فصل دوم

روش‌های تقسیم و غلبه

مقدمه

بسیاری از مسائل کاربردی کامپیوتری با روش تقسیم و حل (Divide & Conquer) قابل حل هستند. در این روش، ابتدا مساله بزرگ به زیر مسائل کوچکتر تقسیم می‌شود و پس آن زیر مسائل کوچکتر نیز در صورت لزوم به زیر مسائل کوچکتری تقسیم می‌شوند و در نهایت با حل مسائل کوچک، مساله اصلی حل می‌شود. در واقع روش تقسیم و غلبه، روشی از بالا به پایین (top-down) می‌باشد. روش تقسیم و غلبه، بازگشتی است و معمولاً یک مسئله کوچک ممکن است چندین بار حل شود. یعنی این روش‌ها معمولاً از نظر زمان و حافظه بهینه نیستند ولی مزیت آنها کد نویسی راحت آنهاست.

لازم به ذکر است که تمام مسائل با روش تقسیم و غلبه قابل حل نیستند، مثلاً نمی‌توان برای یافتن جواب‌های یک دستگاه ۱۰۰ معادله ۱۰۰ مجهول آن را به دو دستگاه ۵۰ معادله ۵۰ مجهول تبدیل کرد؛ ولی مثلاً می‌توان برای یافتن ماکزیمم یک آرایه ۱۰۰ عنصری، آرایه را به ۲ آرایه ۵۰ عنصری تقسیم کرده، ماکزیمم هر قسمت را یافته سپس با مقایسه نتایج، ماکزیمم را پیدا کنیم.

می توان الگوریتم کلی تقسیم و حل را به صورت زیر نوشت:

۱- اگر مسئله به اندازه کافی کوچک است و می‌توان آن را حل کرد، حل کن.

۲- اگر مسئله بزرگ است آن را به مسائل کوچک تقسیم کن و تقسیم را ادامه بده.

۳- مسائل کوچکتر را حل کن (مشابه مساله بزرگ)

۴- مسائل حل شده کوچک را با هم ترکیب کن تا مساله بزرگ حل شود.

الگوریتم کلی تقسیم و حل به صورت شبه کد، در الگوریتم ۱ نوشته شده است:

الگوریتم ۱: الگوریتم انتزاعی روش تقسیم و غلبه

```
Algorithm DandC (Low , high)
   if small (low , high) then return g (Low , high)
      else {
         mid = Divide (Low , high)
         return combine (DandC (Low , mid) , DandC (mid + 1 , high))
   }
```

در این الگوریتم تابع Small مشخص می‌کند آیا مساله به اندازه کافی کوچک هست که بتوان آن را بدون شکستن حل کرد یا خیر. اگر جواب مثبت باشد، تابع g فراخوانی می‌شود در غیر این صورت عمل تقسیم انجام می‌شود.

در ضمن اگر فرض کنیم ورودی در آرایه A[۱..n] باشد، فراخوانی باید به صورت DandC(۱ و n) صورت گیرد.

اگر زمان اجرای الگوریتم Combine در الگوریتم ۱، f(n) فرض شود آنگاه زمان اجرای الگوریتم برابر است با:

$$T(n)=\begin{cases} 2T(\frac{n}{2})+f(n) & \text{برای n های بزرگ} \\ g(n) & \text{برای n های کوچک} \end{cases}$$

در T(n) فرض شده است که n توانی از ۲ است و همچنین در هر بار تقسیم، آرایه دقیقاً نصف می‌شود. و همچنین g(n) زمان تابع g می‌باشد

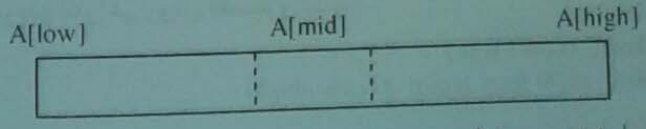
الگوریتم‌هایی که در این بخش بررسی می‌شوند عبارتند از:

۱- جستجوی دودویی
۲- بزرگترین و کوچکترین عنصر آرایه
۳- ضرب چند جمله‌ای‌ها
۴- ضرب اعداد بزرگ
۵- ضرب دو ماتریس
۶- مرتب سازی سریع
۷- مرتب‌سازی ادغامی
۸- یافتن کوچکترین (بزرگترین) کلید دوم و یافتن کوچکترین (بزرگترین) کلید i ام.

۱- جستجوی دودویی

آرایه مرتب شده A به صورت صعودی و کلید x مفروض‌اند. هدف جستجوی x در آرایه است. اگر x در آرایه باشد، اندیس آن را برمی گردانیم (اگر x=A[j] ، j را برمی گردانیم) اگر x در آرایه نباشد، خروجی الگوریتم صفر است (هنگام فراخوانی آرایه را A[۱..n] فرض می‌کنیم).

برای جستجو ابتدا x را با عنصر وسط آرایه مقایسه می‌کنیم. سه حالت پیش می‌آید:



اندیس وسط $mid = \left\lfloor \frac{Low + high}{2} \right\rfloor$

mid را برگردان

if (x = A[mid]) ⇒

if (x > A[mid]) ⇒ جستجو را از mid+۱ تا high با همین روش انجام بده

if (x < A[mid]) ⇒ جستجو را از Low تا mid-۱ با همین روش انجام بده

این عمل تا زمانی انجام می‌شود که یا عنصر پیدا شود و یا اینکه به وضعیتی برسیم که مطمئن شویم عنصر در آرایه نیست.

مثال: در آرایه ۹ عنصری زیر مشخص کنید برای یافتن هر عنصر چند مقایسه انجام می‌شود. و میانگین مقایسه‌ها چند تاست؟

A[۱]	A[۲]	A[۳]	A[۴]	A[۵]	A[۶]	A[۷]	A[۸]	A[۹]
۱۰	۱۲	۱۵	۲۰	۲۷	۳۰	۴۰	۵۰	۶۰

پاسخ: اگر x=۲۷ آنگاه با ۱ مقایسه اندیس آن یعنی ۵ برمی گردد.

اگر x=۱۲ ابتدا x با A[۵] یعنی ۲۷ مقایسه می‌شود و چون ۱۲<۲۷ مقایسه در قسمت

A[۱]			A[۴]
۱۰	۱۲	۱۵	۲۰

انجام می‌شود که حال با A[۲] مقایسه می‌شود $\left(\left\lfloor\frac{۱+۴}{۲}\right\rfloor = ۲\right)$

و ۱۲ پیدا می‌شود. به همین ترتیب به جدول زیر می‌رسیم:

عنصر	۱۰	۱۲	۱۵	۲۰	۲۷	۳۰	۴۰	۵۰	۶۰
تعداد مقایسه	۳	۲	۳	۴	۱	۳	۲	۳	۴

بنابراین میانگین مقایسه‌ها برابر است با:

$$\frac{۳+۲+۳+۴+۱+۳+۲+۳+۴}{۹} = \frac{۲۵}{۹}$$

این مثال جستجوی موفق را نشان داد و میانگین مقایسه‌ها برای جستجوی موفق در این مثال $\frac{۲۵}{۹}$ شد.

مثال: در مثال قبلی مشخص کنید برای یافتن هر یک از عناصر زیر در آرایه A، با چند مقایسه اعلام می‌شود این عناصر در آرایه نیستند (جستجوی ناموفق):

۸ ۱۱ ۱۳ ۱۷ ۲۲ ۲۹ ۳۵ ۴۵ ۵۵ ۶۵

پاسخ: در جستجوی ناموفق، بین هر دو عنصر آرایه، یک عنصر فرض می‌کنیم و جستجو را انجام می‌دهیم. گر x=۸ باشد ابتدا x با A[۵] سپس با A[۲] و سپس با A[۱] مقایسه می‌شود. یعنی با ۳ مقایسه اعلام می‌شود x=۸ در آرایه نیست به همین ترتیب جدول زیر بدست می‌آید:

عنصر	۸	۱۱	۱۳	۱۷	۲۲	۲۹	۳۵	۴۵	۵۵	۶۵
تعداد مقایسه	۳	۳	۳	۴	۴	۳	۳	۳	۴	۴
اندیس‌های مقایسه شده به ترتیب	۱و۲و۵	۱و۳و۵	۳و۲و۵	۴و۳و۲و۵	۵و۳و۲و۵	۶و۷و۵	۶و۷و۵	۸و۷و۵	۹و۸و۷و۵	۹و۸و۷و۵

بنابراین میانگین تعداد مقایسه‌ها برای جستجوی ناموفق برابر است با:

$$\frac{۳+۳+۳+۴+۴+۳+۳+۳+۴+۴}{۱۰} = \frac{۳۴}{۱۰}$$

نکته ۱: *اگر* $s(n)$ *میانگین تعداد مقایسه‌ها برای جستجوی موفق در آرایه* n *عنصری و* $u(n)$ *میانگین مقایسه‌ها برای جستجوی ناموفق باشد آنگاه:*

$$s(n) = (۱+\frac{۱}{n})u(n) - ۱$$

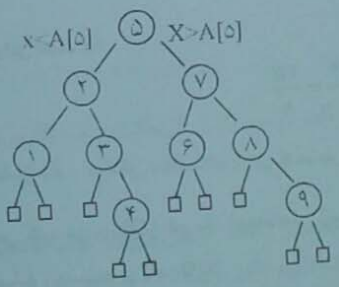
مثلاً با توجه به مثال‌های قبل، $s(۹) = \frac{۲۵}{۹}$ و $u(۹) = \frac{۳۴}{۱۰}$ که تساوی صحیح است.

نکته ۲: *حداکثر تعداد مقایسه‌ها با روش دو دویی در آرایه* n *عنصری برابر است با:* $\lfloor Lgn \rfloor + ۱$ *یا* $\lceil \lg(n+۱) \rceil$.

نکته ۳: *در جستجوی ناموفق برای آرایه* n *عنصری تعداد مقایسه‌ها یا* $\lfloor Lgn \rfloor$ *و یا* $\lfloor Lgn \rfloor + ۱$ *می‌باشد.*

اثبات نکته ۱

می توان جستجوی دو دویی را با یک درخت دو دویی نشان داد. مثلاً برای آرایه ۹ عنصری $A[۱..۹]$ می‌توان درخت زیر را رسم کرد:



در این درخت (درخت تصمیم)، داخل هر نود مقدار mid نوشته شده است. مثلاً ابتدا با $A[۵]$ مقایسه می‌کنیم اگر کمتر بود به سمت چپ می‌رویم و با $A[۲]$ مقایسه می‌کنیم و اگر بیشتر بود به راست می‌رویم و با $A[۷]$ مقایسه می‌کنیم و به همین ترتیب وقتی به برگ‌ها (گره‌های مستطیلی شکل) برسیم یعنی جستجوی ناموفق و اگر به گره‌های داخلی (گره‌های دایره‌ای شکل) کار تمام شود یعنی جستجوی موفق.

$d(x)$ را برابر مسافت از ریشه به گره x تعریف می‌کنیم مثلاً $d(۵)=۰$ یا $d(۱)=۲$ و ...

I را برابر مجموع مسافت‌های گره‌های داخلی تعریف می‌کنیم:

$$(۱)\ I = \sum_{x \text{ گره داخلی}} d(x)$$

E را برابر مجموع طول مسیرهای خارجی تعریف می‌کنیم

$$(۲)\ E = \sum_{x \text{ برگ}} d(x)$$

می توان نشان داد در درخت دودویی $\boxed{E = I + 2n}$ می‌باشد (n تعداد گره‌های داخلی). اگر s(n) میانگین تعداد مقایسه‌ها در جستجوی موفق باشد آنگاه (۳) $\boxed{s(n) = \frac{\sum_{x \text{ گره داخلی}} (d(x)+1)}{n}}$ و اگر u(n) میانگین تعداد مقایسه‌ها در جستجوی ناموفق باشد (۴) $\boxed{u(n) = \frac{\sum_{x \text{ گره خارجی}} d(x)}{n+1}}$ می‌باشد.

از روابط ۱ و ۲ و ۳و ۴ نتیجه می‌شود:

$$s(n) = \frac{I+n}{n} \quad , \quad u(n) = \frac{E}{n+1}$$

$$S(n) = (1 + \frac{1}{n})\, u(n) - 1$$

نکته ۴: *جستجوی دو دویی در حالت ناموفق از مرتبه* $\theta(Lgn)$ *است و در حالت موفق در بهترین حالت از مرتبه* $\theta(1)$ *و در حالت متوسط و بدترین حالت* $\theta(Lgn)$ *می‌باشد.*

نکته ۵: *اگر* $W(n)$ *زمان اجرا (تعداد اجرا) در الگوریتم جستجوی دو دویی در بدترین حالت باشد آنگاه با یک مقایسه، عناصر نصف می‌شوند و می‌توان نوشت* $w(n) = w(\frac{n}{2}) + 1$ *و* $w(1)=1$ *(n توانی از ۲) که نتیجه می‌شود* $w(n)=Lgn+1$ *و اگر n توانی از ۲ نباشد آنگاه* $w(n) = \lfloor Lg\, n \rfloor + 1$

الگوریتم ۲- جستجوی دودویی

```
Algorithm Bsearch (A, Low, high, x)
if Low = high then
    if x = A [Low] then return Low else return 0
else {
    mid = ⌊(Low + high) / 2⌋
    if x = A [mid] then return mid
        else if x < A [mid] then return Bsearch (A, Low, mid-1, x)
        else return Bsearch (A, mid+1, high, x)
}
```

اگر آرایه $A[1..n]$ باشد فراخوانی الگوریتم باید Bsearch (A, 1, n, x) باشد.

۲- بزرگترین و کوچکترین عنصر آرایه

الگوریتم غیر تقسیم و غلبه برای یافتن Min و Max در آرایه A[۱..n] به صورت زیر است:

الگوریتم ۳- الگوریتم غیر تقسیم و غلبه یافتن بزرگترین و کوچکترین

```
Algorithm simple_Max_Min (A , n , max , min)
max = min = A[۱]
for i = ۲ to  n  do
{
   if  A[i] > max  then  max = A[i]
   if  A[i] < min  then  min = A[i]
}
```

در الگوریتم ۳ تعداد مقایسه‌ها ۲(n − ۱) می‌باشد. با توجه به اینکه مقایسه A[i] < min فقط وقتی لازم است که A[i]>max نباشد می‌توان داخل حلقه را به صورت زیر نوشت:

```
if  A[i] > max  then  max = A[i]
   elseif  A[i] < min  then  min = A[i]
```

بنابراین در بهترین حالت (آرایه به صورت صعودی مرتب باشد) تعداد مقایسه‌ها n-۱ و در بدترین حالت (آرایه نزولی مرتب باشد) تعداد مقایسه‌ها ۲(n-۱) می‌باشد.

حال می‌خواهیم یک روش تقسیم و غلبه ارائه دهیم. برای این منظور آرایه را دو قسمت می‌کنیم و در هر قسمت min و max را می‌یابیم و سپس با ۲ مقایسه min و max نهایی را بدست می‌آوریم. مثلاً آرایه

۵	۷	۴	۲۰	۳	۱۲	۸	۹

را در نظر بگیرید. آن را به آرایه‌های

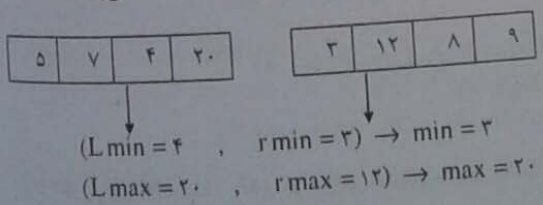
۵	۷	۴	۲۰

و

۳	۱۲	۸	۹

تقسیم می‌کنیم. min و max نیمه اول عبارت است از: Lmin=۴ و Lmax=۲۰ و Min و Max نیمه دوم عبارت است از: rmin=۳ و rmax=۱۲.

حال با مقایسه rmin و Lmin، min=۳ می‌باشد و با مقایسه rmax و Lmax، max=۲۰ می‌باشد:



اگر $T(n)$ تعداد مقایسه‌ها برای آرایه n عنصری باشد آنگاه:

$$T(n)=\begin{cases} T\left(\left\lfloor \frac{n}{۲} \right\rfloor\right)+T\left(\left\lceil \frac{n}{۲} \right\rceil\right)+۲ & , \quad n>۲ \\ ۱ & , \quad n=۲ \\ ۰ & , \quad n=۱ \end{cases}$$

اگر فرض کنیم n توانی از ۲ می‌باشد $(n=۲^k)$ آنگاه:

$$T(n)=\begin{cases} ۲T\left(\frac{n}{۲}\right)+۲ & , \quad n>۲ \\ ۱ & , \quad n=۲ \\ ۰ & , \quad n=۱ \end{cases}$$

الگوریتم ۴، الگوریتم یافتن Min و Max را به صورت بازگشتی نشان می‌دهد:

```
Algorithm maxmin (low, high, max, min)
if Low=high then max=min=A[Low]                    // آرایه تک عنصری است //
else if high = low + ۱  then
// آرایه دو عنصری است //
{
  if  A[low] < A[high]  then
   {
     max = A[high] ; min = A[low]
   }
     else
      {
        max = A[low] ; min = A[high]
      }
     }
Else                                               // آرایه بیش از ۲ عنصری است//
   {
     mid = ⌊(Low + high)/۲⌋
     maxmin (Low, mid, Lmax, Lmin)
     maxmin (mid+۱, high, rmax, rmin)
     if Lmax > rmax  then  max = Lmax
        else  max = rmax
     if  Lmin < rmin  then  min = Lmin
        else  min = rmin
     }
```

الگوریتم ۴- الگوریتم تقسیم و غلبه یافتن کوچکترین و بزرگترین عضو یک آرایه. فراخوانی باید به صورت maxmin(۱,n,Max,Min) انجام شود (برای آرایه A[۱..n])

الگوریتم غیر تقسیم و غلبه دیگری می‌توان ارائه داد به این صورت که عناصر آرایه را جفت کنیم (به شرطی که n زوج باشد) و در هر جفت Min و Max را بیابیم که نیاز به $\frac{n}{۲}$ مقایسه هست. حال $\frac{n}{۲}$ عنصر کوچک وجود دارد و $\frac{n}{۲}$ عنصر بزرگ. با $\frac{n}{۲}-۱$ مقایسه بین عنصرهای کوچک،Min نهایی و با $\frac{n}{۲}-۱$ مقایسه بین عنصرهای بزرگ،Max نهایی پیدا می‌شود یعنی تعداد مقایسه‌ها $\frac{۳n}{۲}-۲=\frac{n}{۲}+\frac{n}{۲}-۱+\frac{n}{۲}-۱$ خواهد بود. اگر n فرد باشد می‌توان نشان داد تعداد مقایسه‌ها $\frac{۳n}{۲}-\frac{۳}{۲}$ خواهد بود (n−۱ عنصر آخر را جفت کنید و عنصر اول را هم با minها و هم با maxها در نظر بگیرید که تعداد مقایسه‌ها $\frac{۳n-۳}{۲}=\frac{n-۱}{۲}+۲(\frac{n-۱}{۲})$ است).

الگوریتم ۵- یافتن Min و Max آرایه A[۱۰۰n] با جفت کردن کلیدها (n زوج)

```
if  (A[۱] < A[۲])
  {
     min = A[۱];
     max = A[۲];
  }
else
  {
     min = A[۲];
     max = A[۱];
  }
for(i = ۳;i <= n − ۱;i = i + ۲)
  {
if(A[i] > A[i + ۱]) swap(A[i], A[i + ۱]);
if (A[i] < min) min = A[i];
if(A[i + ۱] > max) max = A[i + ۱];
  }
```

لم ۱: *هر الگوریتمی که بتواند کوچکترین و بزرگترین عناصر یک آرایه n عنصری را فقط با مقایسه پیدا کند، حداقل تعداد مقایسه‌ها برابر است با T(n) که:*

$$T(n)=\begin{cases}\frac{۳n}{۲}-۲ & \text{اگر n زوج باشد}\\ \frac{۳n}{۲}-\frac{۳}{۲} & \text{اگر n فرد باشد}\end{cases}$$

۳- ضرب چند جمله‌ای‌ها

می‌خواهیم دو چند جمله‌ای درجه n-۱ به نامهای p(x) و Q(x) را در هم ضرب کنیم:

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

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

یعنی می‌خواهیم $R(x) = p(x).Q(x)$ را محاسبه کنیم. R(x) یک چند جمله‌ای درجه ۲n-۲ می‌باشد:

$$R(x) = C_0 + C_1x + C_2x^2 + ... + C_{2n-2}x^{2n-2}$$

که در آن داریم:

$$C_0 = a_0b_0 \ , \ C_1 = a_0b_1 + a_1b_0 \ , \ C_2 = a_2b_0 + a_1b_1 + a_0b_2 \ , ...$$

به طور کلی:

$$C_k = \sum_{i=0}^{k} a_i\, b_{k-i} = \sum_{i+j=k} a_i\, b_j \quad , \quad k = 0,1,2,...,2n-2$$

ضرایب چند جمله‌ای‌های P(x) و Q(x) و R(x) را به ترتیب توسط آرایه‌های A[0..n-1] و B[0...n-1] و C[0..2n-2] نمایش می‌دهیم. هدف یافتن آرایه C می‌باشد. سه روش ارائه می‌شود:

روش ۱. روش معمولی ضرب

الگوریتم ۶ نشان می‌دهد که چگونه از روی آرایه‌های ورودی A و B آرایه خروجی C محاسبه می‌شود.

الگوریتم ۶ - ضرب دو چند جمله‌ای درجه n-۱ باروش معمولی.

```
Algorithm Simple _ Mult (A,B,C)
1) for  i = 0 to 2(n-1)  do
       C[i] = 0
2) for  i = 0 to n-1  do
3) for  j = 0 to n-1  do
4)  C[i+j] = C[i+j] + A[i] * B[j]
```

مرتبه زمانی این الگوریتم $\theta(n^2)$ است.

مثال: مراحل مختلف الگوریتم ۶، برای چند جمله‌ای‌های زیر را نشان دهید:

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

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

پاسخ: در این مثال ۳=۱-n می‌باشد و داریم:

	0	1	2	3
A[0..3] =	1	1	3	-4

	۰	۱	۲	۳
B[۰..۳] =	۱	۲	-۵	-۳

بعد از اجرای حلقه ۱ آرایه C تماماً صفر است:

	۰	۱	۲	۳	۴	۵	۶
C[۰..۶] =	۰	۰	۰	۰	۰	۰	۰

بعد از اجرای حلقه ۲ برای i = ۰:

	۰	۱	۲	۳	۴	۵	۶
C =	۱	۲	-۵	-۳	۰	۰	۰

بعد از اجرای حلقه ۲ برای i=۱ :

	۰	۱	۲	۳	۴	۵	۶
C =	۱	۲	-۳	-۸	-۳	۰	۰

بعد از اجرای حلقه ۲ برای i=۲ :

	۰	۱	۲	۳	۴	۵	۶
C =	۱	۲	۰	-۲	-۱۸	-۱۵	۰

بعد از اجرای حلقه ۲ برای i=۳ :

	۰	۱	۲	۳	۴	۵	۶
C =	۱	۲	۰	-۶	-۲۶	۱۱	۱۲

بنابراین:

$$R(x) = p(x)Q(x) = ۱ + ۲x - ۶x^۳ - ۲۶x^۴ + ۱۱x^۵ + ۱۲x^۶$$

روش ۲. روش تقسیم و غلبه معمولی برای ضرب دو چند جمله ای

دو چند جمله‌ای $P(x)$ و $Q(x)$ را می‌توان به شکل زیر نوشت:

$$P(x) = \left(\underbrace{a_۰ + a_۱x + a_۲x^۲ + \ldots + a_{\frac{n}{۲}-۱}x^{\frac{n}{۲}-۱}}_{P_L(x)} \right) + \left(\underbrace{a_{\frac{n}{۲}} + a_{\frac{n}{۲}+۱}x + \ldots + a_{n-۱}x^{\frac{n}{۲}-۱}}_{P_r(x)} \right) x^{\frac{n}{۲}}$$

$$= P_L(x) + P_r(x)x^{\frac{n}{۲}}$$

$$Q(x)=\left(\underbrace{b_0+b_1x+\ldots+b_{\frac{n}{2}-1}x^{\frac{n}{2}-1}}_{Q_L(x)}\right)+\left(\underbrace{b_{\frac{n}{2}}+b_{\frac{n}{2}+1}x+\ldots+b_{n-1}x^{\frac{n}{2}-1}}_{Q_r(x)}\right)x^{\frac{n}{2}}$$

$$=Q_L(x)+Q_r(x)x^{\frac{n}{2}}$$

البته برای سهولت کار فرض کرده ایم $n=2^m$.

در نتیجه ضرب $R(x)=P(x)Q(x)$ را می‌توان نوشت:

$$R(x)=\left(P_L(x)+P_r(x)x^{\frac{n}{2}}\right)\left(Q_L(x)+Q_r(x)x^{\frac{n}{2}}\right)$$

$$\boxed{=P_L(x)Q_L(x)+\left[P_L(x)Q_r(x)+P_r(x)Q_L(x)\right]x^{\frac{n}{2}}+P_r(x)Q_r(x)x^n}$$

فرمول ۱

یعنی ضرب دو چند جمله‌ای درجه $n-1$ به چهار ضرب چند جمله‌ای از درجات $\frac{n}{2}-1$ تبدیل شده است و هر کدام از این چند جمله‌ای‌های درجه $\frac{n}{2}-1$ را نیز می‌توان در صورت لزوم به چهار عمل ضرب چند جمله‌ای‌های درجات $\frac{n}{4}-1$ تبدیل کرد. می‌توان این روند را تا رسیدن به چند جمله‌ای‌های درجه ۱ یا صفر ادامه داد. (دقت کنید ضرب در x^n یا $x^{\frac{n}{2}}$ عمل ضرب نمی‌خواهد فقط کافی است داده‌ها را جابجا کنیم)

اگر مرتبه زمانی ضرب دو چند جمله‌ای درجه $n-1$، $P(x)$ و $Q(x)$ را با $T(n)$ نشان دهیم می‌توان نوشت:

$$T(n)=4T\left(\frac{n}{2}\right)+f(n)$$

که $f(n)$ مرتبه سه عمل جمع چند جمله‌ای است که در فرمول ۱ مشاهده می‌شود و می‌دانیم جمع چند جمله‌ای‌ها از مرتبه $\theta(n)$ می‌باشد پس:

$$T(n)=4T\left(\frac{n}{2}\right)+cn$$

و با استفاده از قضیه Master آنگاه $T(n)=\theta(n^2)$.

یعنی این روش مزیتی نسبت به روش غیر تقسیم و غلبه ندارد.

مثال: روش گفته شده را با یک مثال نشان می‌دهیم.

پاسخ: فرض می‌کنیم:

$$P(x) = ۱ + x + ۳x^۲ - ۴x^۳ = \underbrace{۱ + x}_{P_L} + \underbrace{(۳ - ۴x)}_{P_r} x^۲ = P_L(x) + P_r(x)\, x^۲$$

$$Q(x) = ۱ + ۲x - ۵x^۲ - ۳x^۳ = \underbrace{۱ + ۲x}_{Q_L} + \underbrace{(-۵ - ۳x)}_{Q_r} x^۲ = Q_L(x) + Q_r(x)\, x^۲$$

بنابراین

$$R(x) = P(x)\, Q(x) = P_L\, Q_L + [P_L\, Q_r + Q_L\, P_r] x^۲ + P_r\, Q_r\, x^۴$$

$$P_L\, Q_L = ۱ + ۳x + ۲x^۲$$

$$P_L\, Q_r = -۵ - ۸x - ۳x^۲$$

$$P_r\, Q_L = ۳ + ۲x - ۸x^۲$$

بنابراین

$$P_L\, Q_r + P_r\, Q_L = -۲ - ۶x - ۱۱x^۲$$

$$P_r Q_r = (۳ - ۴x)(-۵ - ۳x) = -۱۵ + ۱۱x + ۱۲x^۲$$

در نهایت:

$$R(x) = ۱ + ۳x + ۲x^۲ + (-۲ - ۶x - ۱۱x^۲) x^۲ + (-۱۵ + ۱۱x + ۱۲x^۲) x^۴$$

$$= ۱ + ۳x - ۶x^۳ - ۲۶x^۴ + ۱۱x^۵ + ۱۲x^۶$$

روش ۳. روش تقسیم و غلبه سریع برای ضرب دو چند جمله‌ای

روش ۲ به چهار ضرب چند جمله‌ای از درجات $\frac{n}{۲} - ۱$ نیاز داشت. ولی می‌توان با ۳ عمل ضرب چند جمله‌ای درجات $\frac{n}{۲} - ۱$ نیز عمل ضرب دو چند جمله‌ای درجه $n-۱$ را انجام داد:

فرض کنید

$$R_L = P_L\, Q_L$$

$$R_r = P_r\, Q_r$$

$$R_m = (P_L + P_r)(Q_L + Q_r) = P_L\, Q_L + P_L\, Q_r + P_r\, Q_L + P_r\, Q_r$$

$$= R_L + (P_L\, Q_r + P_r\, Q_L) + R_r$$

بنابراین:

$$P_L\, Q_r + P_r\, Q_L = R_m - R_L - R_r$$

بنابراین با جایگذاری روابط فوق، در فرمول ۱ بدست می‌آید:

فرمول ۲

$$R = R_L + (R_m - R_L - R_r)x^{\frac{n}{2}} + R_r x^n$$

همانگونه که مشاهده می‌شود برای محاسبه $R(x)$، ۳ عمل ضرب چند جمله‌ای لازم است ولی در عوض به جای ۳ عمل جمع ۶ عمل جمع چند جمله‌ای انجام می‌شود (عمل ضرب برای R_m و R_L و R_r و عمل جمع ۲ تا برای R_m و ۴ تا عمل در R). می‌توان رابطه بازگشتی زیر را برای زمان اجرا نوشت:

$$T(n) = 3T\left(\frac{n}{2}\right) + cn$$

و با قضیه Master داریم $T(n) = \theta\left(n^{\log_2^3}\right)$ یعنی $T(n) = \theta\left(n^{1/58}\right)$

مثال: عملیات روش ۳ را نشان می‌دهیم.

$$P(x) = 1 + x + (3 - 4x)x^2 = P_L + P_r x^2$$

$$Q(x) = 1 + 2x + (-5 - 3x)x^2 = Q_L + Q_r x^2$$

$$R_L = (1 + x)(1 + 2x) = 1 + 3x + 2x^2$$

$$R_r = (3 - 4x)(-5 - 3x) = -15 + 11x + 12x^2$$

$$R_m = (4 - 3x)(-4 - x) = -16 + 8x + 3x^2$$

بنابراین

$$R_m - R_L - R_r = -2 - 6x - 11x^2$$

بنابراین با توجه به فرمول ۲ داریم:

$$R = (1 + 3x + 2x^2) + (-2 - 6x - 11x^2)x^2 + (-15 + 11x + 12x^2)x^4$$

$$= 1 + 3x - 6x^2 - 26x^4 + 11x^5 + 12x^6$$

۴- ضرب اعداد صحیح بزرگ - ویرایش ۱

برای ضرب دو عدد صحیح بزرگ می‌توان ارقام را به عنوان ضریب یک چند جمله‌ای در نظر گرفت و با روش ضرب چند جمله‌ای‌ها، عمل ضرب را انجام می‌دهیم.

مثال: فرض کنید ما فقط ضرب اعداد یک رقمی را می‌توانیم انجام دهیم و همچنین فرض کنید ضرب در اعداد 10^x با عمل شیفت انجام می‌شود. برای ضرب دو عدد ۱۲۳۴×۵۶۱۸ حداقل تعداد عمل ضرب چند تاست؟

پاسخ: با توجه به روش تقسیم و غلبه سریع برای ضرب دو چند جمله‌ای (فرمول ۲) داریم.

$$P = 1234 = 12 \times 10^2 + 34$$

$$Q = 5618 = 56 \times 10^2 + 18$$

$$R_r = P_r.Q_r = 12 \times 56$$

$$R_L = P_L.Q_L = ۳۴\times۱۸$$

$$R_m = (P_r + P_L)(Q_r + Q_L) = (۱۲+۳۴)(۵۶+۱۸) = ۴۶\times۷۴$$

$$R = R_L + (R_m - R_L - R_r)x^{\frac{n}{۲}} + R_r x^n$$

$$= ۳۴\times۱۸ + (۴۶\times۷۴ - ۱۲\times۵۶ - ۳۴\times۱۸)\times۱۰^{۲} + ۱۲\times۵۶\times۱۰^{۴}$$

حال اعداد ۲ رقمی ظاهر شده در عبارت بالا را با همین روش در هم ضرب می‌کنیم:

$$۴۶\times۷۴ = (۴\times۱۰^{۱}+۶)(۷\times۱۰^{۱}+۴) = ۴\times۷\times۱۰^{۲} + (۱۰\times۱۱ - (۴\times۷) - (۶\times۴))\times۱۰^{۱} + (۶\times۴)$$

$$۱۲\times۵۶ = (۱\times۱۰^{۱}+۲)(۵\times۱۰^{۱}+۶) = ۱\times۵\times۱۰^{۲} + (۳\times۱۱ - (۱\times۵) - (۲\times۶))\times۱۰^{۱} + (۲\times۶)$$

$$۳۴\times۱۸ = (۳\times۱۰^{۱}+۴)(۱\times۱۰^{۱}+۸) = ۱\times۳\times۱۰^{۲} + ((۷\times۹) - (۳\times۱) - (۴\times۸))\times۱۰^{۱} + (۴\times۸)$$

بعضی ضرب‌ها مثل ۱۰×۱۱ خود نیاز به شکستن دارند:

$$۱۰\times۱۱ = (۱\times۱۰^{۱}+۰)(۱\times۱۰^{۱}+۱) = ۱\times۱\times۱۰^{۲} + ((۱\times۲) - (۱\times۱) - (۰\times۱))\times۱۰^{۱} + (۰\times۱)$$

$$۲\times۱۱ = (۰\times۱۰^{۱}+۳)(۱\times۱۰^{۱}+۱) = ۰\times۱\times۱۰^{۲} + ((۳\times۲) - (۰\times۱) - (۳\times۱))\times۱۰^{۱} + (۱\times۳)$$

تعداد ضرب‌های تک رقمی ۱۸ تاست که چند تای آنها تکراری هستند و ۱۲ تا ضرب تک رقمی متمایز وجود دارد. نتیجه به صورت زیر است:

$$۴۶\times۷۴ = ۳۴۰۴$$

$$۱۲\times۴۶ = ۶۷۲$$

$$۳۴\times۱۸ = ۶۱۲$$

$$۱۲۳۴\times۵۶۱۸ = ۶۷۲\times۱۰^{۴} + ۲۱۲\times۱۰^{۳} + ۶۱۲ = ۶۹۳۲۶۱۲$$

۴- ضرب اعداد صحیح بزرگ – ویرایش ۲

می توان یک عدد صحیح بزرگ را با یک آرایه نشان داد به عنوان مثال عدد ۵ ۴ ۳ ۱ ۲ ۷ را می‌توان با آرایه

۵	۴	۳	۱	۲	۷
S[۶]	S[۵]	S[۴]	S[۳]	S[۴]	S[۱]

می توان خانه سمت چپ آرایه را برای علامت در نظر گرفت (۰ مثبت، ۱ منفی)

با الگوریتم‌های ارائه شده برای ضرب دو چند جمله ای، می‌توان دو عدد صحیح بزرگ را در هم ضرب کرد. عدد n رقمی u را می‌توان به صورت زیر نوشت:

$$u = \underset{\text{رقم } n \text{ رقم}}{x} \times ۱۰^{m} + \underset{\text{رقم } \left\lfloor \frac{n}{۲} \right\rfloor}{y} \quad , \quad m = \left\lfloor \frac{n}{۲} \right\rfloor$$

به عنوان مثال:

$$۵۶۷۸۳۲ = \overbrace{۵۶۷}^{\text{۳ رقم}} \times ۱۰^{۳} + \overbrace{۸۳۲}^{\text{۳ رقم}}$$

$$۹۴۲۳۷۲۳ = ۹۴۲۳ \times ۱۰^{۳} + ۷۲۳$$

$\left\lfloor \frac{۷}{۲} \right\rfloor$ رقم $\quad$ $\left\lceil \frac{۷}{۲} \right\rceil$ رقم

بنابراین اگر u و v اعداد n رقمی باشند:

$$u = x \times ۱۰^{m} + y$$

$$v = w \times ۱۰^{m} + z$$

حال ضرب u×v را می‌توان نوشت:

$$u \times v = (x \times ۱۰^{m} + y)(w \times ۱۰^{m} + z) = xw \times ۱۰^{۲m} + (xz + wy) \times ۱۰^{m} + yz$$

یعنی می‌توان با انجام چهار ضرب $\frac{n}{۲}$ رقمی (xw و xz و wy و yz) دو عدد n رقمی uv را در هم ضرب کنیم.

توجه: عملیات جمع و تفریق و div و mod اعداد بزرگ (آرایه) را می‌توان با مرتبه خطی انجام داد.

اگر فرض کنیم اعداد u و v رقم صفر ندارند آنگاه بدترین حالت می‌باشد اگر w(n) را برابر پیچیدگی زمانی در بدترین حالت فرض کنیم آنگاه:

$$w(n) = \underbrace{۴w\left(\frac{n}{۲}\right)}_{\text{۴ ضرب } \frac{n}{۲} \text{ رقمی}} + \underbrace{cn}_{\text{هزینه جمع و ...}}$$

با قضیه Master:

$$w(n) \in \theta\left(n^{\log_{۲}^{۴}}\right) = \theta(n^{۲})$$

همانطور که ملاحظه می‌شود ضرب اعداد صحیح بزرگ از مرتبه درجه ۲ می‌باشد می‌توان تعداد عملیات ضرب را کاهش داد برای این منظور همانطور که دیدیم ما نیاز به xw و xz+wy و yz داریم می‌توان نوشت:

$$r = (x + y)(w + z) = xw + (xz + yw) + yz$$

در این صورت:

$$xz + yw = r - xw - yz$$

یعنی می‌توان ۳ عامل xw و xz+xw و yz را با تعیین مقادیر زیر بدست آورد:

$$r = (x + y)(w + z)\ ,\ xw, yz$$

یعنی فقط با ۳ عمل ضرب به جای ۴ عمل ضرب و می‌توان مرتبه را به صورت زیر نوشت:

$$w(n) = ۳w\left(\frac{n}{۲}\right) + cn \Rightarrow w(n) \in \theta\left(n^{\log_{۲}^{۳}}\right) = \theta(n^{۱/۵۸})$$

۵- ضرب دو ماتریس

می خواهیم دو ماتریس $A=[a_{ij}]$ و $B=[b_{ij}]$ با ابعاد $n\times n$ را در هم ضرب کنیم و حاصل را در ماتریس $C=[c_{ij}]_{n\times n}$ قرار دهیم.

می‌دانیم روش معمولی ضرب به صورت زیر است:

```
for i:=۱ to n do
for j:=۱ to n do
begin
    C[i,j]:=۰;
    for k:=۱ to n do
      C[i,j]:= C[i , j]+A[i , k]*B[k , j];
end;
```

بدیهی است مرتبه این الگوریتم $\theta(n^۳)$ می‌باشد. در این الگوریتم، تعداد ضرب و جمع $n^۳$ می‌باشد ولی اگر بجای $C[i , j]:=۰$ بنویسیم $C[i , j]:=A[i , ۱]*B[۱ , j]$; و حلقه سوم را از $k:=۲$ شروع کنیم آنگاه تعداد جمع $n^۳-n^۲$ می‌شود.

می توان یک روش تقسیم و غلبه معمولی برای ضرب دو ماتریس ارائه داد. برای سهولت فرض کنید n توانی از ۲ می‌باشد می‌توان دو ماتریس A و B را به صورت زیر نوشت:

$$A=\begin{bmatrix}A_{۱۱} & A_{۱۲}\\ A_{۲۱} & A_{۲۲}\end{bmatrix}_{n\times n},\quad B=\begin{bmatrix}B_{۱۱} & B_{۱۲}\\ B_{۲۱} & B_{۲۲}\end{bmatrix}_{n\times n}$$

$$C=A\times B=\begin{bmatrix}C_{۱۱} & C_{۱۲}\\ C_{۲۱} & C_{۲۲}\end{bmatrix}_{n\times n}$$

یعنی هر ماتریس را می‌توان به ۴ ماتریس $\frac{n}{۲}\times\frac{n}{۲}$ تقسیم کرد. داریم:

$$C_{۱۱}=A_{۱۱}B_{۱۱}+A_{۱۲}B_{۲۱}$$
$$C_{۱۲}=A_{۱۱}B_{۱۲}+A_{۱۲}B_{۲۲}$$
$$C_{۲۱}=A_{۲۱}B_{۱۱}+A_{۲۲}B_{۲۱}$$
$$C_{۲۲}=A_{۲۱}B_{۱۲}+A_{۲۲}B_{۲۲}$$

همانطور که ملاحظه می‌شود ضرب دو ماتریس $n\times n$ به ۸ ضرب $\frac{n}{۲}\times\frac{n}{۲}$ تقسیم شده است. این عمل تقسیم آنقدر انجام می‌شود تا به وضعیتی برسیم که نیاز به تقسیم نباشد (مثلاً $n=۱$).

لازم به ذکر است جمع ماتریس‌ها از مرتبه $n^۲$ می‌باشد پس می‌توان رابطه بازگشتی زیر را برای زمان اجرا نوشت:

$$T(n)=۸T\left(\frac{n}{۲}\right)+cn^۲ \quad\Rightarrow\quad T(n)=\theta(n^۳)$$

مشاهده می‌شود که این روش مزیتی نسبت به روش معمولی ضرب ندارد.

روش دیگر ضرب، روش استراسن است. در این روش تعریف می‌کنیم:

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

برای محاسبه P و Q و R و S و T و U و V، هفت ضرب ماتریسی و ۱۰ جمع یا تفریق ماتریس نیاز است. حال می‌توان مقادیر ماتریس C را از روابط زیر بدست آورد:

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

محاسبه C_{11} و C_{12} و C_{21} و C_{22} فقط نیاز به ۸ جمع یا تفریق ماتریسی دارد. بنابراین می‌توان برای تعداد عملیات ضرب رابطه بازگشتی زیر را نوشت

$$\begin{cases} T(n) = 7T\left(\frac{n}{2}\right) \\ T(1) = 1 \end{cases} \Rightarrow T(n) \in \theta\left(n^{\log_2^7}\right) = \theta(n^{2/81})$$

و همچنین برای تعداد عملیات جمع می‌توان نوشت:

$$\begin{cases} T(n) = 7T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2 \\ T(1) = 1 \end{cases}$$

و می‌توان نشان داد:

$$T(n) = 6n^{\log 7} - 6n^2 = 6n^{2/81} - 6n^2 \in \theta(n^{2/81})$$

نکته: *تعداد ضرب و جمع در روش معمولی ضرب به ترتیب n^3 و $n^3 - n^2$ می‌باشد ولی تعداد ضرب و جمع در روش استراسن به ترتیب $n^{2/81}$ و $6n^{2/81} - 6n^2$ می‌باشد.*

نکته: *در روش استراسن فرض کردیم $n = 2^m$ می‌باشد اگر چنین نبود می‌توان تعداد کافی سطر و ستون صفر به ماتریس‌ها اضافه کرد تا ابعاد ماتریس توانی از ۲ شود.*

نکته: کاپراسمیت و وینوگراد (۱۹۸۷) الگوریتم برای ضرب ماتریس‌ها ارائه کردند که پیچیدگی زمانی آن برای تعداد عمل‌های ضرب $O(n^{2/38})$ است. ولی ثابت آن چنان بزرگ است که معمولاً خود روش استراسن کارآیی بیشتری دارد.

نکته: شکل دیگری از روش استراسن توسط ساموئل وینوگراد ارائه شده است که پیچیدگی زمانی جمع و تفریق‌های آن از رابطه $T(n) \simeq 5n^{2/81} - 5n^2$ بدست می‌آید.

نکته: ثابت می‌شود که ضرب ماتریس‌ها نیاز به الگوریتمی دارد که پیچیدگی زمانی آن حداقل درجه دوم است. این که آیا ضرب ماتریس‌ها را می‌توان با الگوریتم درجه ۲ انجام داد هنوز پاسخی به آن داده نشده است.

۶- مرتب‌سازی سریع Quicksort

آرایه A[۱۰۰n] را در نظر بگیرید. می‌خواهیم آرایه را به صورت صعودی مرتب کنیم. یک عنصر (برای سادگی عنصر اول A[۱]) به عنوان محور (Pivot) انتخاب می‌شود سپس با انجام عملیاتی (Partition) آرایه دو قسمت می‌شود، قسمت اول عناصر کمتر مساوی محور و قسمت دوم عناصر بیشتر مساوی محور:

۱	j-۱	j	j+۱	n
کوچکتر مساوی محور		Pivot محور	بزرگتر مساوی محور	

سپس به صورت بازگشتی برای هر قسمت (۱-j ... ۱ و n ... j+۱)، عمل گفته شده انجام می‌شود. الگوریتم Partition را می‌توان به صورت الگوریتم ۷ ارائه داد:

```
Function partition (A , Low , high): integer;
var
   i , j , pivot: integer;
begin
   pivot:= A [Low]; i:=Low; j:=high+1;
   repeat
      repeat i:=i+1; until A[i]>=pivot;
      repeat j:=j-1; until A[j]<=pivot;
      if i<j then swap (A[i] , A[j]);
   until i>=j;
   swap (A[Low] , A[j])
   return j
end;
```

در الگوریتم ۷، i برابر اندیس شروع و j از اندیس پایان یک واحد بیشتر تعیین می‌شود سپس i به جلو حرکت می‌کند تا به عنصری برسیم که از محور بیشتر یا مساوی است و j به عقب حرکت می‌کند تا به عنصری برسیم که از محور کمتر یا مساوی است سپس اگر i<j باشد A[i] و A[j] با هم تعویض می‌شوند و عمل تکرار می‌شود. اگر $i \geq j$ باشد آنگاه عنصر محور با A[j] تعویض می‌شود.

مثال: الگوریتم ۷ را به صورت j := partition(A,۱,۱۰) فراخوانی کنید (با توجه به آرایه داده شده زیر)

A[۱]	A[۲]	A[۳]	A[۴]	A[۵]	A[۶]	A[۷]	A[۸]	A[۹]	A[۱۰]
۲۶	۵	۳۷	۱	۶۱	۱۱	۵۹	۱۵	۴۸	۱۹

i=۱ : مقدار اولیه

j=۱۱

شروع Low=۱ و high=۱۰ و pivot=۲۶ و i=۱ و j=۱۱

i زیاد می‌شود تا به ۳۷ برسد و j کم می‌شود تا به ۱۹ برسد سپس ۳۷ و ۱۹ با هم تعویض می‌شوند:

۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
۲۶	۵	۱۹	۱	۶۱	۱۱	۵۹	۱۵	۴۸	۳۷

i=۳

j=۱۰

مجدداً i زیاد می‌شود تا به عنصری که از ۲۶ بیشتر است برسد (۶۱) و j کم می‌شود تا به ۱۵ برسد و سپس این دو با هم تعویض می‌شوند:

۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
۲۶	۵	۱۹	۱	۱۵	۱۱	۵۹	۶۱	۴۸	۳۷

i=۵

j=۸

مجدداً i زیاد می‌شود تا برابر ۷ شود و j کم می‌شود تا برابر ۶ شود و چون $j \le i$، حلقه‌های repeat پایان می‌یابند و A[j] با محور (A[۱]) تعویض می‌شود:

۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
۱۱	۵	۱۹	۱	۱۵	۲۶	۵۹	۶۱	۴۸	۳۷

j=۶

i=۷

بنابراین تابع partition مقدار ۶ برمی گرداند.

حال می‌توان الگوریتم مرتب سازی سریع را به صورت تقسیم و غلبه و براساس الگوریتم partition نوشت. الگوریتم ۸ مرتب‌سازی سریع را نشان می‌دهد:

```
Procedure quicksort (Var A: Arraylist; Low , high: integer);
Var j:integer;
begin
    if Low < high then
    begin
    j:=partition (A  , Low , high);
    quicksort (A , Low , j-1);
    quicksort (A , j+1 , high);
    end
end;
```

الگوریتم ۸- فراخوانی این الگوریتم برای آرایه A[۱..n] باید به صورت quicksort(A,۱,n) باشد.

نکته: *الگوریتم دیگری برای* partition *ارائه شده است که کمی با الگوریتم گفته شده در ۷ متفاوت است.*

الگوریتم partition نسخه دوم به صورت الگوریتم ۹ می‌باشد:

```
function partition (A , Low , high): integer;
var
  pivot , i , j:integer;
begin
  pivot:=A[Low];
  j:=Low;
  for i:=Low+1 to high do
  if A[i]<pivot then
  begin
     j:=j+1;
     swap (A[i] , A[j]);
  end;
  swap (A[Low] , A[j]);
  return j;
end;
```

مثال: الگوریتم ۹ را روی آرایه A با فراخوانی (۱۰ و ۱ و A) partition اعمال کنید:

	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
A	۲۶	۵	۳۷	۱	۶۱	۱۱	۵۹	۱۵	۴۸	۱۹

j=۱

شروع :

Pivot=26 , j=1 , i=2 to 10

i = ۲ : A[۲] < ۲۶ ⇒ j = ۲, A[۲] ↔ A[۲]

i = ۳ : A[۳] < ۲۶ ×

i = ۴ : A[۴] < ۲۶ ⇒ j = ۳, A[۴] ↔ A[۳] ⇒

۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
۲۶	۵	۱	۳۷	۶۱	۱۱	۵۹	۱۵	۴۸	۱۹

i = ۵ : A[۵] < ۲۶ ×

i = ۶ : A[۶] < ۲۶ ⇒ j = ۴ , A[۶] ↔ A[۴] ⇒

۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
۲۶	۵	۱	۱۱	۶۱	۳۷	۵۹	۱۵	۴۸	۱۹

i = ۷ : A[۷] < ۲۶ ×

i = ۸ : A[۸] < ۲۶ ⇒ j = ۵, A[۸] ↔ A[۵] ⇒

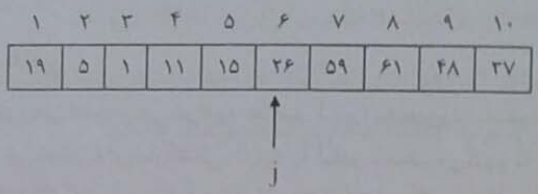
۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
۲۶	۵	۱	۱۱	۱۵	۳۷	۵۹	۶۱	۴۸	۱۹

i = ۹ : A[۹] < ۲۶ ×

i = ۱۰ : A[۱۰] < ۲۶ ⇒ j = ۶ , A[۱۰] ↔ A[۶] ⇒

۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
۲۶	۵	۱	۱۱	۱۵	۱۹	۵۹	۶۱	۴۸	۳۷

پس از پایان حلقه for محور با A[j] تعویض می‌شود:



و مقدار j یعنی ۶ برگشت داده می‌شود.

ملاحظه می‌شود که نتیجه الگوریتم ۷ و ۹ یکسان نیست ولی هر دو، آرایه را به دو قسمت افراز می‌کنند، قسمت اول کمتر از محور و قسمت دوم عناصر بزرگتر از محور.

نکته: *پیچیدگی زمانی الگوریتم* partition $T(n) = n - 1 = \theta(n)$ *می‌باشد.*

نکته: *اگر با عمل پارتیشن، L عنصر قبل از محور قرار گیرد آنگاه زمان مرتب‌سازی سریع از رابطه* $T(n) = T(L) + T(n - L - 1) + \theta(n)$ *به دست می‌آید.*

نکته: *اگر عنصر اول به عنوان محور انتخاب شود آنگاه بدترین حالت وقتی پیش می‌آید که آرایه از قبل مرتب باشد.*

در این حالت در هر فراخوانی partition، یک سمت محور هیچ عنصری نیست و می‌توان برای $w(n)$ رابطه بازگشتی زیر را نوشت:

$$w(n) = \underbrace{w(0)}_{} + \underbrace{w(n-1)}_{} + \underbrace{n-1}_{}$$

یعنی $W(n)$ (زمان در بدترین حالت) برابر است با:

$$\begin{cases} w(n) = w(n-1) + n - 1 \\ w(0) = 0 \end{cases}$$

که می‌توان نتیجه گرفت $w(n) = \frac{n(n-1)}{2} \in \theta(n^2)$

لازم به ذکر است که اگر عنصر اول محور نباشد، بدترین حالت وقتی که آرایه مرتب است پیش نمی‌آید ولی باز هم بدترین حالت دارای پیچیدگی زمانی $\theta(n^2)$ خواهد بود.

نکته: *اگر* $A(n)$ *پیچیدگی زمانی مرتب‌سازی سریع در حالت میانگین باشد می‌توان ثابت کرد:*

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

که با حل این رابطه بازگشتی داریم:

$$A(n) = 2(n+1)\text{Ln}n = 2(n+1)\text{Ln}2\ \text{Log}_2^n \simeq 1/38(n+1)\text{Log}_2^n \in \theta(n\,\text{Log}_2^n)$$

نکته: *بهترین حالت زمانی اتفاق می‌افتد که عنصر افراز (محور) همواره وسط آرایه قرار گیرد. در این حالت اگر برای سهولت فرض کنیم* $n = 2^m$ *آنگاه برای پیچیدگی زمانی در بهترین حالت* $B(n)$ *می‌توان نوشت:*

$$B(n) = 2\ B\left(\frac{n}{2}\right) + cn \quad \Rightarrow \quad B(n) \in \theta(n\,\text{Log}_2^n)$$

نکته: *نسخه دیگری از مرتب‌سازی سریع موسوم به* randomized *وجود دارد که هر بار محور به صورت تصادفی انتخاب می‌شود.*

۷- مرتب سازی ادغامی

آرایه A[۱۰۰n] مفروض است و می‌خواهیم عناصر آن را به صورت صعودی مرتب کنیم.

ابتدا با استفاده از فراخوانی‌های بازگشتی، آرایه را آنقدر نصف می‌کنیم تا به آرایه‌های تک عنصری برسیم و سپس آرایه‌های تک عنصری را با هم ادغام (merge) می‌کنیم تا لیست‌های ۲ عنصری مرتب پدید آید. و عمل merge را به همین ترتیب روی لیست‌های ۲ عنصری و ... انجام می‌دهیم:

```
void mergesort (Low , high)
  {
    If (Low< high)
      {
          mid = ⌊(Low + high)/۲⌋;
          mergesort (Low, mid);
          mergesort (mid+1, high);
          merge (Low, mid, high);
      }
  }
```

این الگوریتم باید به صورت (n و ۱) mergesort فراخوانی شود.

الگوریتم merge زیر آرایه‌های مرتب A[low .. mid] و A[mid+1 .. high] را دریافت می‌کند و خروجی مرتب A[Low .. high] را تولید می‌کند:

```
void merge (int Low, int mid , int high)
  {
    int h=Low , i=Low , j=mid+1 , B[1 .. n];
    While (h<=mid && j<=high)
     {
       if (A[h]<=A[j])
         {
           B[i]=A[h]; h+ +;
         }
       else {
           B[i]=A[j]; j++;
         }
       i+ +;
     }
 if (h>mid) //                    اگر نیمه اول تمام شده، بقیه نیمه دوم را در B کپی کن
    for (k=j; k<=high; k++ , i++)
       B [i]=A[k]
 else //                          اگر نیمه دوم تمام شده، بقیه نیمه اول را در B کپی کن
    for (k=h; k<=mid; k++ , i++)
       B[i]=A[k];
 for (k=low; k<=high; k++) //     آرایه B را در A کپی کن
       A[k]=B[k];
 }
```

نکته: *حداکثر تعداد مقایسه‌ها برای ادغام دو لیست n و m عنصری $n+m-1$ و حداقل مقایسه‌ها $\min\{m,n\}$ می‌باشد. بنابراین ادغام دو لیست n عنصری از مرتبه $\theta(n)$ می‌باشد.*

نکته: *اگر $T(n)$ زمان اجرای* mergesort *باشد آنگاه با توجه به اینکه آرایه دو قسمت مساوی می‌شود (n را توانی از ۲ فرض می‌کنیم) و با مرتبه $\theta(n)$ ادغام می‌شوند می‌توان نوشت:*

$$T(n) = ۲T\left(\frac{n}{۲}\right) + \theta(n)$$

که با حل این رابطه $T(n) = \theta(n\,Log n)$ بدست می‌آید.

نکته: *در بدترین حالت زمان اجرای مرتب‌سازی ادغامی برای آرایه n عنصری:*

$$T(n) = ۲T(\frac{n}{۲}) + n - ۱ \;,\; T(۱) = ۰$$

است که حل دقیق آن $T(n) = n\,Log\,n - n + ۱$ است.

نکته: *می‌توان ثابت کرد، پیچیدگی زمانی میانگین تعداد مقایسه‌ها برای مرتب‌سازی ادغامی برابر است با:*

$$A(n) = n\,Log n - ۲n \sum_{i=۱}^{\log n} \frac{۱}{۲^i + ۲} = n\,Log n - ۱/۲۶n = \theta(n\,Log n)$$

مثال: مراحل فراخوانی اجرای (۱۰ و ۱) mergesort را بر روی آرایه زیر نشان دهید:

	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
A	۴۰	۳۰	۱۰	۹۰	۵۰	۶۰	۱۰۰	۲۰	۷۰	۸۰

پاسخ: اولین فراخوانی آرایه را به دو زیر آرایه $A[۱..۵]$ و $A[۶..۱۰]$ تقسیم می‌کند. سپس $A[۱..۵]$ به دو زیر آرایه $A[۱..۳]$ و $A[۴..۵]$ تقسیم می‌شود. و سپس $A[۱..۳]$ به $A[۱..۲]$ و $A[۳..۳]$ افراز می‌شود. و سرانجام $A[۱..۲]$ به $A[۱..۱]$ و $A[۲..۲]$ تقسیم می‌شود:

۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
۴۰	۳۰	۱۰	۹۰	۵۰	۶۰	۱۰۰	۲۰	۷۰	۸۰

خطوط عمودی تقسیم را نشان می‌دهند. سپس $A[۱]$ و $A[۲]$ با هم ادغام می‌شوند:

(۳۰ ۴۰ | ۱۰ | ۹۰ ۵۰ | ۶۰ ۱۰۰ ۲۰ ۷۰ ۸۰)

حال $A[۱..۲]$ و $A[۳]$ با هم ادغام می‌شوند:

(۱۰ ۳۰ ۴۰ | ۹۰ ۵۰ | ۶۰ ۱۰۰ ۲۰ ۷۰ ۸۰)

سپس A[۴..۵] به A[۴] و A[۵] شکسته می‌شود و ادغام می‌شود:

(۱۰ ۳۰ ۴۰ ۵۰ ۹۰ | ۶۰ ۱۰۰ ۲۰ ۷۰ ۸۰)

در این لحظه، الگوریتم به اولین فراخوانی mergesort بازگشته و در حال پردازش دومین فراخوانی بازگشتی است. فراخوانی‌های بازگشتی متوالی، منجر به تقسیم زیر آرایه دوم به صورت زیر خواهد شد:

(۱۰ ۳۰ ۴۰ ۵۰ ۹۰ | ۶۰ | ۱۰۰ | ۲۰ | ۷۰ ۸۰)

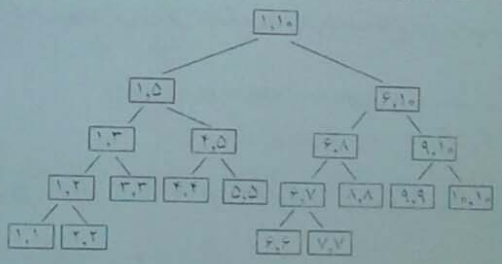
در نهایت با ادغام آنها دو زیر آرایه مرتب حاصل می‌شود

(۱۰ ۳۰ ۴۰ ۵۰ ۹۰ | ۲۰ ۶۰ ۷۰ ۸۰ ۱۰۰)

و ادغام این دو زیر آرایه مرتب، آرایه مرتب نهایی را می‌دهد:

۱۰ ۲۰ ۳۰ ۴۰ ۵۰ ۶۰ ۷۰ ۸۰ ۹۰ ۱۰۰

درخت بازگشتی که دنباله فراخوانی‌های mergesort را نشان می‌دهد به شکل زیر است:



زوج مقادیر داخل هر گره نشان‌دهنده مقادیر Low و high هستند.

نکته: *فضای کمکی مرتب‌سازی ادغامی* $\theta(n)$ *می‌باشد و غیر درجاست.*

توجه: در الگوریتم merge وقتی یک لیست تمام شود باید لیست دیگر را کپی کرد که این نیاز به بررسی دارد. برای جلوگیری از بررسی این وضعیت می‌توان انتهای هر یک از دولیست مقداری که از همه مقادیر بزرگتر است قرار داد، این مقدار را sentinel گویند و با ∞ نشان می‌دهیم. واضح است که هر دو لیست باید به این مقدار برسند:

```
MERGE (A,p,q,r) // A[p..q] , A[q+1..r]
n1 ← q − p + 1 , n2 ← r − q
Create arrays L[1..n1 + 1] and R[1..n2 + 1]
for i ← 1 to n1 do L[i] ← A[p + i − 1]
for j ← 1 to n2 do R[j] ← A[q + j]
L[n1 + 1] ← ∞ , R[n2 + 1] ← ∞
i ← 1 , j ← 1
for k ← p to r
    do if L[i] ≤ R[j] then A[k] ← L[i] , i ← i + 1
                      else A[k] ← R[j] , j ← j + 1
```

۸- مساله انتخاب - یافتن بزرگترین کلید دوم و کوچکترین کلید iام

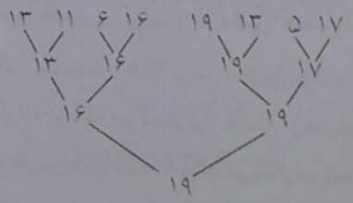
مساله انتخاب، پیدا کردن i امین بزرگترین کلید یا i امین کوچکترین کلید در آرایه‌ای نامرتب n عنصری می‌باشد.

یک راه ساده آن است که ابتدا آرایه را مرتب کرده سپس i امین عنصر را استخراج کنیم که مرتبه آن $\theta(n\lg n)$ خواهد بود. ولی در ادامه روش‌های سریعتری از مرتبه $\theta(n)$ معرفی می‌کنیم. حالت خاص i = ۱ همان ماکزیمم یا مینیمم یابی است که حل شده است. حالت خاص i = ۲ را بررسی می‌کنیم.

یافتن بزرگترین کلید دوم

یک روش ساده آن است که با الگوریتم استاندارد یافتن بزرگترین کلید در زمان $T_1(n) = n - 1$ عنصر ماکزیمم را یافته آن را کنار بگذاریم، سپس در زمان $T_2(n) = n - 2$ بزرگترین عنصر بعدی را در n-۱ داده باقیمانده بیابیم که در کل $T(n) = 2n - 3$ می‌شود. ولی با تکنیک تورنمنت می‌توان این زمان را کاهش داد.

روش تورنمنت همان روشی است که در مسابقات ورزشی حذفی استفاده می‌شود. فرض کنید ۸ عدد به صورت درخت زیر با یکدیگر مسابقه دهند و هر بار برنده عدد بزرگتر باشد:



برای حالت n=۸، ۳ دور مسابقه و در کل ۷ مسابقه نیاز است یعنی در حالت کلی $\left\lceil \mathrm{Log}_2^n \right\rceil$ دور مسابقه و n-۱ مسابقه نیاز است. برای یافتن بزرگترین کلید دوم، باید از بین کلیدهایی که به بزرگترین کلید باخته‌اند، بزرگترین را بیابیم. یعنی از بین [۱۶ و ۱۷ و ۱۳] باید ماکزیمم را بیابیم که این عمل $\left\lceil \mathrm{Log}_2^n \right\rceil - 1$ مقایسه نیاز دارد.

پس تعداد کل مقایسه‌ها برابر است با:

$$T(n) = (n-1) + \left\lceil \mathrm{Log}_2^n \right\rceil - 1 = n + \left\lceil \mathrm{Log}_2^n \right\rceil - 2$$

در این رابطه، n-۱ تعداد مقایسه‌های یافتن بزرگترین می‌باشد.

نکته: *هر الگوریتمی که بتواند بزرگترین (کوچک‌ترین) کلید دوم را برای هر ورودی دلخواه تنها با مقایسه کلیدها پیدا کند، در بدترین حالت، حداقل مقایسه‌های آن برابر است با:*

$$T(n) = n + \left\lceil \mathrm{Log}_2^n \right\rceil - 2$$

یافتن کوچکترین کلید iام

برای یافتن کوچکترین i امین عنصر، می‌توان از تابع partition که در مرتب‌سازی سریع دیدیم استفاده کرد. به این صورت که خروجی partition را q می‌نامیم حال اگر i ، A[q] امین عنصر باشد خودش جواب است ولی اگر چنین نباشد یا سمت چپ محور یا سمت راست آن (نه هر دو سمت) باید partition تکرار شود:

```
SELECT (A,p,r,i) // A[p..r]
if p=r then return A[p]  // یک عنصر وجود دارد
q ← partition (A,p,r)  // q نقطه محوری است
k ← q-p+1  // k تعداد عناصر قبل با خود محور
if i=k then return A[q]
elseif i<k then SELECT (A,p, q-1,i)
else return SELECT (A, q+1, r, i-k)
```

دقت کنید اگر نقطه محوری q باشد و تعداد عناصر آرایه از ابتدا تا محل افراز (A[p..q]) برابر i باشد $(i = k = q-p+1)$ پس A[q] جواب است (i امین کوچکترین عنصر) ولی اگر $i < k$ یعنی باید سمت چپ نقطه محوری، تابع SELECT اجرا شود و اگر $i > k$ یعنی باید سمت راست نقطه محوری تابع SELECT اجرا شود و $i-k$ امین عنصر را بیابد و نه i امین عنصر را.

مرتبه تابع SELECT در حالت متوسط $\theta(n)$ و در بدترین حالت $\theta(n^2)$ است.

ولی می‌توان تابع سریعتری ارائه داد که در بدترین حالت از مرتبه $\theta(n)$ بتواند i امین کوچکترین عنصر را بیابد. این الگوریتم که نامش را SELECTION می‌گذاریم شامل ۵ مرحله است:

۱- آرایه ورودی را به $\left\lceil \frac{n}{5} \right\rceil$ گروه تقسیم کنید ($\left\lfloor \frac{n}{5} \right\rfloor$ گروه ۵ عنصری است و حداکثر یک گروه یعنی آخرین گروه، کمتر از ۵ عنصری است). مرتبه این عمل $\theta(n)$ است.

۲- میانه هر گروه را بیابید (اگر تعداد عناصر فرد باشد، میانه عنصر وسط در لیست مرتب شده و اگر تعداد عناصر زوج باشد، ۲ تا میانه وجود دارد که میانه سمت چپ منظور است). چون هر گروه ۵ عنصری است برای یافتن میانه هر گروه، هر گروه را مرتب کرده و سومین عنصر میانه است. مرتبه این مرحله $O(n)$ است.

۳- به طور بازگشتی میانه $\left\lceil \frac{n}{5} \right\rceil$ میانه‌ای که در مرحله ۲ یافته‌اید را بیابید و اسم آن را x می‌گذاریم (میانه میانه‌ها). زمان این مرحله $T\left(\left\lceil \frac{n}{5} \right\rceil\right)$ است.

۴- x را محور قرار دهید و آرایه را Partition کنید. فرض کنید قبل از x ، $k-1$ عنصر قرار گرفته‌اند یعنی x برابر k امین کوچکترین عنصر است و $n-k$ عنصر سمت راست محور هستند. زمان این مرحله $O(n)$ است.

۵- اگر $i = k$ یعنی x جواب است. در غیر این صورت اگر $i < k$ الگوریتم SELECTION را روی نیمه سمت چپ x انجام دهید. و اگر $i > k$ آنگاه $i-k$ امین کوچکترین را سمت راست x بیابید. زمان این مرحله $T\left(\frac{۷n}{۱۰}+۶\right)$ است.

توضیح: حداقل نصف میانه‌های یافته شده در مرحله ۲ از x بزرگتر هستند. بنابراین $\frac{۱}{۲}\left\lceil\frac{n}{۵}\right\rceil$ میانه از x بزرگتر هستند یعنی $\frac{۱}{۲}\left\lceil\frac{n}{۵}\right\rceil$ گروه وجود دارند که میانه‌شان از x بزرگتر است و در هر گروه ۲ عنصر از میانه آن گروه بزرگتر است پس در هر گروه ۳ عنصر وجود دارد که از x بزرگتر است البته بجز گروه آخر (که ممکن است کمتر از ۵ عنصر داشته باشد) و خود گروهی که x عضو آن است. پس تعداد عناصری که از x بزرگتر هستند حداقل برابر $۳\left(\left(\frac{۱}{۲}\left\lceil\frac{n}{۵}\right\rceil\right)-۲\right)\geq\frac{۳n}{۱۰}-۶$ است به طور مشابه، تعداد عناصری که از x کوچکتر هستند حداقل $\frac{۳n}{۱۰}-۶$ است. بنابراین در بدترین حالت در مرحله ۵، الگوریتم روی $n-\left(\frac{۳n}{۱۰}-۶\right)=\frac{۷n}{۱۰}+۶$ عنصر فراخوانی می‌شود. می‌توان رابطه بازگشتی زیر را برای زمان اجرای SELECTION نوشت:

$$T(n)\leq\begin{cases}\theta(۱) & \text{if} \quad n\leq ۱۴۰\\ T\left(\left\lceil\frac{n}{۵}\right\rceil\right)+T\left(\frac{۷n}{۱۰}+۶\right)+\theta(n) & \text{if} \quad n>۱۴۰\end{cases}$$

در این رابطه بجای عدد ۱۴۰ می‌توان هر عدد بیشتر یا مساوی ۷۰ گذاشت (اثبات نمی‌کنیم) با حل این رابطه بازگشتی $T(n)\in O(n)$ است.

مثال: در الگوریتم SELECTION ، آرایه ورودی به گروه‌های ۵ تایی تقسیم شدند. آیا اگر به گروه‌های ۷ عنصری تقسیم کنیم، زمان همچنان خطی است؟ به گروه‌های ۳ عنصری چطور؟

پاسخ: فرض کنید گروه‌ها k عنصری باشند. تعداد اعداد بزرگتر (یا کوچکتر) از میانه میانه‌ها (x) حداقل $\left\lceil\frac{k}{۲}\right\rceil\left(\left(\frac{۱}{۲}\left\lceil\frac{n}{k}\right\rceil\right)-۲\right)\geq\frac{n}{۴}-k$ است. پس در بدترین حالت SELECTION به طور بازگشتی روی حداکثر $n-\left(\frac{n}{۴}-k\right)=\frac{۳n}{۴}+k$ عنصر اجرا می‌شود:

$$T(n)\leq T\left(\left\lceil\frac{n}{k}\right\rceil\right)+T\left(\frac{۳n}{۴}+k\right)+O(n)$$