فصل ۲

روش‌های تقسیم و غلبه

روش‌های تقسیم و غلبه روش‌هایی بازگشتی هستند. ایدهٔ یک روش تقسیم و غلبه به این صورت است که اگر اندازهٔ مساله‌ای (فرض n) بزرگ باشد، آن‌گاه آن را به 2 یا چند زیر مسالهٔ با اندازه‌های کوچک‌تر از n تقسیم می‌کنیم. اگر اندازهٔ این زیر مسایل نیز بزرگ باشند، که حل آن‌ها بدیهی نباشد، باز آن‌ها را به صورت بازگشتی و به همان روال، تقسیم می‌کنیم. تقسیمات آن‌قدر ادامه می‌یابد تا حل زیر مسایل، بدیهی شود. پس از حل زیر مسایل، باید به نحوی آن‌ها را با هم ترکیب کرد. روش تقسیم و غلبه را می‌توان مشابه ساختن یک سیستم پیچیده در نظر گرفت که به صورت یک‌پارچه قابل ساخت نیست. مسلماً این سیستم از اجزای ساده‌تر تشکیل شده است که به راحتی می‌توان آن‌ها را ساخت. بعد از ساخت اجزای ساده‌تر، بدیهی است که برای به دست آوردن کل سیستم و راه‌اندازی آن، باید به نحو مطلوبی قطعات ساخته شده را با هم مونتاژ کرد.

در مسایلی که در این فصل و احیاناً فصول بعدی خواهند آمد، اگر اندازهٔ مسالهٔ داده شده بزرگ باشد، آن را به 2 مسالهٔ کوچک‌تر تقسیم خواهیم کرد. با این توضیح، اگر فرض کنیم که ورودی برای مسالهٔ موردنظر، در آرایهٔ A[1..n] داده شده است، صورت عمومی یک الگوریتم تقسیم و غلبه به شکل زیر خواهد بود:

```
Algorithm D&C ( low , high)        / Divide and Conquer
if Small (low , high)  Then Return  (G (low , high))
  else
    {
        mid=Divide (low , high)
        Return (Combin (D&C(low , mid) , D&C (mid +1 , high))
    }
```

که در آن اندیس mid معمولاً وسط اندیس‌های low و high انتخاب می‌شود. فراخوانی این الگوریتم به صورت D&C (1 , n) خواهد بود.

اگر زمان اجرای به کار رفته توسط زیر الگوریتم Combin را در الگوریتم D&C (1 , n) برابر f(n) فرض کنیم، در این صورت زمان اجرای این الگوریتم را می‌توان به شکل رابطهٔ تقسیم و غلبهٔ زیر بیان کرد:

$$\begin{cases} t(n)=t\left(\left\lceil \frac{n}{2}\right\rceil\right)+t\left(\left\lfloor \frac{n}{2}\right\rfloor\right)+f(n) & \text{برای n های بزرگ} \\ t(n)=g(n) & \text{برای n های کوچک} \end{cases}$$

که در آن $g(n)$ زمان اجرای تابع $G(1\ ,\ n)$ به کار رفته در الگوریتم فوق است. در حالت خاصی که $n=2^m$ باشد، رابطهٔ تقسیم و غلبهٔ بالا به صورت سادهٔ زیر بیان خواهد شد:

$$t(n)=2t\left(\frac{n}{2}\right)+f(n)$$

که با توجه به قضیهٔ اصلی به راحتی قابل حل خواهد بود.

الگوریتم تقسیم و غلبه برای پیدا کردن عنصر ماکزیمم یک آرایه مثل A[1..n]

فرض کنید A[1..n] داده شده و هدف پیدا کردن عنصر ماکزیمم آن است. بدیهی است که اگر n=1 باشد، در این صورت عنصر ماکزیمم همان A[1] است. اگر n=2 باشد، با یک مقایسه بین A[1] و A[2]، عنصر ماکزیمم به دست خواهد آمد (مثل پایهٔ استقراء).

حال فرض کنید که عناصر ماکزیمم را برای زیر آرایهٔ $A\left[1..\frac{n}{2}\right], A\left[\frac{n}{2}+1..n\right]$ پیدا کرده باشیم (مثل فرض استقراء). اگر rmax , lmax به ترتیب این عناصر باشند بدیهی است که عنصر ماکزیمم آرایهٔ A[1..n] (فرض fmax) را می‌توان با یک مقایسه بین rmax , lmax به دست آورد. مطالب گفته شده در الگوریتم Findmax زیر خلاصه شده است:

```
Algorithm FindMax (low , high , fmax)
if (low=high) then fmax=A[low]
   else
      if high = low+1  then
            if  A[low]  >  A [high] then
            fmax=A [low]
         else
            fmax=A[high]
      else
      {
         mid= ⌊(low + high)/2⌋
         FindMax (low , mid , lmax)
         FindMax (mid+1 , high , rmax)
         fmax = Maximum {lmax , rmax}
      }
```

فراخوانی این الگوریتم به صورت $\text{FindMax}(1,n,\text{fmax})$ خواهد بود که عنصر ماکزیمم توسط fmax برگردانیده می‌شود.

برای تحلیل این الگوریتم، اگر برای سهولت فرض کنیم که $n=2^m$، در این صورت تعداد مقایسه‌های لازم برای به دست آوردن عنصر ماکزیمم را می‌توان به صورت رابطهٔ تقسیم و غلبه زیر بیان کرد:

$$\begin{cases} t(n)=2t\left(\frac{n}{2}\right)+1 & n>1 \\ t(1)=0 \end{cases}$$

همان‌گونه که قبلاً هم مشاهده کرده بودیم، از قضیهٔ اصلی نتیجه می‌شود که $t(n)=\theta(n)$. بنابراین تعداد مقایسه‌ها به صورت $t(n)=an+b$ خواهد بود که با اعمال شرایط مرزی داریم $a+b=0$ و چون می‌دانیم که $t(n)=1$، لذا $1=2a+b$ بنابراین $a=1$, $b=-1$ و در نتیجه:

$$\boxed{t(n)=n-1}$$

مثال : برای به دست آوردن همزمان بزرگ‌ترین و کوچک‌ترین عنصر در یک آرایهٔ نامرتب $A[1..n]$، بهترین الگوریتمی که می‌توان ارایه داد، چند مقایسه خواهد داشت؟ (فرض $n=2^m$)

حل: الگوریتم تقسیم و غلبهٔ زیر برای این منظور نوشته شده است:

```
Algorithm FindMinMax ( low , high , fmin , fmax)
    if low=high  Then   fmin=fmax= A[low]
        else
            if  high = low +1  Then
            if A[low]  <  A[high] then
                {fmin=A[low] , fmax =A[high]}
            else
                {fmin=A[high] , fmax=A[low]}
            else
        }
            mid = ⌊(low + high)/2⌋
            FindMinMax (low , mid , lmin , lmax)
            FindMinMax (mid+1, high , rmin , rmax)
            fmax = Maximum (lmax , rmax)
            fmin = Minimum (lmin , rmin)
        }
```

فراخوانی این الگوریتم به صورت FindMinMax (1 , n , fmin , fmax) خواهد بود.

تعداد مقایسه‌های انجام شده با این الگوریتم به صورت رابطهٔ تقسیم و غلبه زیر بیان می‌شود:

$$\begin{cases} t(n)=2t\left(\frac{n}{2}\right)+2 & n>2 \\ t(1)=0\ ,t(2)=1 & n\le 2 \end{cases}$$

با توجه به قضیهٔ اصلی $t(n)=\theta(n)$ و بنابراین فرم عمومی $t(n)$ برای $n>2$ به صورت $t(n) = an + b$ است. چون مقادیر زیر را برای $t(n)$ از روی رابطهٔ تقسیم و غلبه بالا داریم:

$$t(4)=4$$
$$t(8)=10$$

لذا با قرار دادن آن‌ها در $t(n)=an+b$ خواهیم داشت:

$$\begin{matrix} 4=4a+b \\ 10=8a+b \end{matrix} \Rightarrow 6=4a \Rightarrow \boxed{a=\frac{3}{2}}$$

$$b=4-4\times\frac{3}{2}=2 \Rightarrow \boxed{b=-2}$$

و لذا

$$\boxed{t(n)=\frac{3n}{2}-2}$$

مسالهٔ ضرب دو چند جمله‌ای: فرض $A(x)=a_0+a_1x+a_2x^2+.....+a_nx^n$ ، $B(x)=b_0+b_1x+b_2x^2+.....+b_nx^n$ دو چند جمله‌ای از درجهٔ n باشند، هدف به دست آوردن چند جمله‌ای حاصل‌ضرب $C(x)=A(x)B(x)$ است. C(x) یک چند جمله‌ای از درجهٔ 2n به صورت زیر خواهد بود:

$$C(x)=c_0+c_1x+c_2x^2+.....+c_{2n}x^{2n}$$

که در آن $c_k=\sum_{i=0}^{k}a_ib_{k-i}$ $\quad k=0,1,2,....,2n$

با این فرض که ضرایب چند جمله‌ای‌های A(x) , B(x) به وسیلهٔ آرایه‌های a[0..n] و b[0..n] داده شده‌اند، در این صورت کافی است که الگوریتم موردنظر، ضرایب C(x) را به عنوان خروجی ارایه دهد. برای محاسبهٔ ضرایب c_i از روی ضرایب a_i, b_i، سه الگوریتم زیر ارایه می‌شود:

الف) روش معمولی

```
Algorithm_Multy_ Poly (a,b,c)
  for  i=0  to  2n  do  c[i]=0
  for  i=0  to  n  do
       for  j=0  to  n  do
            c[i + j] = c[i + j] + a[i]×b[j]
```

به عنوان نمونه برای

$$A(x)=1+x+3x^2-4x^3$$

$$B(x)=1+2x-5x^2-3x^3$$

و $n=3$ داریم

	0	1	2	3
a[0..3] =	1	1	3	-4
b[0..3] =	1	2	-5	-3

در ابتدا

	0	1	2	3	4	5	6
c[0..6] =	0	0	0	0	0	0	0

	0	1	2	3	4	5	6
برای i=0	1	2	-5	-3	0	0	0
برای i=1	1	3	-3	-8	-3	0	0
برای i=2	1	3	0	-2	-18	-9	0

برای i=3

1	3	0	-6	-26	11	12

در نتیجه

$$C(x)=A(x)B(x)=1+3x-6x^3-26x^4+11x^5+12x^6$$

ب) روش تقسیم و غلبهٔ معمولی

روش را با چند جمله‌ای‌های مثال قبل توضیح می‌دهیم. این دو چند جمله‌ای را می‌توان به صورت‌های زیر نوشت:

$$A(x)=1+x+(3-4x)x^2=A_\ell(x)+x^2A_r(x)$$

$$B(x)=1+2x+(-5-3x)x^2=B_\ell(x)+x^2B_r(x)$$

و لذا

$$A(x)B(x)=\left[A_\ell(x)+x^2A_r(x)\right]\left[B_\ell(x)+x^2B_r(x)\right]$$

$$=A_\ell(x)B_\ell(x)+\left[A_\ell(x)B_r(x)+A_r(x)B_\ell(x)\right]x^2+A_r(x)B_r(x)x^4$$

اما

$$A_\ell(x)B_\ell(x)=\boxed{1+3x+2x^2}$$

$$A_\ell(x)B_r(x)=-5-8x-3x^2$$

$$A_r(x)B_\ell(x)=3+2x-8x^2\Rightarrow$$

$$A_\ell(x)B_r(x)+A_r(x)B_\ell(x)=\boxed{-2-6x-11x^2}$$

$$A_r(x)B_r(x)=(3-4x)(-5-3x)=\boxed{-15+11x+12x^2}$$

$$C(x)=A(x)B(x)=1+3x+2x^2+\left(-2-6x-11x^2\right)x^2+\left(-15+11x+12x^2\right)x^4$$

$$\boxed{C(x)=1+3x-6x^3-26x^4+11x^5+12x^6}$$

در حالت عمومی اگر $n=2^m$ و چند جمله‌ای‌های $A(x)$, $B(x)$ از درجه n-1 باشند:

$$A(x)=a_0+a_1x+a_2x^2+.....+a_{n-1}x^{n-1}$$

$$B(x)=b_0+b_1x+b_2x^2+.....+b_{n-1}x^{n-1}$$

در این صورت اگر $A(x)$ و $B(x)$ را به صورت‌های زیر بنویسیم:

$$A(x)=A_\ell(x)+A_r(x)x^{\frac{n}{2}}=\left(a_0+a_1x+a_2x^2+.....+a_{\frac{n}{2}-1}x^{\frac{n}{2}-1}\right)+\left(a_{\frac{n}{2}}+a_{\frac{n}{2}+1}x+...+a_nx^{\frac{n}{2}-1}\right)x^{\frac{n}{2}}$$

$$=A_\ell(x)+A_r(x)x^{\frac{n}{2}}$$

$$B(x)=B_\ell(x)+B_r(x)x^{\frac{n}{2}}=\left(b_0+b_1x+b_2x^2+.....+b_{\frac{n}{2}-1}x^{\frac{n}{2}-1}\right)+\left(b_{\frac{n}{2}}+b_{\frac{n}{2}+1}x+...+b_nx^{\frac{n}{2}-1}\right)x^{\frac{n}{2}}$$

$$=B_\ell(x)+B_r(x)x^{\frac{n}{2}}$$

در این صورت

$$C(x)=\left[A_\ell(x)+A_r(x)x^{\frac{n}{2}}\right]\left[B_\ell(x)+B_r(x)x^{\frac{n}{2}}\right]$$

$$C(x)=A_\ell(x)B_\ell(x)+\left[A_\ell(x)B_r(x)+A_r(x)B_\ell(x)\right]x^{\frac{n}{2}}+A_r(x)B_r(x)x^n \qquad (۱)$$

بنابراین، مسالهٔ ضرب دو چند جمله‌ای از درجهٔ n-1، به چهار ضرب چند جمله‌ای‌های از درجهٔ $\frac{n}{2}-1$ تبدیل شده است که در صورت لزوم هر کدام از این چندجمله‌ای‌ها را می‌توان به چهار ضرب چند جمله‌ای از درجهٔ $\frac{n}{4}-1$ تبدیل کرد. این روند تا رسیدن به چند جمله‌ای‌های از درجه 1 ادامه خواهد داشت.

مرتبهٔ بزرگی این روش را می‌توان از رابطهٔ تقسیم و غلبهٔ زیر به دست آورد:

$$t(n)=4t\left(\frac{n}{2}\right)+3n$$

که با توجه به قضیهٔ اصلی

$$\boxed{t(n)=\theta(n^2)}$$

ج) روش تقسیم و غلبهٔ سریع

اگر فرض کنیم که

$$R_\ell(x)=A_\ell(x)B_\ell(x)$$
$$R_m(x)=\left[A_\ell(x)+A_r(x)\right]\left[B_\ell(x)+B_r(x)\right]$$
$$R_r(x)=A_r(x)B_r(x)$$

در این صورت جملهٔ وسط عبارت (1) را می‌توان به صورت زیر نوشت:

$$A_\ell(x)B_r(x)+A_r(x)B_\ell(x)=R_m(x)-R_\ell(x)-R_r(x)$$

و در نتیجه

$$C(x)=R_\ell(x)+\left[R_m(x)-R_\ell(x)-R_r(x)\right]x^{\frac{n}{2}}+R_r(x)x^n \qquad (۲)$$

نکته‌ای که برای محاسبه C(x) در فرمول تقسیم و غلبهٔ (2) وجود دارد این است که در این‌جا از 3 ضرب چند جمله‌ای از درجات $\frac{n}{2}-1$ استفاده شده است. بنابراین مرتبهٔ بزرگی روش (2) به صورت زیر است:

$$t(n)=3t\left(\frac{n}{2}\right)+6n$$

و با توجه به قضیه اصلی

$$t(n)=\theta\left(n^{\mathrm{Log}_2^3}\right)=\boxed{\theta\left(n^{1.58}\right)}$$

که سریع‌تر از روش (1) است.

از این روش می‌توان برای ضرب اعداد صحیح خیلی بزرگ استفاده کرد.

مثال : فرض کنید که یک ماشین انتزاعی داریم که در مبنای 10 کار می‌کند و فقط می‌تواند اعداد یک رقمی را در هم ضرب کند، اما برای جمع یا تفریق دو عدد صحیح هر چند رقمی محدودیتی ندارد و ضرب در اعداد 10^x را به راحتی از طریق شیفت عمل می‌کند . برای ضرب دو عدد 1234×5618 حداقل تعداد ضرب‌های به کار رفته توسط این ماشین چقدر است؟

حل: از روش تقسیم و غلبهٔ سریع استفاده می‌کنیم:

$$1234\times5618=\left(12\times10^2+34\right)\left(56\times10^2+18\right)=$$
$$12\times56\times10^4+\left(46\times74-12\times56-34\times18\right)\times10^2+34\times18$$

$$46\times74=\left(4\times10^1+6\right)\left(7\times10^1+4\right)=4\times7\times10^2+\left(10\times11-\boxed{4\times7}-\boxed{6\times4}\right)\times10^1+6\times4$$

$$12\times56=\left(1\times10^1+2\right)\left(5\times10^1+6\right)=1\times5\times10^2+\left(3\times11-\boxed{1\times5}-\boxed{2\times6}\right)\times10^1+2\times6$$

$$34\times18=\left(3\times10^1+4\right)\left(1\times10^1+8\right)=1\times3\times10^2+\left(\boxed{7\times9}-\boxed{3\times1}-\boxed{4\times8}\right)\times10^1+4\times8$$

$$10\times11=\left(1\times10^1+0\right)\left(1\times10^1+1\right)=1\times1\times10^2+\left(\boxed{1\times2}-\boxed{1\times1}-\boxed{0\times1}\right)\times10^1+0\times1$$

$$3\times11=\left(0\times10^1+3\right)\left(1\times10^1+1\right)=0\times1\times10^2+\left(\boxed{3\times2}-\boxed{0\times1}-\boxed{3\times1}\right)\times10^1+1\times3$$

بنابراین تعداد ضرب‌های به کار رفته برابر 13 است. نتیجه هم به صورت زیر خواهد بود:

$$46\times47=3404$$

$$12\times56=672$$

$$34\times18=612$$

$$1234\times5618=672\times10^4+212\times10^3+612=\boxed{6932612}$$

الگوریتم جستجوی دودویی: آرایهٔ مرتب شدهٔ A[1..n] و کلید x مفروض است. هدف این است که ببینیم آیا کلید x در آرایه هست یا خیر. اگر x در آرایه باشد، الگوریتم جستجوی دودویی اندیس آن‌را بر می‌گرداند (جستجوی موفق) و در غیر این صورت اندیس 0 را به عنوان خروجی بیرون می‌دهد (جستجوی ناموفق).

```
Algorithm BinSearch (low , high , x,j)
  if     low ≤ high   then
  {
      mid = ⌊(low + high) / 2⌋
    if  x < A [mid] then  BinSearch (low , mid-1 , x,j)
        else
            if    x=A[mid]   then {j=mid , Return}
                  else BinSearch (mid+1 , high , x,j)
  }
  else {j=0 , Return}
```

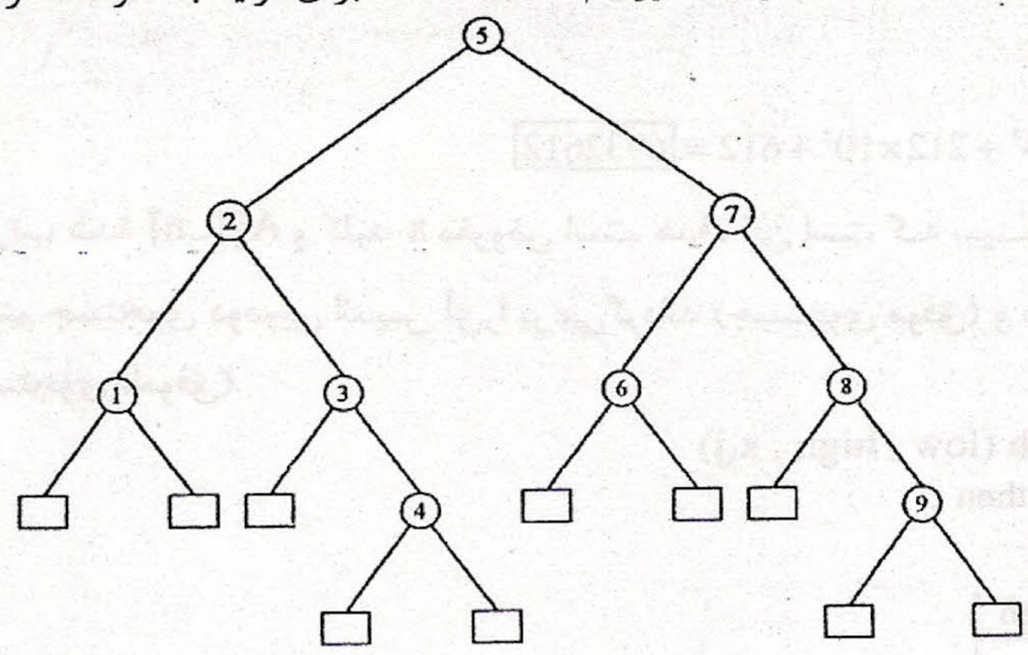
فراخوانی این الگوریتم به صورت BinSearch (1 , n , x , j) خواهد بود.

تحلیل الگوریتم جستجوی دودویی

در این تحلیل فقط دستور مقایسهٔ x با عناصر آرایهٔ A را در نظر می‌گیریم. تعداد مقایسه‌ها برای همهٔ حالات جستجوی موفق و ناموفق برای یک مثال در زیر آمده است:

i	1	2	3	4	5	6	7	8	9	
A[i]	-15	-6	0	7	9	23	54	82	101	
تعداد مقایسه‌ها برای جستجوی موفق	3	2	3	4	1	3	2	3	4	⇒ جمع مقایسه‌ها = 25
تعداد مقایسه‌ها برای جستجوی ناموفق	3	3	3	4	4	3	3	3	4	4 ⇒ جمع مقایسه‌ها =34

مشاهده می‌شود که برای این مثال میانگین تعداد مقایسه‌ها برای جستجوی موفق برابر $\frac{25}{9} = 2.77$ و این میانگین برای جستجوی ناموفق برابر $\frac{34}{10} = 3.4$ است. مراحل مختلف اجرای الگوریتم BinSearch برای آرایهٔ بالا توسط درخت دودویی تصمیم زیر به تصویر کشیده شده است:



درخت دودویی تصمیم

برای مثال بالا مشاهده می‌شود که برای جستجوی موفق حداقل تعداد مقایسه‌های لازم (بهترین حالت) برابر 1، حداکثر تعداد مقایسه‌های انجام شده (بدترین حالت) برابر 4 و میانگین تعداد مقایسه‌ها هم برابر 2.77 است. برای جستجوی ناموفق هم این مقایسه‌ها به ترتیب برابر 3, 4 , 3.4 است.

حال این تحلیل را در حالت عمومی و بر حسب n بیان می‌کنیم.

قضیه : اگر $2^{k-1} \le n < 2^k$ آن‌گاه الگوریتم BinSearch مستلزم حداکثر k مقایسه برای جستجوی موفق و k-1 یا k مقایسه برای جستجوی ناموفق است. به عبارت دیگر زمان جستجوی موفق $O(\log n)$ و زمان جستجوی ناموفق $\theta(\log n)$ است.

بنابراین کافی است که میانگین تعداد مقایسه‌ها را برای حالت جستجوی موفق نیز به دست آوریم. برای این منظور مفاهیم زیر را یادآوری می‌کنیم:

$d(x)$ = مسافت گره x از ریشه = تعداد یال‌های از گره x به ریشه

$I(n)$ = طول مسیر داخلی در یک درخت دودویی = $\sum_{x \text{ گره داخلی}} d(x)$

$E(n)$ = طول مسیر خارجی در یک درخت دودویی = $\sum_{x \text{ گره خارجی}} d(x)$

قضیه ۱: در یک درخت دودویی با n گره داخلی داریم:

$$\boxed{E(n)=I(n)+2n} \qquad (۱)$$

قضیه ۲: در یک درخت دودویی با n گره داخلی داریم:

$$n_0 = \text{تعداد گره‌های خارجی} = n+1$$

با توجه به مثال و تعاریف بالا در حالت عمومی نتایج زیر را برای میانگین تعداد مقایسه‌ها داریم:

$$S(n) = \text{میانگین تعداد مقایسه‌ها برای جستجوی موفق} = \frac{\sum_{\text{x گره داخلی}} (1+d(x))}{n} = 1+\frac{1}{n} \qquad (۲)$$

$$U(n) = \text{میانگین تعداد مقایسه‌ها برای جستجوی ناموفق} = \frac{\sum_{\text{x گره خارجی}} d(x)}{n+1} = \frac{E}{n+1} \qquad (۳)$$

از روابط (1) , (2) , (3) نتیجه می‌شود که $S(n)=\left(1+\frac{1}{n}\right)U(n)-1$

اما از قضیهٔ (1) داشتیم $U(n)=\theta(\log n)$ و بنابراین $S(n)=\theta(\log n)$

نتایج خلاصهٔ زیر را برای الگوریتم جستجوی دودویی داریم:

	بهترین حالت	حالت میانگین	بدترین حالت
جستجوی موفق	$\Omega(1)$	$S(n)=\theta(\log n)$	$O(\log n)$
جستجوی ناموفق	$\Omega(\log n)$	$U(n)=\theta(\log n)$	$O(\log n)$

الگوریتم مرتب‌سازی ادغامی: یکی از روش‌های تقسیم و غلبه برای مرتب کردن عناصر یک آرایه مثل A[1..n] ، الگوریتم مرتب‌سازی ادغامی است. ایدهٔ این الگوریتم به این صورت است که اگر آرایه یک عضو داشته باشد نیازی به مرتب‌سازی نیست. حال اگر آرایه بیش از یک عضو داشته باشد، آن را به دو زیر آرایه با تعداد عناصر تقریباً مساوی افراز می‌کنیم. اگر فرض کنیم که عناصر این دو زیر آرایه را مرتب کرده‌ایم، در این صورت کافی است که برای مرتب کردن همه عناصر آرایهٔ A، عناصر این دو نیم آرایهٔ مرتب شده را با هم ادغام بکنیم. مطالب گفته شده به صورت مشخص در الگوریتم‌های زیر آمده است:

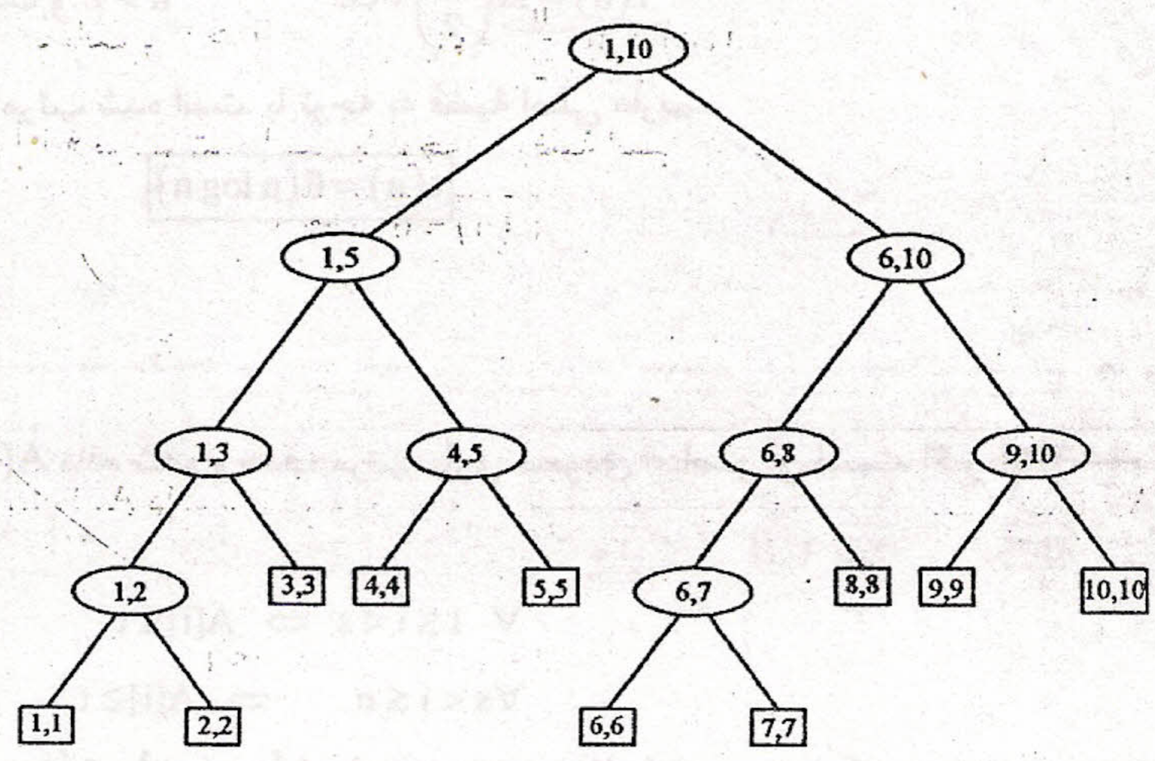
```
Algorithm MergeSort (low , high)
    if  low < high then
    {
        mid = ⌊(low + high) / 2⌋
        MergeSort (low , mid)
        MergeSort (mid +1 , high)
        Merge (low , mid , high)
    }

Algorithm Merge (low , mid , high)
k= low   i=low , j=mid +1
 while (i ≤ mid) and (j ≤ high) do
  {
    if A[i] ≤ A[j]  then
        {B[k] =A[i] , i=i+1}
    else
        {B[k]= A[j] , j=j+1}
   k=k+1
  }
 if i > mid  then
    For  l=j  to  high  do
        {B [k]=A[l], k=k+1}
   else
   For  l=i  to  mid  do
    {B [k] =A [ l] , k=k+1
   }
for  l=low  to  high  do
   A [l] = B[l]
```

فراخوانی این الگوریتم به صورت MergeSort (1 , n) خواهد بود.

مثال : مراحل مختلف اجرای الگوریتم Mergesort (1 , 10) را بر روی داده‌های زیر بنویسید.

A [1..10] = (310 , 285 , 179 , 652 , 325 , 423 , 861 , 254 , 450 , 520)



حل:

مراحل مختلف تقسیم و ادغام در زیر آمده است:

i	1	2	3	4	5	6	7	8	9	10
A[i]	310	285	179	652	351	423	861	254	450	520
	285	310	179	652	351	423	861	254	450	520
	179	285	310	652	351	423	861	245	450	520
	179	285	310	351	652	423	861	254	450	520
	179	285	310	351	652	423	861	254	450	520
	179	285	310	351	652	423	861	254	450	320
	179	285	310	351	652	254	423	861	450	520
	179	285	310	351	652	254	423	861	450	520
	179	285	310	351	652	254	423	450	520	861
	179	254	285	310	351	423	450	520	652	861

تحلیل الگوریتم Mergesort

اگر برای سهولت فرض کنیم که $n = 2^m$ ، در این صورت زمان اجرای الگوریتم مزبور به صورت رابطهٔ تقسیم و غلبهٔ زیر بیان می‌شود:

$$t(n) = 2t\left(\frac{n}{2}\right) + cn \qquad n > 1 \text{ و } c \text{ ثابت}$$

که در آن cn زمان لازم برای ادغام دو نیم آرایهٔ مرتب شده است. با توجه به قضیهٔ اصلی داریم:

$$\boxed{t(n) = \theta(n \log n)}$$

الگوریتم Quicksort

در این‌جا نیز فرض کنید که آرایهٔ نامرتب A[1..n] داده شده و هدف مرتب‌سازی صعودی عناصر آن است. اگر t=A[s] به گونه‌ای انتخاب شود که:

$$\forall\ 1 \le i < s \ \Rightarrow\ A[i] \le t$$

$$\forall s < i \le n \ \Rightarrow\ A[i] \ge t$$

در این صورت کافی است که زیر آرایه‌های $A[1..s-1]$, $A[s+1..n]$ را به صورت مستقل از هم مرتب بکنیم.

الگوریتم Partition که در زیر ارایه می‌شود، با انتخاب عنصر افراز A[m] ، افرازی بر روی عناصر $A[m...p-1]$ برای رسیدن به هدف بالا انجام می‌دهد. فرض بر این است که A [p] بزرگ‌تر از بقیهٔ عناصر A[m...p-1] است. انتخاب A [m] به عنوان عنصر افراز صرفاً برای سهولت در ارایهٔ مطالب است وگرنه هر عنصر از عناصر A[m...p-1] را می‌توان به عنوان عنصر افراز انتخاب کرد. موقع فراخوانی، m=1 ، p-1=n و بنابراین A[n+1] را می‌توان مقدار ∞ قرار داد.

```
Algorithm Partition (m,p)
v=A[m] , i=m
Loop
    Loop   i=i+1    until  A[i]  ≥ v    Repeat
    Loop   p=p-1    until  A[i]  ≤ v    Repeat
    if     i < p    then A[i]  ↔ A[p]
                        else
                            exit
Repeat
A[m] = A [p] , A[p]=v;
```

مثال : مراحل مختلف Partition (1, 10) را بر روی داده‌های زیر بنویسید.

i	1	2	3	4	5	6	7	8	9	10	i	P
A[i]	65	70	75	80	85	60	55	50	45	∞	2	9
	65	45	75	80	85	60	55	50	70	∞	3	8
	65	45	50	80	85	60	55	75	70	∞	4	7
	65	45	50	55	85	60	80	75	70	∞	5	6
	65	45	50	55	60	85	80	75	70	∞	6	5
	60	45	50	55	$\boxed{65}$	85	80	75	70	∞		

```
Algorithm  QuickSort (low , high)
if low < high    then
{
    p=high+1
    Partition (low , p)
    QuickSort (low , p-1)
    QuickSort (p+1 , high)
}
```

تحلیل الگوریتم Quicksort: در تحلیل زیر فقط تعداد دستورهای مقایسه‌ای را خواهیم شمرد. تعداد این مقایسه‌ها در هر فراخوانی Partition برابر p-m+1 است. برای QuickSort (1,n) این مقایسه‌ها را می‌توان به صورت رابطهٔ زیر بیان کرد:

$$t(n)=n+1+t(n_1)+t(n_2) \qquad (۱)$$

که در آن n_1 تعداد عناصر واقع در سمت چپ عنصر افراز و n_2 تعداد عناصر واقع در سمت راست عنصر افراز هستند.

بدترین حالت برای رابطهٔ (1) زمانی اتفاق می‌افتد که $n_1=0$ یا $n_2=0$ باشد. مثلا زمانی که خود آرایه به صورت صعودی مرتب شده باشد، همواره $n_1=1$ خواهد بود. در این حالت داریم:

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

و بنابراین

$$t(n)=(n+1)+n+(n-1)+(n-2)+....2+1=n+1+\frac{n(n+1)}{2}=O(n^2)$$

و بهترین حالت برای رابطهٔ (1) زمانی اتفاق می‌افتد که n_2, n_1 تقریباً مساوی باشند. اگر برای سهولت فرض کنیم که $n = 2^m$ در این صورت خواهیم داشت:

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

که با توجه به قضیهٔ اصلی:

$$\boxed{t(n) = O(n \log n)}$$

حال به بررسی حالت میانگین می‌پردازیم.

بررسی میانگین زمان اجرای الگوریتم Quicksort

برای این بررسی دو محدودیت زیر را قایل هستیم:

الف) عناصر آرایهٔ A متمایز هستند.

ب) عنصر افراز v در الگوریتم Partition به وسیلهٔ یک فرآیند تصادفی انتخاب می‌شود. برای این منظور می‌توان دو دستور سطر اول الگوریتم Partition را با دستورهای زیر جایگزین کرد:

$$i = \text{Random}(m, p-1), v = A[i]\ ,\ \ A[i] = A[m], i = m$$

با فرض‌های مزبور خواهیم داشت:

$$\text{میانگین تعداد مقایسه‌ها} \begin{cases} C_A(n) = n + 1 + \frac{1}{n}\sum_{K=1}^{n}\left(C_A(k-1) + C_A(n-k)\right) \\ C_A(0) = C_A(1) = 0 \end{cases}$$

$$nC_A(n) = n(n+1) + 2\sum_{k=0}^{n-1} C_A(k) \qquad (۱)$$

$$(n-1)C_A(n-1) = n(n-1) + 2\sum_{k=0}^{n-2} C_A(k) \qquad (۲)$$

اگر (2) را از (1) کم کنیم:

$$nC_A(n) - (n-1)C_A(n-1) = 2n + 2C_A(n-1)$$

$$nC_A(n) = (n+1)C_A(n-1) + 2n$$

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

$$\frac{C_A}{n+1} = \left(\frac{C_A(n-2)}{n-1} + \frac{2}{n}\right) + \frac{2}{n+1} = \ldots = \frac{2}{n+1} + \frac{2}{n} + \frac{2}{n-1} + \ldots\ldots + \frac{2}{3} + \frac{C_A(1)}{2}$$

$$\Rightarrow \boxed{C_A(n) = O(n \log n)}$$

یکی از کاربردهای الگوریتم Partition پیدا کردن k امین عنصر کوچک یک آرایه، بدون نیاز به مرتب سازی آن است. این مطلب به وسیلهٔ الگوریتم Selection در زیر توضیح داده شده است.

الگوریتم انتخاب k امین عنصر کوچک آرایه A[1..n] :

فرض کنید که بعد از فراخوانی الگوریتم Partition عنصر افراز در مکان jام قرار بگیرد، در این صورت بدیهی است که j-1 عنصر آرایهٔ کوچک‌تر یا مساوی A[j] است و n-j عنصر باقی‌مانده بزرگ‌تر یا مساوی آن خواهند بود. بنابراین سه حالت زیر امکان‌پذیر است:

اگر k<j آن‌گاه، kامین عنصر کوچک‌تر آرایه در A[1.. j-1] قرار دارد.

اگر k=j آن‌گاه A[j] ، kامین عنصر کوچک‌تر است.

اگر k>j آن‌گاه k امین عنصر کوچک‌تر آرایه برابر (k-j) امین عنصر کوچک‌تر آرایهٔ A[j+1 .. n] خواهد بود.

مطالب گفته شده توسط الگوریتم Selection در زیر ارایه شده است:

```
Algorithm Select (k)
 m=1    ,  r=n+1     A[n+1]= ∞
 Loop
 {
        j=r
        partition (m , j)
        if  k=j   then  Return (A[j])
            else
                if  k < j  then    r=j
                           else      m=j+1
 }
```

برای مثال، در آرایهٔ ارایهٔ شده برای الگوریتم Quicksort اگر k=5 آن‌گاه اولین فراخوانی Partition (1 , 10) کافی خواهد بود، اما اگر k=7 باشد، مراحل مختلف اجرا به صورت زیر خواهند بود:

فراخوانی بعدی، Partition (6,10) خواهد بود:

i	5	6	7	8	9	10	I	J
A[i]	65	85	80	75	70	∞		

65	70	80	75	$\boxed{85}$	∞

چون j=9، لذا مرحلهٔ بعد، Partition (6,9) فراخوانده می‌شود:

$\boxed{70}$	80	75	85

در این مرحله 6 امین عنصرکوچک‌تر پیدا شده (j=6) و بنابراین مرحلهٔ بعد Partition (7 , 9) فراخوانی خواهد شد:

80	75	85

75	$\boxed{80}$	85

J=8 و لذا مرحلهٔ بعد Partition (7 , 8) فراخوانده می‌شود:

$\boxed{75}$	80

الآن j=7 و بنابراین k=7 امین عنصر کوچک‌تر آرایه برابر $A[7]=75$ است.

هر چند بدترین حالت زمان اجرای الگوریتم Select برابر $O(n^2)$ است، ولی می‌توان ثابت کرد که در حالت میانگین مرتبهٔ بزرگی آن O (n) است.

مثال : در الگوریتم پیدا کردن k امین عنصر کوچک‌تر آرایهٔ A[1..n] ، ابتداء همهٔ عناصر را به دسته‌های 5تایی (به جز احتمالاً دستهٔ آخر) تقسیم می‌کنیم. میانهٔ هر دسته را به دست می‌آوریم و سپس میانهٔ میانه‌ها را به صورت بازگشتی پیدا می‌کنیم. این عنصر را به عنوان عنصر افراز انتخاب می‌کنیم و الگوریتم Partition را بر روی عناصر آرایه اعمال می‌کنیم. پس از آن، همین الگوریتم را به صورت بازگشتی (و برای یک k دیگر) بر روی یکی از بخش‌ها اجراء می‌کنیم تا عنصر موردنظر پیدا شود. رابطهٔ بازگشتی بیان‌کنندهٔ زمان اجرای این الگوریتم را به دست آورید.

حل:

برای سهولت درک مطالب، موضوع را با یک مثال به تصویر می‌کشیم(برای n=42)

2	17	1	15	10	11	55	12	
3	18	2	20	20	12	60	15	6
$\boxed{10}$	$\boxed{20}$	$\boxed{30}$	$\boxed{40}$	mm= $\boxed{50}$	$\boxed{60}$	$\boxed{70}$	$\boxed{80}$	$\boxed{90}$
12	25	31	100	51	61	75	90	
21	40	32	200	52	62	80	95	

شکل ۱

مطالب بیان شده را می‌توان با جزییات بیشتر به صورت زیر بیان کرد:

۱ـ n عنصر ورودی را به $\left\lceil \frac{n}{5} \right\rceil$ دسته 5 تایی ، به جز احتمالاً دسته آخر که شامل (n mod 5) عضو باشد، تقسیم می‌کنیم. O(n)

۲ـ میانهٔ هر کدام از این $\left\lceil \frac{n}{5} \right\rceil$ دسته را می‌توان به وسیلهٔ یکی از الگوریتم‌های مرتب‌سازی، مثل مرتب‌سازی درجی به دست آورد. بعد از مرتب کردن عناصر هر دسته، که هر کدام حداکثر 5 عنصر دارد، عنصر میانه به راحتی به دست می‌آید. در شکل بالا، میانهٔ هر دسته به وسیلهٔ عدد داخل مربع مشخص شده است. O(n)

۳ـ الگوریتم Select را به صورت بازگشتی برای پیدا کردن میانهٔ میانه‌ها mm از میان $\left\lceil \frac{n}{5} \right\rceil$ میانه اعمال می‌کنیم. $t\left(\left\lceil \frac{n}{5} \right\rceil\right)$

۴ـ الگوریتم Partition را با انتخاب میانهٔ میانه‌ها به عنوان عنصر افراز فرا می‌خوانیم. O(n)

۵ـ اگر k=j آن‌گاه mm برگردانیده می‌شود، در غیر این صورت الگوریتم Select را به صورت بازگشتی برای پیدا کردن kامین عنصر کوچک‌تر عناصر چپ j، در حالت k<j یا برای پیدا کردن (k-j) امین عنصر کوچک‌تر عناصر راست j، در حالت k>j اعمال می‌کنیم.

با این توضیحات، برای تحلیل کامل الگوریتم و به دست آوردن رابطهٔ بازگشتی خواسته شده کافی است که فقط مرحلهٔ 5 این الگوریتم را بررسی کنیم. حداقل نصف میانه‌های پیدا شده در مرحلهٔ 2، بزرگ‌تر یا مساوی میانهٔ میانه‌ها(mm) هستند. بنابراین، حداقل نصف این $\left\lceil \frac{n}{5} \right\rceil$ دسته، به جز دسته‌ای که شامل mm است و احیاناً دسته‌ای که کمتر از 5 عضو دارد، شامل 3 عنصر بزرگ‌تر یا مساوی mm هستند (برای درک بهتر مطلب به شکل 1 مراجعه کنید). با کنار گذاشتن دو دستهٔ مزبور، نتیجه این می‌شود که تعداد عناصری که بزرگ‌تر یا مساوی mm هستند برابر است با:

$$3\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{5} \right\rceil \right\rceil - 2\right) \geq \frac{3n}{10} - 6$$

به طریق مشابه می‌توان نشان داد که تعداد عناصری که کوچک‌ترین یا مساوی mm هستند، برابر حداقل $\frac{3n}{10} - 6$ است. بنابراین در

بدترین حالت، الگوریتم Select در مرحلهٔ 5، به صورت بازگشتی حداکثر بر روی $n-\left(\frac{3n}{10}-6\right)=\frac{7n}{10}+6$ عنصر فراخوانده خواهد شد.

بنابراین رابطهٔ بازگشتی موردنظر به صورت زیر خواهد بود:

$$\boxed{t(n)=t\left(\left\lceil\frac{n}{5}\right\rceil\right)+t\left(\frac{7n}{10}+6\right)+O(n)} \qquad n\geq 140$$

برای توضیح این‌که چرا $n\geq 140$ به مرجع 3 مراجعه کنید.

مسالهٔ ضرب دو ماتریس

فرض $A=\left[a_{ij}\right], B=\left[b_{ij}\right]$ دو ماتریس به ترتیب با اندازه‌های $n\times p$ و $p\times m$ باشند، می‌دانیم که $C=A\times B=\left[C_{ij}\right]$ یک ماتریس $n\times m$ خواهد بود. برای محاسبهٔ عناصر C_{ij} سه روش زیر ارایه می‌شود:

الف) روش معمولی

```
Algorithm Matrix_ Multiplication (a , b , c)
  for  i=1    to    n    do
     for =  j=1   to   m     do
  {
       C[i , j] =0
       for  k=1   to    p    do
            C [i , j] =C [i ,j] +a[i ,k]*b[k , j]
  }
```

بدیهی است که مرتبهٔ بزرگی الگوریتم مزبور $\theta(nmp)$ است که در حالت خاصی که ماتریس‌ها مربع باشند، $n=m=p$ ، این مرتبهٔ بزرگی برابر $\boxed{\theta(n^3)}$ خواهد بود.

ب) روش تقسیم و غلبهٔ معمولی

برای سهولت فرض می‌کنیم که $n=m=p=2^r$

اگر ماتریس‌های A , B را به صورت زیر، به چهار زیر ماتریس با اندازه‌های $\frac{n}{2}\times\frac{n}{2}$ تقسیم کنیم:

$$A=\left[\begin{array}{c|c}A_{11} & A_{12}\\ \hline A_{21} & A_{22}\end{array}\right] \qquad B=\left[\begin{array}{c|c}B_{11} & B_{12}\\ \hline B_{21} & B_{22}\end{array}\right]$$

در این صورت اگر فرض کنیم که:

$$C=AB=\left[\begin{array}{c|c}C_{11} & C_{12}\\ \hline C_{21} & C_{22}\end{array}\right]$$

می‌توان نشان داد که:

$$\begin{cases}C_{11}=A_{11}B_{11}+A_{12}B_{21}\\ C_{12}=A_{11}B_{12}+A_{12}B_{22}\\ C_{21}=A_{21}B_{11}+A_{22}B_{21}\\ C_{22}=A_{21}B_{12}+A_{22}B_{22}\end{cases} \qquad (1)$$

به عبارت دیگر ما مسالهٔ ضرب دو ماتریس $n\times n$ را به مسالهٔ ضرب ماتریس‌های با اندازهٔ $\frac{n}{2}\times\frac{n}{2}$ تبدیل کرده‌ایم، که هر کدام از آن‌ها تا رسیدن به ماتریس‌های به اندازه‌های 1×1 تقسیم خواهند شد. بنابراین رابطهٔ تقسیم و غلبه برای مرتبهٔ بزرگی این روش به صورت

زیر بیان می‌شود:

$$t(n)=8t\left(\frac{n}{2}\right)+4\frac{n^2}{4}$$

که در آن n^2 تعداد اعمال جمع لازم برای 4 جمع ماتریسی با اندازه‌های $\frac{n}{2}\times\frac{n}{2}$ است.

با توجه به قضیهٔ اصلی $\boxed{t(n)=O(n^3)}$

که مرتبهٔ بزرگی آن مشابه روش معمولی است.

ج) روش تقسیم و غلبهٔ سریع (روش استراسن)

آقای استراسن نشان داد که C_{ij} , $i,j=1,2$ ارایه شده در رابطهٔ (1) را می‌توان به صورت زیر محاسبه کرد:

$$\begin{cases} C_{11}=P+S-T+V \\ C_{12}=R+T \\ C_{21}=Q+S \\ C_{22}=P+R-Q-U \end{cases}$$

که در آن

$$P=(A_{11}+A_{22})(B_{11}+B_{22})$$
$$Q=(A_{21}+A_{22})B_{11}$$
$$R=A_{11}(B_{12}-B_{22})$$
$$S=A_{22}(B_{21}-B_{11})$$
$$T=(A_{11}+A_{12})B_{22}$$
$$U=(A_{21}-A_{11})(B_{11}+B_{12})$$
$$V=(A_{12}-A_{22})(B_{21}+B_{22})$$

نکتهٔ موجود در روش استراسن این است که در آن به جای 8 ضرب ماتریسی از 7 ضرب ماتریسی استفاده شده است. هر چند تعداد اعمال جمع ماتریسی از 4 به 18 افزایش پیدا کرده، ولی همان‌گونه که رابطهٔ تقسیم و غلبهٔ زیر نشان می‌دهد، مرتبهٔ بزرگی الگوریتم استراسن نسبت به روش تقسیم و غلبهٔ معمولی نسبتاً پایین است:

$$t(n)=7t\left(\frac{n}{2}\right)+18\frac{n^2}{4}$$

که اگر از قضیه اصلی استفاده کنیم:

$$t(n)=\theta\left(n^{\log_2^7}\right)\boxed{=\theta\left(n^{2.81}\right)}$$

برای مشاهدهٔ اختلاف بین $n^3, n^{2.81}$ ، اگر $n=1000$ باشد، آن‌گاه $n^3=1000000$ ولی $n^{2.81}=416870$