فصل سوم

رویکرد تقسیم و حل

(Divide And Conquer)

روش تقسیم و حل

- یک مسئله به دو یا چند مسئله کوچکتر تقسیم می‌شود (عمل تقسیم تا زمانی که مسائل کوچکتر حاصل قابل حل باشند، ادامه می‌یابد)
- مسائل کوچکتر حل می‌شوند.
- با ترکیب نتایج حل مسائل کوچکتر، جواب مسئله اصلی حاصل می‌شود.
- یک روش Top-Down است (روش به کار گرفته شده توسط ناپلئون)
- الگوریتم‌ها در این روش بیشتر به صورت بازگشتی پیاده سازی می‌شوند.

الگوریتم‌های نمونه روش تقسیم و حل

- جستجوی دودویی
- مرتب‌سازی ادغامی
- مرتب‌سازی سریع
- ضرب ماتریس‌ها
- ضرب اعداد صحیح بزرگ
- یافتن بزرگترین و کوچکترین کلیدها
- مسئله انتخاب
- و . . .

جستجوی دودویی

اگر x (مقدار مورد جستجو) با عنصر وسط آرایه برابر است، خارج شو، در غیر این صورت:

۱. تقسیم

آرایه به دو زیر آرایه با اندازه‌ای تقریبا برابر نصف اندازه آرایه اولیه. اگر x کوچکتر از عنصر وسط می‌باشد، آرایه سمت چپ را انتخاب کن. اگر x بزرگتر از عنصر وسط می‌باشد، آرایه سمت راست را انتخاب کن.

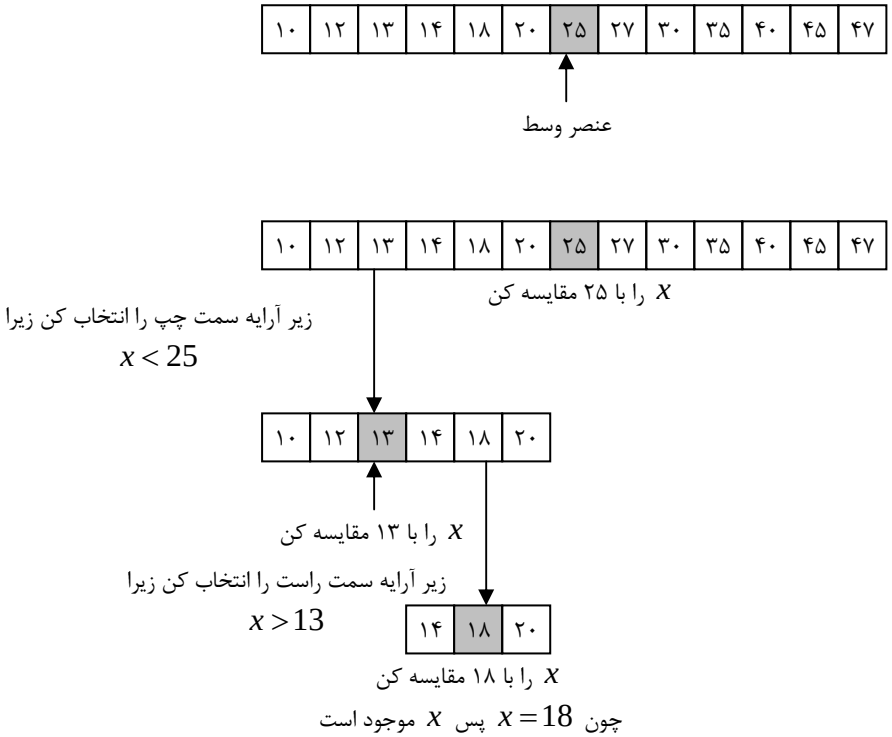
۲. حل

زیر آرایه به صورت بازگشتی با تعیین این که آیا x در آن زیر آرایه قرار دارد یا خیر، مگر این که اندازه زیر آرایه به اندازه کافی کوچک باشد.

۳. راه حل

راه حل آرایه را با توجه به راه حل زیر آرایه تعیین کن.

مثال: فرض کنید $x=18$ و آرایه به صورت زیر باشد:



توسعه یک الگوریتم بازگشتی

توسعه، روشی برای بدست آوردن راه حل یک نمونه از روی راه حل یک یا چند نمونه کوچکتر (تعمیم)

تعیین شرط (شرایط) نهایی نزدیک شدن به نمونه (های) کوچکتر

تعیین راه حل در شرط (شرایط) نهایی

مثال:

الگوریتم ۲-۱ جستجوی دودویی (بازگشتی)

مساله: تعیین کنید که آیا x (مقدار مورد جستجو) در آرایه مرتب شده S به اندازه n وجود دارد یا خیر.

ورودی‌ها: عدد صحیح مثبت (آرگومان) n (پارامتر)، آرایه مرتب S که از ۱ تا n اندیس گذاری شده است، مقدار x.

خروجی‌ها: location، موقعیت x در S (یعنی محل x در S) (اگر x در S نباشد برابر صفر می‌باشد)

location: نام الگوریتم

low: کران پایئن (اندیس اول آرایه)

high: کران بالا (اندیس آخر آرایه)

```
index location (index low, high)
{
        index mid;
        if (low > high)
                return 0;
        else
                {
                mid = [(low+high)]/2;
                if (x == S[mid])
                        return mid;
                else if (x < S[mid])
                        return location (low, mid - 1);
                else
                        return location (mid + 1, high);
                }
}
```

بررسی خط به خط:

برای مثال آرایه S را به صورت زیر در نظر میگیریم:

آرایه :S	20	25	30	35	40
اندیس	۱	۲	۳	۴	۵

x = 25 , L = 1 , H = 5

L	H	Location (low, High)	mid	آرایهS
1	5	Location (1, 5)	3	20 25 30 25 40 (۱ ۲ ۳ ۴ ۵) — L mid H
1	2	Location (1, 2)	1	20 25 (۱ ۲) — L mid H
2	2	Location (2, 2)	2	25 (۲) — L H mid

مقدار ۲ به عنوان اندیس خانه‌ای که حاوی x (مقدار ۲۵) می‌باشد، به محل فراخوانی الگوریتم برگردانده می‌شود.

تمرین: این الگوریتم را برای مقدار $x = 60$ Trace کنید (نکته، آرایه ۵ خانه‌ای باشد، با همان مقادیر مثال حل شده در کلاس)

پیچیدگی زمانی (بدترین حالت)

عمل اصلی: مقایسه x با $S[mid]$

اندازی ورودی: تعداد عناسر آرایه، n

پیچیدگی زمانی:

هزینه آرایه n خانه ای برابر است با هزینه یک آرایه $\frac{n}{2}$ خانه‌ای بعلاوه ۱ (که ۱ هزینه خانه وسط است):

$$W(n) = W\left(\frac{n}{2}\right) + 1 \qquad \text{for} \qquad n > 1,\ \text{n a power of 2}$$

هزینه آرایه‌ی تک خانه‌ای

$$W(1) = 1$$

که به ۲ معادلات بالا، معادله بازگشتی هزینه گفته می‌شود.

حل:

$W(n)=\log_2 n+1 \Rightarrow W(n)=\theta(\log_2 n), T(n)=\theta(\log_2 n)$

و اگر n به توانی از دو محدود نباشد:

$W(n)=[\log_2 n]+1 \in \theta(\log_2 n)$

یافتن کوچکترین و بزرگترین عنصر آرایه

تقسیم

آرایه به دو زیر آرایه به اندازه $\frac{n}{2}$ عنصر.

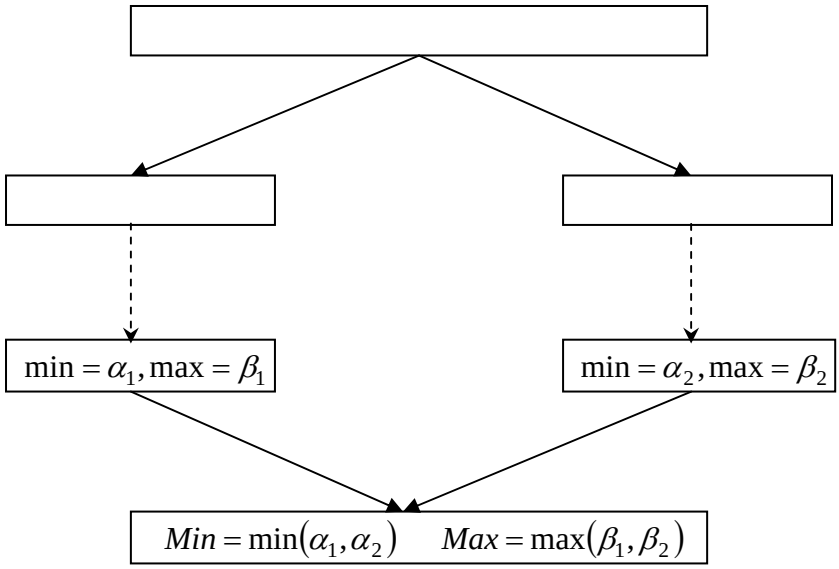
حل

هر یک از زیر آرایه‌ها با یافتن کوچکترین و بزرگترین عنصر در آن.

اگر زیر آرایه به اندازه کافی کوچک نبود ($n>2$)، برای حل آن به روش بازگشتی عمل میکنیم.

ترکیب

راه حل‌های زیر آرایه‌ها با یافتن کوچکترین عنصر در بین کوچکترین عنصر دو زیر آرایه و یافتن بزرگترین عنصر در بین بزرگترین عنصر دو زیر آرایه.



پیچیدگی زمانی: همه حالات

عمل اصلی: مقایسه

اندازه ورودی: تعداد عناصر آرایه n

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

$T(n)$: هزینه پیدا کردن max و min در یک آرایه $\frac{n}{2}$ عنصری

تذکر:

علت استفاده از $2T\left(\frac{n}{2}\right)$ به جای $\frac{n}{2}$ این است که : ۲ بخش با $\frac{n}{2}$ عنصر وجود دارد.

علت استفاده از $2T\left(\frac{n}{2}\right)+2$ به جای $2T\left(\frac{n}{2}\right)+1$ این است که: ۲ مقایسه وجود دارد، یکی بین ۲ مینیمم (min) و یکی بین ۲ ماکسیمم (max) داریم.

پیچیدگی زمانی:

$$T(n) = 2T\left(\frac{n}{2}\right)+2 \overset{n=\frac{n}{2}}{=\!=\!=\!=\!=} 2\left[2T\left(\frac{\frac{n}{2}}{2}\right)\right]=2\left[2T\left(\frac{n}{2^2}\right)+2\right]+2$$

$$=2^2T\left(\frac{n}{2^2}\right)+2^2+2=2\left[2T\left(\frac{\left(\frac{n}{2^2}\right)}{2}\right)+2\right]+2^2+2=2\left[2T\left(\frac{n}{2^3}\right)+2\right]+2^2+2$$

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

$$=\ldots$$

$$=2^iT\left(\frac{n}{2^i}\right)+\sum_{k=1}^{i}2^k$$

تغییر متغیر: $\frac{n}{2^i}=2\Rightarrow n=2^{i+1}\Rightarrow\frac{n}{2}=2^i$

$$T(n)=\left(\frac{n}{2}\right)T(2)+2^{i+1}-2=\frac{n}{2}+n-2=3\frac{n}{2}-2$$

و اگر n به توانی از دو محدود نباشد:

$$T(n)=\left[3\frac{n}{2}\right]-2\in\theta(n)$$

مرتب سازی ادغامی (*MergeSort*)

تقسیم

آرایه به دو زیر آرایه به اندازه $\frac{n}{2}$ عنصر.

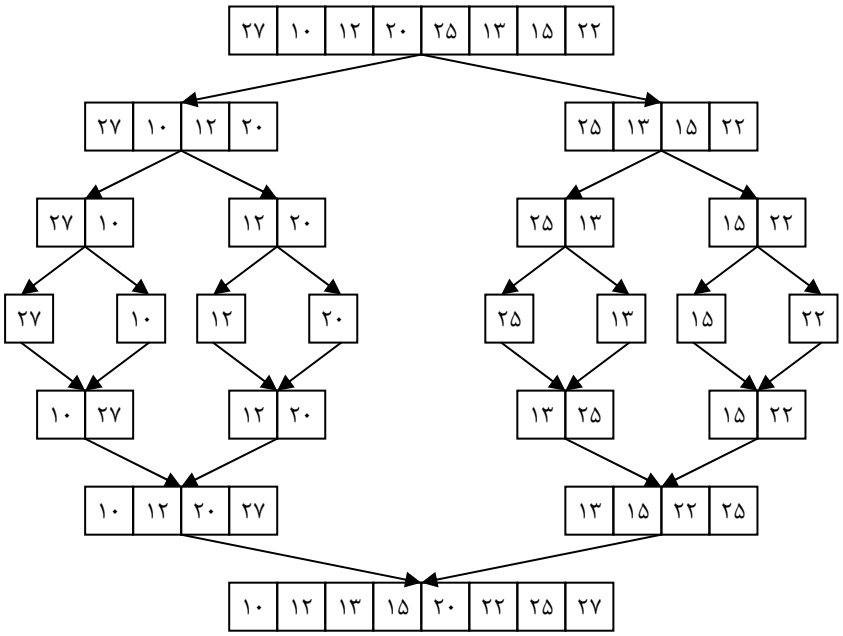
حل

هر یک از زیر آرایه‌ها با مرتب سازی آن. اگر زیر آرایه به اندازه کافی کوچک نبود، برای حل آن به روش بازگشتی عمل می‌کنیم.

ترکیب

راه حل‌های زیر آرایه‌ها با ادغام آن‌ها در یک آرایه مرتب.

مثال: در خصوص مرتب سازی ادغامی



الگورتم مرتب سازی ادغامی

Merge Sort	
Problem: Sort n keys in non-decreasing order.	مشکل مرتب سازی n مقدار در یک آرایه نامرتب
Input: positive integer n, array of keys S indexed from 1 to n.	ورودی عدد صحیح و مثبت n، آرایه‌ای از مقادیر با اندیس‌های ۱ تا n
Output: the array S containing the keys in non-decreasing order.	خروجی آرایه S که شامل مقادیر به صورت مرتب شده صعودی

Void mergesort (int n, keytype S[])	void یعنی تابعی که مقداری بر نمیگرداند mergesort نام الگوریتم است. n تعداد عناصر آرایه (طول آرایه) n و S پارامتر هستند S آرایه است.
{	
const int h = $\left\lceil \frac{n}{2} \right\rceil, m = n - h$;	
keytype $U[1...h], V[1...m]$;	
if $(n > 1)$	
{	
copy S[1] through S[h] to U[1] through U[h];	
copy S[h + 1] through S[n] to V[1] through V[m];	
mergesort (h, U);	
mergesort (m, V);	
merge (h, m, U, V, S);	
}	
}	

الگوریتم ادغام

ادغام	Merge
مشکل ادغام ۲ آرایه مرتب در یک آرایه مرتب	Problem: merge two sorted array into one sorted array.
ورودی اعداد صحیح و مثبت h و m آرایه مرتب U با اندایس‌های ۱ الی h آرایه مرتب V با اندیس‌ها ۱ تا m	Inputs: positive integer h and m, array of sorted keys U indexed from 1 to h, array of sorted keys V indexed from 1 to m.
خروجی S شامل مقادیر مرتب شده به صورت صعودی	Outputs: the array S containing the keys in nondecreasing order.

h: طول آرایه U i: متغیر برای کنترل آرایه U	Void merge (int h, int m, const keytype U[], keytype V[], keytype S[])

		{
		index i,j,k;
		i=1; j=1; k=1;
این حلقه عمل چیدمان در S را تا خالی شدن یکی از دو آرایه U یا V ادامه میدهد	در هر حالت مقدار حلقه در هر بار اجراء یک واحد اضافه میشود	While (i <= h && j <= m)
		{
	در صورتی که عنصر U کوچکتر از عنصر V باشد	if (U[i] < V[j])
		{
		S[K] = U[i]
		i++;
		}
		else
		{
		S[K] = V[j]
		j++;
		}
		k++;
		} //end of while
		if (i > h)
	copy V[j] in صورتیکه آرایه U خالی شده باشد	copy V[j] throughtV[m] to S[k] throught S[h + m];
		else
	copy U[i] در صورتی که آرایه V خالی شده باشد	copy U[i] throughtU[h] to S[k] throught S[h + m];
		}

پیچیدگی ادغام: بدترین حالت

عمل اصلی: مقایسه U[i] با V[j]

اندازه ورودی: h و m، تعداد عناصر موجود در هر یک از دو آرایه ورودی

پیچیدگی زمانی:

$$W(h,n)=h+m-1$$

علت یک واحد کم کردن:

عنصر آخر در یکی از ۲ آرایه، بدون مقایسه به S اضافه میشود.

یعنی زمانی که هنگام خروج از حلقه به دلیل مقایسه تمام عناصر یکی از آرایه‌ها، آرایه دیگر فقط یک عنصر مقایسه نشده داشته باشد.

پیچیدگی زمانی مرتب سازی ادغامی: بدترین حالت

عمل اصلی: مقایسه‌ای که در Merge انجام می‌شود.

اندازه ورودی: n تعداد عناصر آرایه S.

پیچیدگی زمانی:

$$W(n)=W(h)+W(m)+h+m-1$$

اگر n توانی از ۲ باشد:

$$\begin{cases} W(n)=2W\left(\frac{n}{2}\right)+n-1 & \text{for} \quad n>1,\ n \ \text{a power of 2} \\ W(1)=0 \end{cases}$$

$$W(n)=T(n)\Rightarrow T(n)=T(h)+T(m)+h+m-1$$

با فرض : $\frac{n}{2}\approx h=m$ (n به توانی از ۲ محدود نباشد)

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

پیچیدگی حافظه الگوریتم مرتب سازی ادغامی

$$n\left(1+\frac{1}{2}+\frac{1}{4}+\ldots\right)=2n$$

یعنی در مجموع به اندازه ۲ برابر آرایه S (آرایه اولیه)، فضا (حافظه) مورد نیاز است.

برای کاهش پیچیدگی حافظه به n

برای این منظور الگوریتم mergesort2 به جای الگوریتم mergesort مورد استفاده قرار میگیرد که تقسیم آرایه به ۲ بخش را به صورت درجا (in place) انجام می‌دهد.

نکته:

الگوریتم قبلی (mergesort) این کار را به صورت برون از جا (Out Place) انجام می‌داد.

```
* * * * * * * * * * فقط جهت مطالعه * * * * * * * * * *
void mergesort2 (index low, index high)
{
        index mid;
        if (low < high)
                {
                mid = [(low + high)/2];
                mergesort2 (low, mid);
                mergesort2 (mid + 1, high);
                merge2 (low, mid, high);
                }
}
* * * * * * * * * * * * * * * * * * * *
```

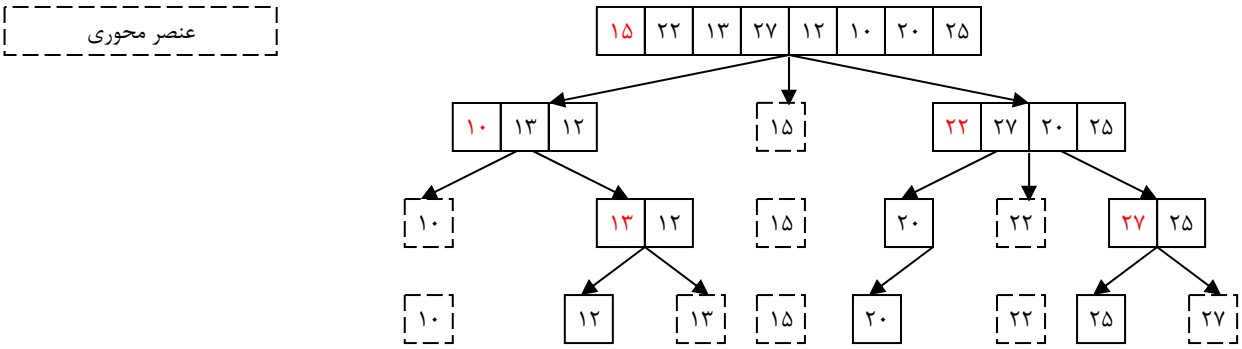
مرتب سازی سریع

توسعه یافته توسط Hoare (1962)

مراحل:

- انتخاب عنصر محوری (معمولا عنصر اول)
- تقسیم آرایه به دو بخش به طوری که عناصر کوچکتر از عنصر محوری در سمت چپ و عناصر بزرگتر از آن در سمت راست آن قرار بگیرند.

مثال:



الگوریتم مرتب سازی سریع (*QucikSort*)

Algorithm: QuickSort

void: یعنی هیچ مقداری برنمیگرداند

quicksort : نام الگوریتم

low: کران پایین آرایه

high: کران بالا آرایه

pivotpoint: متغیری است حاوی اندیس عنصر محوری

```
void quicksort (index low, index high)
{
index pivotpoint
if (high > low)
{
        partition (low, high, pivotpoint);
        quicksort (low, pivotpoint - 1);
```

خود فراخوانی برای بخش اول آرایه.

```
        quicksort (pivotpoint + 1, high);
```

خود فراخوانی برای بخش دوم آرایه

```
}
}
```

الگوریتم بخش بندی (*Partition*)

این زیر الگوریتم، آرایه را بر اساس عنصر محوری به دو بخش تقسیم میکند.

void: الگوریتم هیچ مقداری بر نمیگرداند

partition: نام الگوریتم

low: آرگومانی که دریافت میکند، کران پایین آرایه

high: آرگومانی که دریافت میکند، کران بالای آرایه

pivotpoint: قرار است، مقدار این پارامتر توسط این الگوریتم (Partition) مشخص و به الگوریتم QucikSort گزارش شود. که حاوی اندیس عنصر محوری است.

pivotitem: متغیری است، حاوی خود عنصر محوری

void partition (index low, index high, index& pivotpoint)	
{	
index i, j;	
keytype pivotitem;	
pivotitem = S[low]; //Choose First item for	اولین مقدار در آرایه به عنوان عنصر محوری انتخاب شده و در متغیر pivotitem قرار میگیرد.
j = low; //pivotitem	

for (i = low + 1; i <= high; i++)
{
if (S[i] < pivotitem)
{
j++;
exchange S[i] and S[j];
}
}
	این حلقه، مقدار عنصر محوری را با مقادیر عنصرهای دوم الی آخر آرایه، مقایسه میکند و در صورتیکه مقدار عنصر محوری، بزرگتر از مقدار عنصر دیگری باشد، یک واحد به متغیر j اضافه نموده و جابه‌جایی عناصر را بر مبنای آن انجام می‌دهد. (تعداد عناصر کوچکتر از عنصر محوری مشخص می‌شود)
	در پایان این حلقه، مقدار j، همان اندیس عنصر محوری خواهد بود. (اندیس محل جدید عنصر محوری)

pivotpoint = j;	اندیس جدید عنصر محوری که در j قرار دارد، در متغیر pivotpoint ریخته می‌شود.
exchange S[low] and S[pivotpoint]; //Put pivotitem at pivotpoint	
عنصر محوری که در خانه اول آرایه قرار دارد، به مکان با اندیس j منتقل می‌شود. همچنین مقدار موجود در اندیس j به محل عنصر محوری منتقل خواهد شد.	
}	

مثال در خصوص الگوریتم بخش بندی (Partition)

i	j	S[1]	S[2]	S[3]	S[4]	S[5]	S[6]	S[7]	S[8]	
–	–	15	22	13	27	12	10	20	25	← Initial Values
2	1	15	22	13	27	12	10	20	25	
3	2	15	22	13	27	12	10	20	25	
4	2	15	13	22	27	12	10	20	25	
5	3	15	13	22	27	12	10	20	25	
6	4	15	13	12	27	22	10	20	25	
7	4	15	13	12	10	22	27	20	25	
8	4	15	13	12	10	22	27	20	25	
-	4	10	13	12	15	22	27	20	25	← Final Values
		بخش اول					بخش دوم			

تمرین ۳: مثال ارائه شده در اسلاید ۲۵، یک مرحله از اجرای الگوریتم Partition را برای بخش‌بندی آرایه S نشان می‌دهد. ضمن بررسی مثال این اسلاید، اجرای الگوریتم QuickSort را برای یکی از بخش‌های بدست آمده، بررسی و ارائه نمایید.

پیچیدگی زمانی الگوریتم *Partition*: همه حالات

عمل اصلی: مقایسه $S[i]$ با *pivotitem*

اندازه ورودی: $n = high - low + 1$ (اندازه آرایه)

پیچیدگی زمانی: از آنجا که هر یک از عناصر (به جز اولی) یک بار مقایسه می‌شوند:

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

پیچیدگی زمانی مرتب‌سازی سریع: بدترین حالت

عمل اصلی: مقایسه $S[i]$ با *pivotitem* در رویه *partition*

اندازه ورودی: n اندازه آرایه S

پیچیدگی زمانی:

$$T(n) = \underbrace{T(0)}_{\substack{\text{Time to sort} \\ \text{left subarray}}} + \underbrace{T(n-1)}_{\substack{\text{Time to sort} \\ \text{right subarray}}} + \underbrace{n-1}_{\substack{\text{Time to} \\ \text{partition}}}$$

$$\Rightarrow \begin{cases} T(n) = T(n-1) + n - 1 & \text{for } n > 0 \\ T(0) = 0 \end{cases}$$

حل:

$$T(n) = n\frac{(n-1)}{2} \in \theta(n^2)$$

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

الگوریتم ضرب ماتریس‌ها

الگوریتم عادی ضرب ماتریس‌ها به روش استاندارد

الگوریتم ساده ضرب ماتریس‌ها به صورت زیر است:

```
for (i = 1, i <= n, i++)
        for (j = 1, j <= n, j++)
                {
                C[i][j] = 0;
                for (k = 1, k <= n, k++)
                        C[i][j] = C[i][j] + A[i][k] * B[k][j];
                }
```

تعداد ضرب‌ها: $n^3 = \theta(n^3)$

تعداد جمع‌ها: $n^3 = \theta(n^3)$

با تغییر ساده زیر می‌توان تعداد جمع‌ها را کاهش داد و به $T(n) = n^3 - n^2$ رساند:

```
for (i = 1, i <= n, i++)
        for (j = 1, j <= n, j++)
                {
                C[i, j] = A[i][k] * B[k][j];
                for (k = 2, k <= n, k++)
                        C[i][j] = C[i][j] + A[i][k] * B[k][j];
                }
```

تعداد ضرب‌ها: $n^3 = \theta(n^3)$

تعداد جمع‌ها: $n^3 - n^2 = \theta(n^3)$

الگوریتم ضرب ماتریس‌ها به روش تقسیم و حل

n توانی از ۲ میباشد.

ماتریس A, B را به چهار ماتریس کوچکتر که هر یک $\frac{n}{2} \times \frac{n}{2}$ هستند تقسیم نمود:

$$\begin{matrix} & \overset{\frac{n}{2}}{\longleftrightarrow} & & \\ \frac{n}{2} \updownarrow & \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} & \times & \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} \end{matrix}$$

C_{ij} های بالا را میتوان به همان روش الگوریتم استاندارد ضرب ماتریس‌ها به دست آورد. یعنی:

$$C_{11} = A_{11}B_{11} + A_{12}B_{21} \quad , \quad C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21} \quad , \quad C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

اگر در الگوریتم بالا، عمل اصلی را تعداد ضرب‌ها در نظر بگیریم، آنگاه تعداد ضرب‌ها برای ماتریس‌های $n \times n$ به ۸ عمل ضرب ماتریس‌های $\frac{n}{2} \times \frac{n}{2}$ نیاز خواهد داشت. از طرف دیگر بدیهی است که ضرب دو ماتریس 1×1 فقط به یک عمل ضرب اسکالر نیاز دارد، لذا برای الگوریتم ساده تقسیم و غلبه ضرب ماتریس‌های داریم:

$$\left.\begin{matrix} T(n)=8T\left(\frac{n}{2}\right) \\ T(1)=1 \end{matrix}\right\} \Rightarrow \theta(n^3)$$

$$T(n)=n^{\log_2 8} \Rightarrow T(n)=n^3=\theta(n^3)$$

ضرب ماتریس‌ها طبق تعریف الگوریتم عادی ضرب ماتریس‌ها به روش استاندارد:

تعداد ضرب‌ها: $T(n)=n^3$

تعداد جمع‌ها: $T(n)=n^3-n^2$

الگوریتم استراسن برای ضرب ماتریس‌ها (۱۹۶۹)

- پیچیدگی بهتر از درجه سوم (جمع و ضرب)

روش استراسن (ضرب ماتریس)

$$\underset{C}{\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}} = \underset{A}{\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}} \times \underset{B}{\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}}$$

$$m_1=(a_{11}+a_{22})(b_{11}+b_{22})$$
$$m_2=(a_{21}+a_{22})b_{11}$$
$$m_3=a_{11}(b_{21}-b_{22})$$
$$m_4=a_{22}(b_{21}-b_{11})$$
$$m_5=(a_{11}+a_{12})b_{22}$$
$$m_6=(a_{21}-a_{11})(b_{11}+b_{22})$$
$$m_7=(a_{12}-a_{22})(b_{21}+b_{22})$$

آنگاه:

$$C=\begin{bmatrix} m_1+m_4-m_5+m_7 & m_3+m_5 \\ m_2+m_4 & m_1+m_3-m_2+m_6 \end{bmatrix}$$

برای حالت $n=2$ به سادگی میتوانید آن را اثبات کنید.

برای ماتریس‌های بزرگ

تقسیم ماتریس‌ها:

$$\begin{matrix} & \leftarrow\ n/2\ \rightarrow & & & & & & \\ \uparrow & & & & & & & \\ n/2 & \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} & = & \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} & \times & \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \\ \downarrow & & & & & & & \end{matrix}$$

محاسبه Mها، مثلا:

$$M_1=(A_{11}+A_{22})(B_{11}+B_{22})$$

مثال:

برای ماتریس‌های زیر:

$$A=\begin{bmatrix}1&2&3&4\\5&6&7&8\\9&1&2&3\\4&5&6&7\end{bmatrix}\quad,\quad B=\begin{bmatrix}8&9&1&2\\3&4&5&6\\7&8&9&1\\2&3&4&5\end{bmatrix}$$

و M_1 به صورت زیر محاسبه می‌شود:

$$M_1=(A_{11}+A_{22})(B_{11}+B_{22})=$$
$$\left(\begin{bmatrix}1&2\\5&6\end{bmatrix}+\begin{bmatrix}2&3\\6&7\end{bmatrix}\right)\times\left(\begin{bmatrix}8&9\\3&4\end{bmatrix}+\begin{bmatrix}9&1\\4&5\end{bmatrix}\right)=\begin{bmatrix}3&5\\11&13\end{bmatrix}\times\begin{bmatrix}17&10\\7&9\end{bmatrix}=\begin{bmatrix}86&75\\278&227\end{bmatrix}$$

- همانطور که مشاهده می‌کنید، روش استراسن (برای ضرب دو ماتریس $n\times n$) به تعداد ۷ عمل ضرب و ۱۸ عمل جمع و تفریق (ماتریس‌های $\frac{n}{2}\times\frac{n}{2}$) نیاز دارد.
- برای $n=1$ (دو ماتریس 1×1) یک عمل ضرب و صفر عمل جمع و تفریق نیاز است.
- لذا تعداد ضرب‌ها:

$$\begin{cases}T(n)=7T\left(\frac{n}{2}\right)\Rightarrow T(n)=\theta\left(n^{\log_2 7}\right)=\theta\left(n^{2.81}\right)\\T(1)=1\end{cases}$$

- تعداد جمع‌ها و تفریق‌ها:

$$\begin{cases}T(n)=7T\left(\frac{n}{2}\right)+18\left(\frac{n}{2}\right)^2\Rightarrow T(n)\in\theta\left(n^{\log_2 7}\right)=\theta\left(n^{2.81}\right)\\T(1)=0\end{cases}$$

- هنگامی که دو ماتریس $\frac{n}{2}\times\frac{n}{2}$ با هم جمع یا تفریق می‌شوند. تمام خانه‌های متناظر آن‌ها با هم جمع یا تفریق خواهند شد، لذا ماتریس حاصل $\left(\frac{n}{2}\right)^2$ خانه دارد.

تمرین ۴: Call by reference و Call by Value تحقیق شود. (فراخوانی با مقدار - فراخوانی با ارجاع)

الگوریتم استراسن

Algorithm Strassen	
Problem Determinate the product of two $n\times n$ matrices where n is a power of 2.	ضرب ۲ ماتریس $n\times n$
inputs: an integer n that is a power of 2, and two $n\times n$ matrices A and B.	یک عدد صحیح n که توانی از ۲ است.
Outputs: the product C of A and B.	

بررسی خط به خط الگوریتم استراسن:

تذکر: هر جایی که از & قبل از یک متغیر استفاده میکنیم. یعنی آن مقدار، یک مقدار خروجی الگوریتم است.

void strassen (int n $n\times n$_matrix A, $n\times n$_matrix B, $n\times n$+matrix &C)	void: الگوریتم مقدار بر نمیگرداند. strassen: نام الگوریتم. ماتریس A و B ورودی‌ها هستند. ماتریس C خروجی است.
{	
if (n <= threshold)	
compute C = A × B using the standard algorithm;	اگر ابعاد ماتریس از یک حد آستانه‌ای کمتر است، آن را به روش عادی ضرب کن. (نه با روش استراسن)
else	در غیر این صورت:
{	
partition A into four submatrices A11, A12, A21, A22;	ماتریس A را به چهار ماتریس فرعی تبدیل کن
partition B into four submatrices B11, B12, B21, B22;	ماتریس B را به چهار ماتریس فرعی تبدیل کن
compute C = A × B using Strassen's method;	ماتریس C را که حاصل ضرب A در B میباشد به روش استراسن محاسبه کن
// example recursive call;	
// strassen (n/2, A11 + A22, B11 + B22, M1)	
}	
}	

مقایسه ضرب ماتریس‌ها (روش عادی و استراسن)

Table 2.3 A comparison of two algorithms that multiply $n\times n$ matrices

	Standard Algorithm	Strassen's Algorithm
تعداد ضرب‌ها: Multiplications	n^3	$n^{2.81}$
تعداد جمع‌ها: Additions/Subtractions	n^3-n^2	$6n^{2.81}-6n^2$

محاسبات با اعداد صحیح بزرگ

نمایش اعداد صحیح بزرگ: جمع و عملیات خطی دیگر

- استفاده از آرایه‌ای از اعداد صحیح، در هر خانه یک رقم
- ذخیره علامت، در بالاترین محل آرایه
- الگوریتم‌های زمان خطی:
  - جمع
  - تفریق
  - $u\times 10^m$
  - $u\ divide\ 10^m$
  - $u\ rem\ 10^m$

تذکر:

rem نشان‌دهنده باقیمانده است.

ضرب اعداد صحیح بزرگ

تقسیم یک عدد صحیح n رقمی به دو عدد صحیح که هر کدام، تقریبا دارای $\frac{n}{2}$ رقم، می‌باشد. مثلا:

$$-567{,}832 = 567 \times 10^3 + 832$$

$$-9{,}423{,}723 = 9423 \times 10^3 + 723$$

به طور کلی

$$\underbrace{u}_{n\ digits} = \underbrace{x}_{\left\lceil \frac{n}{2} \right\rceil digits} \times 10^m + \underbrace{y}_{\left\lfloor \frac{n}{2} \right\rfloor digits}$$

که $m = \left\lfloor \frac{n}{2} \right\rfloor$

تذکر

علامت $\lfloor x \rfloor$ به معنای کف مقدار محاسباتی x است و علامت $\lceil x \rceil$ به معنای سقف مقدار محاسباتی x می‌باشد. به طور مثال :

$\lfloor 3.5 \rfloor = 3$ $\quad\quad$ $\lfloor -3.5 \rfloor = -4$

$\lceil 3.5 \rceil = 4$ $\quad\quad$ $\lceil -3.5 \rceil = -3$

ضرب

برای ضرب نمودن دو عدد صحیح n رقمی

$$u = x \times 10^m + y$$

$$v = w \times 10^m + z$$

حاصل ضرب برابر است با:

$$uv = xw \times 10^{2m} + (xz + wy) \times 10^m + yz$$

مثال:

$$\overbrace{\overbrace{x}^{567}\overbrace{y}^{832}}^{u} \times \overbrace{\overbrace{w}^{9{,}423}\overbrace{z}^{723}}^{v} = \left(\overbrace{\overbrace{x}^{567} \times 10^3 + \overbrace{y}^{832}}^{u}\right) \times \left(\overbrace{\overbrace{w}^{9423} \times 10^3 + \overbrace{z}^{723}}^{v}\right)$$

$$= 567 \times 9423 \times 10^6 + (567 \times 723 + 9423 \times 832) \times 10^3 + 832 \times 723$$

الگوریتم ضرب

ضرب اعداد بزرگ	Algorithm: Large Integer Multiplication
ضرب ۲ عدد بزرگ u و v	Problem: Multiply two large integers, u and v.
حاصلضرب u در v	Inputs: large integers u and v.
نام الگوریتم $prod$ میباشد. پارامترها: u و v.	Outputs: prod, the product of u and v.

	large_integer prod (large_integer u, large_integer v)
	{
تعریف ۴ متغیر	large _integer x, y, w, z;
تعریف n و m	int n, m;
و n را برابر قرار بده با تعداد ارقام عددی از u و v که بیشترین تعداد رقم را دارد.	n = maximum (number of digits in u, number of digits in v)
اگر u یا v برابر با صفر باشد.	if ($u == 0 \| v == 0$)
صفر را برگردان	return 0;
اگر تعداد ارقام بزرگترین عدد، کوچکتر از یک حد آستانه باشد.	else if ($n <= thresold$)
از این روش برای ضرب استفاده نمی‌نماید و آنها را به روش معمول در هم ضرب می‌کند.	return $u \times v$ obtained in the usual way;
	else
	{
مقدار جزء صحیح n تقسیم بر ۲ را در m میریزد.	$m = \left\lfloor \frac{n}{2} \right\rfloor$
x را برابر با تقسیم u بر ۱۰ به توان m قرار میدهد	$x = u$ divide 10^m;
	$y = u$ rem 10^m;
	$w = v$ divide 10^m;
	$z = v$ rem 10^m;
در این بخش این الگوریتم، $prod$ خود را ۴ بار خود فراخوانی می‌کند.	return $prod(x,w) \times 10^{2m} + (prod(x,z) + prod(w,y)) \times 10^m + prod(y,z)$
	}
	}

پیچیدگی زمانی : بدترین حالت

عمل اصلی

دستکاری یک رقم دهدهی در یک عدد صحیح بزرگ هنگام جمع کردن، تفریق کردن، یا انجام اعمال تقسیم بر 10^m، ضرب در 10^m و محاسبه باقیمانده بر 10^m.

اندازه ورودی

n، تعداد ارقام هر یک از دو عدد

پیچیدگی زمانی:

$$\begin{cases} T(n) = 4T\left(\frac{n}{2}\right) + Cn & for \quad n > s, n \quad a \quad power \quad of \quad 2 \\ T(s) = 0 \end{cases}$$

هزینه ضرب ۲ عدد بزرگ n رقمی برابر با ۴ برابر ضرب ۲ عدد $\frac{n}{2}$ رقمی می‌باشد (۴ بار خود فراخوانی انجام می‌شود)

$T(0) = 0$ باشد، یعنی یکی از اعداد ورودی صفر باشد.

حل:

$$W(n) \in \theta\left(n^{\log 4}\right) = \theta\left(n^2\right)$$

تابع $prod$ چهار بار خودش را با اندازه $\frac{n}{2}$ صدا می‌زند و عملیات جانبی مثل جمع و تفریق و rem و $divide$، همگی از مرتبه $O(n)$ است.

کاهش تعداد ضرب‌ها

الگوریتم $prod$ باید موارد زیر را محاسبه کند:

- xw
- $xz + yw$
- yz

الگوریتم $prod$ برای محاسبه موارد مذکور، ۴ مرتبه فراخوانی می‌شود.

اگر

$$r = \underbrace{(x+y)}_{u}\underbrace{(w+z)}_{v} = xw + (xz + yw) + yz$$

آنگاه

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

بنابراین تنها باید سه حاصل ضرب زیر را محاسبه کنیم:

- $\underbrace{(x+y)\times(w+z)}_{r}$
- xw
- yz

الگوریتم جدید ضرب

ضرب اعداد بزرگ	Algorithm: Large Integer Multiplication
ضرب ۲ عدد بزرگ u و v	Problem: Multiply two large integers, u and v.
حاصلضرب u در v	Inputs: large integers u and v.
نام الگوریتم $prod$ میباشد. پارامترها: u و v.	Outputs: prod, the product of u and v.

	large_integer prod (large_integer u, large_integer v)
	{
تعریف ۴ متغیر	large _integer x, y, w, z;
تعریف n و m و n را برابر قرار بده با تعداد ارقام عددی از u و v که بیشترین تعداد رقم را دارد.	int n, m; n = maximum (number of digits in u, number of digits in v)
اگر u یا v برابر با صفر باشد.	if ($u == 0 \| v == 0$)
صفر را برگردان.	return 0;
اگر تعداد ارقام بزرگترین عدد، کوچکتر از یک حد آستانه باشد،	else if ($n <= thresold$)
از این روش برای ضرب استفاده نمی‌نماید و آنها را به روش معمول در هم ضرب می‌کند.	return $u \times v$ obtained in the usual way;
در غیر این صورت	else
	{
مقدار جزء صحیح n تقسیم بر ۲ را در m میریزد.	$m = \left\lceil \frac{n}{2} \right\rceil$
x را از خارج قسمت تقسیم u بر ۱۰ به توان m	$x = u \text{ divide } 10^m$;
y را از باقیمانده تقسیم u بر ۱۰ بدست می‌آوریم.	$y = u \text{ rem } 10^m$;
	$w = v \text{ divide } 10^m$;
	$z = v \text{ rem } 10^m$;
	$r = prod2(x + y, w + z)$;
	$p = prod2(x, w)$
	$q = prod2(y, z)$;
در این بخش این الگوریتم، $prod2$ خود را ۴ بار خود فراخوانی می‌کند. در اینجا p همان معادل xw و q معادل yz می‌باشد.	return $p \times 10^{2m} + (r - p - q) \times 10^m + q$;
	}
	}

پیچیدگی زمانی : بدترین حالت

عمل اصلی:

دستکاری یک رقم دهدهی در یک عدد صحیح بزرگ هنگام جمع کردن، تفریق کردن، یا انجام اعمال تقسیم بر 10^m، ضرب در 10^m و محاسبه باقیمانده بر 10^m

اندازه ورودی:

n، تعداد ارقام هر یک از دو عدد

پیچیدگی زمانی:

$$\begin{cases} T(n) = 3T\left(\frac{n}{2}\right) + Cn & for\ n > s,\ n\ a\ power\ of\ 2 \\ T(s) = 0 \end{cases}$$

حل:

$$T(n) \in \theta\left(n^{\log_2 3}\right) = \theta\left(n^{1.58}\right)$$

هزینه ضرب ۲ عدد n رقمی، برابر است با ۳ برابر هزینه ضرب ۲ عدد $\frac{n}{2}$ رقمی. زیرا $prod2$ ، ۳ بار فراخوانی می‌شود.

یافتن بزرگترین و کوچکترین مقادیر

روش معمول

الگوریتم استاندارد یافتن بزرگترین و کوچکترین عنصر در آرایه S

```
max = s[1];
min = s[1];
```

مقدار خانه اول آرایه، هم در متغیر max، هم در متغیر min قرار می‌گیرد.

```
for (i=2 ; i<n ; i++)
```

برای مقایسه min و max با خانه‌های دوم الی nام آرایه

```
{
        if (s[i] < min)
```

اگر مقدار خانه iام کوچکتر از min باشد

```
                {
                min = s[i];
```

در min قرار می‌گیرد و مقدار قبلی آن حذف می‌شود.

```
                }
        else if (s[i] > max)
```

اگر مقدار خانه iام بزرگتر از max باشد.

```
                {
                max = s[i];
```

آن مقدار در max قرار می‌گیرد

```
                }
}
```

مرتبه اجرایی (تعداد مقایسه‌ها):

- بدترین حالت (بردار به صورت صعودی مرتب است): شرط جلوی if و else if مرتب آزمایش می‌شود.

$W(n)=2(n-1)$ $T(n)=2n-2=\theta(n)$

علت ضریب ۲ این است که حلقه for از ۲ شروع شده است.

- بهترین حالت (بردار به صورت نزولی مرتب است): هچیگاه بخش else if اجراء نمی‌شود.

$B(n)=n-1$ $T(n)=n-1=\theta(n)$

روش تقسیم و حل

الگوریتم

آرایه اولیه به دو برابر آرایه هر کدام با $\frac{n}{2}$ عنصر تقسیم می‌شود (مرتب تکرار می‌شود تا به تک عنصر برسد)

بزرگترین و کوچکترین مقدار در هر دو آرایه فرعی به دست می‌آید.

بین دو عدد بزرگترین، و بین دو عدد کوچکترین، کوچکتر را انتخاب می‌شود.

الگوریتم:

find (low, high, s, max, min)	find نام الگوریتم که ۵ پارامتر دارد. low کران پائین high کران بالا s آرایه است
{	
if (low == high)	یعنی اگر آرایه تک عنصری باشد.
{	
max = min = s[low];	یعنی مقدار min و max و کران پائین آرایه برابر همان تک خانه خواهد بود.
return;	خروج
}	
if (low == high - 1)	اگر آرایه ۲ عنصری است.
{	
if (s[low] < s[high])	s[low] : مقدار اولین خانه
{	
max = s[high];	
min = s[low];	
}	
else	
{	
max = s[low];	
min = s[high];	
}	
return;	خروج از الگوریتم.
}	
اگر آرایه بیش از ۲ خانه پر داشته باشد. از این بخش به بعد اجراء می‌شود.	
mid = (low + high) / 2;	
find (low, mid, s, max1, min1);	
find (mid + 1, high, s, max2, min2);	
max = maximum (max1, max2);	
min = minimum (min1, min2);	
}	

پیچیدگی زمانی الگوریتم

$$T(n)=\begin{cases}0 & if \quad n=۱\\ 1 & if \quad n=۲\\ \underbrace{T\left(\left\lceil \frac{n}{2}\right\rceil\right)}_{a}+\underbrace{T\left(\left\lfloor \frac{n}{2}\right\rfloor\right)}_{b}+\underbrace{2}_{c}\approx 2T\left(\frac{n}{2}\right)+2 & if \quad n\geq ۳\end{cases}$$

در ردیف اول پیچیدگی صفر است زیرا مقایسه ای نداریم.
در ردیف دوم پیچیدگی ۱ است زیرا یک مقایسه انجام می‌دهیم.
در ردیف سوم پیچیدگی برای ۳ عنصر به بالا با فرمول مربوطه محاسبه می‌شود.

a: هزینه $T(n)$ پیدا کردن min و max در یک آرایه $\frac{n}{2}$ عنصری (خود فراخوانی اول)

b: هزینه پیدا کردن min و max در یک آرایه $\frac{n}{2}$ عنصری (خود فراخوانی دوم)

c: ۲ مقایسه، اولی برای انتخاب max از بین ۲ مقدار ماکزیمم و دومی برای انتخاب min از بین ۲ مقدار مینیمم.

با حل معادله فوق:

$$T(n)=\begin{cases}\frac{3n}{2}-2 & n=2k\\ \frac{3n}{2}-\frac{3}{2} & n=2k+1\end{cases}$$

لذا با توجه به نتیجه فوق و همچنین قضیه اصلی:

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

این فصل از اینجا به بعد اختیاری است.