > 1 \\ 1 & \text{if } n = 1. \end{cases}$$

We can simplify the inductive case to

$$S(n) = \frac{2}{n-1} \sum_{1 \le k \le n-1} S(k) + n.$$

This implies that for $n \ge 2$,

$$\begin{aligned} nS(n+1) &= 2(S(1) + \cdots + S(n)) + (n+1)n, \\ (n-1)S(n) &= 2(S(1) + \cdots + S(n-1)) + n(n-1). \end{aligned}$$

Subtracting gives

$$nS(n+1) - (n-1)S(n) = 2S(n) + 2n \implies nS(n+1) = (n+1)S(n) + 2n \implies \\ \frac{S(n+1)}{n+1} = \frac{S(n)}{n} + \frac{2}{n+1}.$$

This shows that

$$\frac{S(n)}{n} = \frac{2}{n} + \frac{2}{n-1} + \cdots + \frac{2}{2} + \frac{S(1)}{1} = 2H_n - 1 = \Theta(\log n),$$

and so $S(n) = \Theta(n \log n)$.