فصل چهارم

طراحي الگوريتم ها

Computer algorithms

تقسم و حل

Fereshteh_mohammadi@ymail.com

- مساله مرتب سازی ادغام
- مساله مرتب سازی سریع
- مساله ضرب ماتریس استراسن

حل مسأله مرتب‌سازی ادغام به روش تقسیم و حل

فرض کنید لیست نامرتبی با n عنصر داریم که می‌خواهیم این لیست را به ترتیب صعودی مرتب کنیم .

Low → 1 … 10 ← High

1	2	3	4	5	6	7	8	9	10
310	285	179	652	351	423	861	254	450	520

الگوریتم merge sort به روش تقسیم و حل

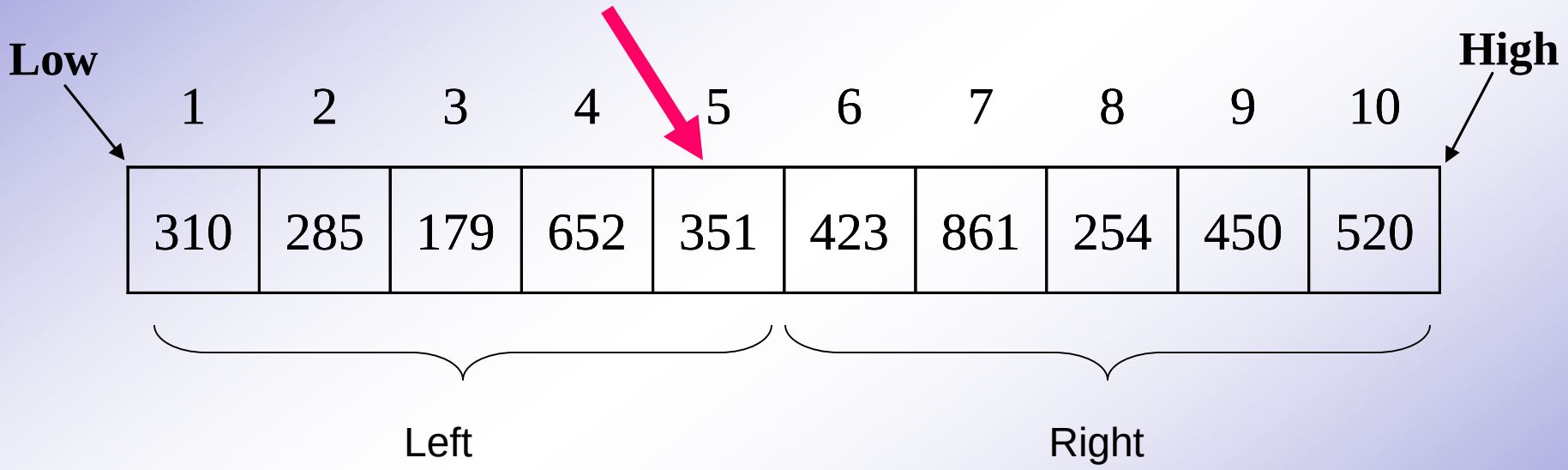
```
Alg mergesort (low , high)
{
  if (low < high)                          // !small(p)
  {
      mid := ⌊(low + high) / 2⌋ ;          // Divide
      mergsort (low , mid) ;               // recursive (left)
      mergesort (mid + 1 , high) ;         // recursive (right)
      merge (low , mid , high) ;           // combine
  }
}
```

در عمل combine هم مرتب سازی و هم ترکیب صورت می گیرد

لیست زیر را با استفاده از الگوریتم merge sort مرتب کنید

310 285 179 652 351 423 861 254 450 520

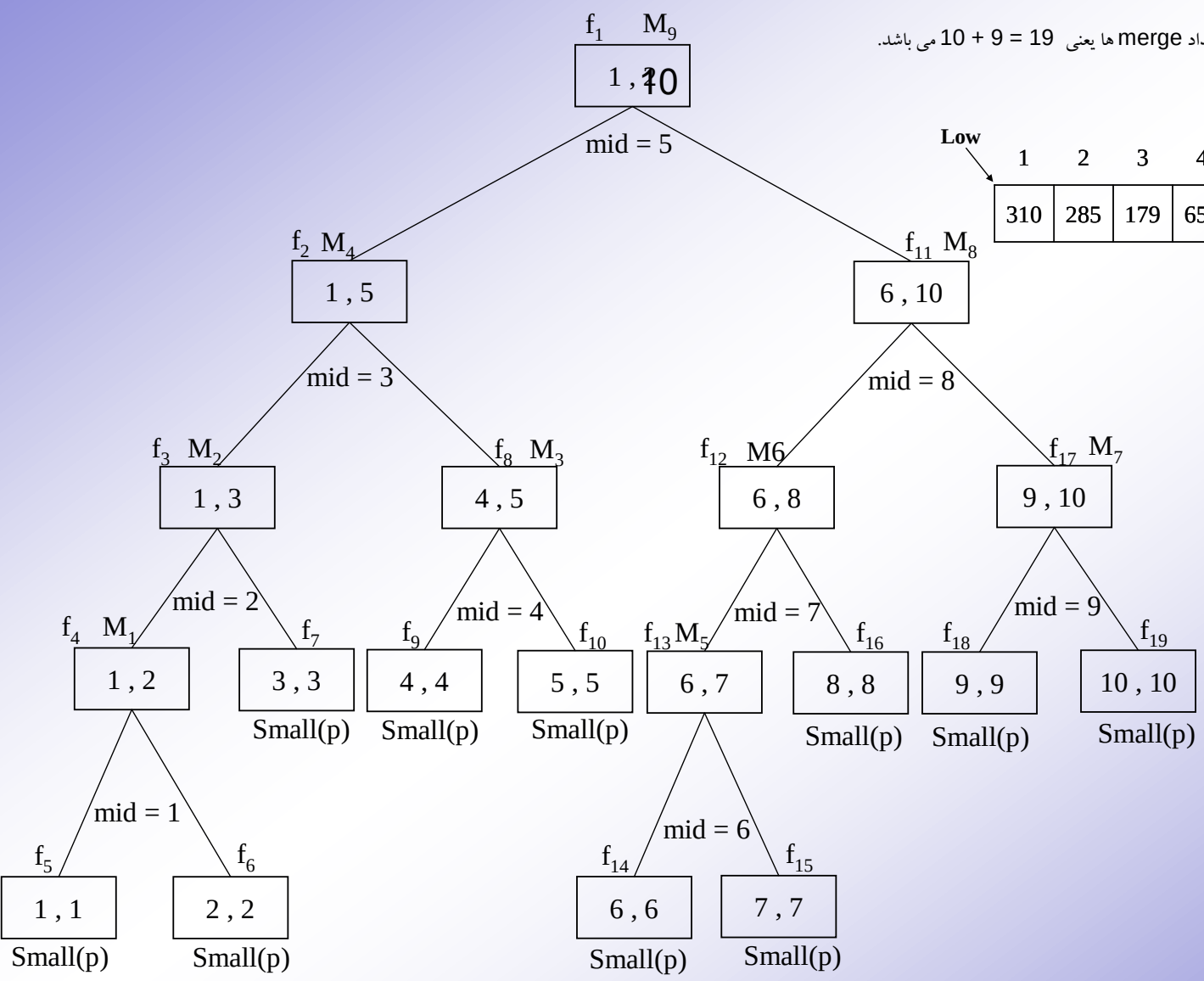
حل: ابتدا لیست اعداد را به صورت زیر در آرایه‌ای قرار می‌دهیم:

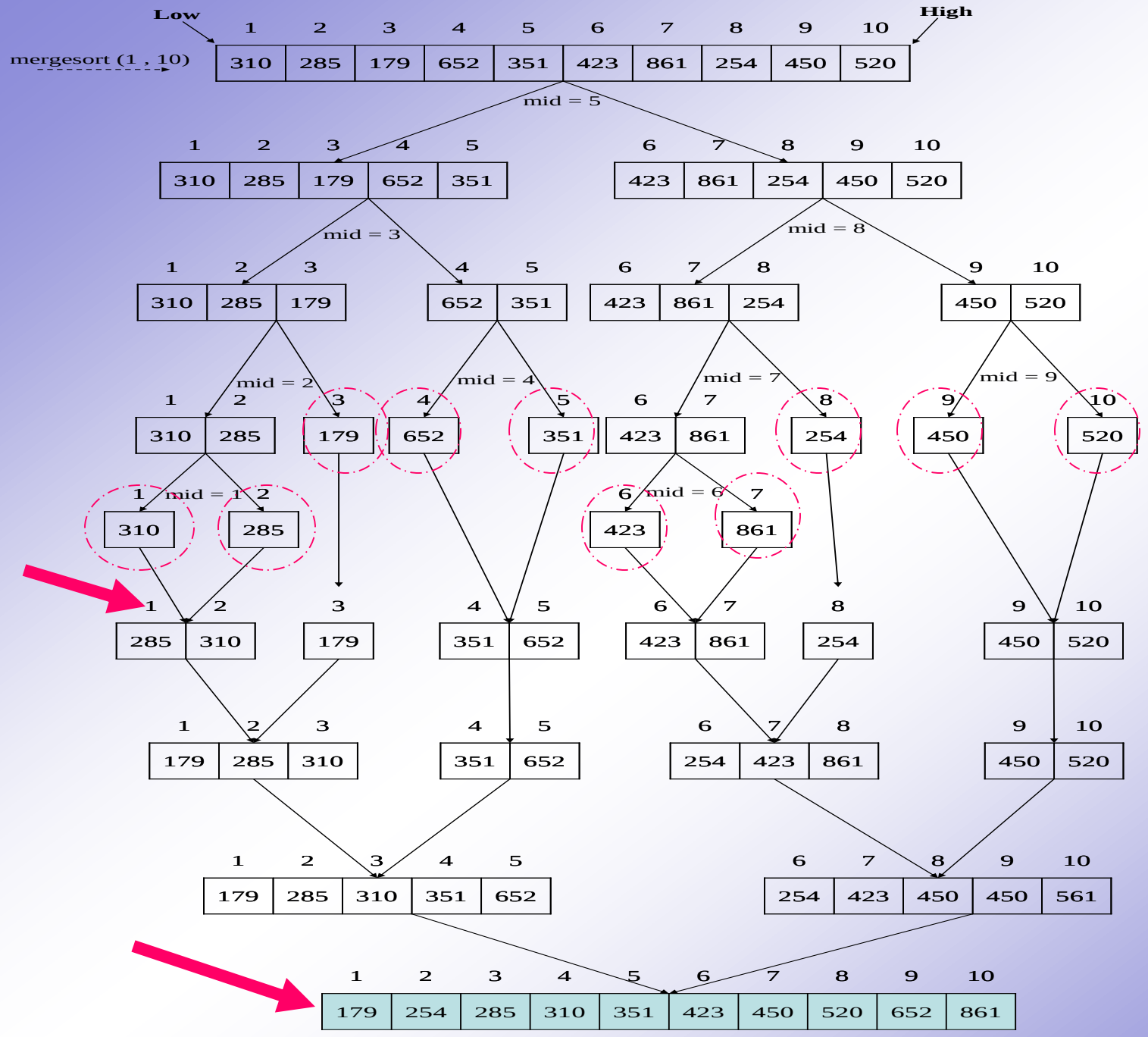


الف) برای مرتب کردن لیست فوق درخت فراخوانی‌های تابع mergesort به صورت زیر می‌باشد:

- تعداد کل فراخوانی‌ها برابر با تعداد small (p) ها + تعداد merge ها یعنی 19 = 9 + 10 می باشد.
- تعداد فراخوانی بازگشتی برابر 18 است .

Low									Hig
1	2	3	4	5	6	7	8	9	10
310	285	179	652	351	423	861	254	450	520




Low
High
mergesort (1 , 10)
1 2 3 4 5 6 7 8 9 10
310 285 179 652 351 423 861 254 450 520
mid = 5
310 285 179 652 351
423 861 254 450 520
mid = 3
mid = 8
310 285 179
652 351
423 861 254
450 520
mid = 2
mid = 4
mid = 7
mid = 9
310 285
179
652
351
423 861
254
450
520
mid = 1
mid = 6
310
285
423
861
285 310
179
351 652
423 861
254
450 520
179 285 310
351 652
254 423 861
450 520
179 285 310 351 652
254 423 450 450 561
179 254 285 310 351 423 450 520 652 861

پیچیدگی الگوریتم merge sort

$$T(n) = \begin{cases} c & n = 1 \\ T(\frac{n}{2}) + T(\frac{n}{2}) + n & \text{else} \end{cases}$$

$$T(n) = n[1 + \log\ n] = n + n\ \log\ n \quad \Rightarrow \quad T(n) \in O(n\ \log\ n)$$

نام الگوریتم	روش حل	بهترین حالت	بدترین حالت	حالت میانگین
merge sort	تقسیم و حل	O(n log n)	O(n log n)	O(n log n)

حل مسأله مرتب‌سازی سریع به روش تقسیم و حل

☺به عنوان مثال فرض کنید آرایه حاوی عناصر زیر باشد. نحوه عمل مرتب‌سازی سریع را نشان می‌دهیم:

15 22 13 27 12 10 20 25

۱- ابتدا محل واقعی عنصر اول یعنی ۱۵ را در لیست مرتب مشخص میکنیم:

عنصر محوری

$$\underbrace{10 \quad 13 \quad 12}_{\text{همه کوچکترند}} \quad \mathbf{15} \quad \underbrace{22 \quad 27 \quad 20 \quad 25}_{\text{همه بزرگترند}}$$

الگوریتم Quick Sort

فرض کنید آرایه ای با نام a داریم که تعداد عناصر آن n می باشد و می خواهیم این عناصر را به ترتیب صعودی مرتب کنیم.
برای اینکار می‌توان از الگوریتم زیر استفاده نمود:

الگوریتم Quick Sort به روش تقسیم و حل

```
Alg QuickSort (p , q)
{
  if (p < q) then                          // !small(p)
  {
     j := partition (p , q + 1) ;          // Divide
     QuickSort (p  ,  j – 1) ;             // recursive   left
     QuickSort (j + 1  ,  q) ;             // recursive   right
  }
}
```

مقایسه ها در تابع partition انجام می شود و در ضمن تابع combine هم نداریم

وظیفه تابع partition پیدا کردن اندیس عنصری مانند K است که تمام عناصر قبل از آن کوچکتر از K و تمام عناصر بعد از آن بزرگتر از K می‌باشند.

تابع partition

الگوریتم partition ()	

Alg **partition** (p , q + 1)
{
V = a[p] ; i = p ; j = q + 1 ;
repeat
{
repeat
i = i + 1 ;
until (a[i] ≥ V);
	V : عنصر محور
	i : اندیس ابتدای لیست
	j : اندیس انتهای لیست
	
	حلقه با اندیس i عناصر سمت چپ وحلقه با

repeat
j = j – 1 ;
until (a[j] ≤ V) ;
 	
if (i < j) then **interchange** (a , i , j) ;
 	
} until (i ≥ j) ;
a[p] = a[j] ; a[j] = V ;
return j ;
}
	اندیس j عناصر سمت راست را با عنصر محور
	مقایسه می کند. و در صورت لزوم دو عنصر سمت
	چپ و سمت راست را با یکدیگر جابجا می نماید.

Alg **interchange** (a , i , j)
{
int temp;
temp = a [i] ;
a[i] = a [j] ;
a[j] = temp ;
}
	این دوال دو عنصر a [i] و a [j] را از طریق
	عنصر کمکی temp تعویض می کند.

- خروجی الگوریتم partition اندیس عنصر محور (V)خواهد بود.
- تعداد مقایسه ها در الگوریتم پارتیشن n – 1 می باشد.
- الگوریتم partition در هر مرحله چندین جابجایی عناصر در لیست انجام می‌دهد.
- تعداد جابجایی‌ها در الگوریتم پارتیشن به چگونگی قرار گرفتن عناصر لیست نامرتب بستگی دارد.
- پیچیدگی الگوریتم partition از مرتبه O(n) می باشد.
- در هر مرحله که الگوریتم اجرا می‌شود، یکی از عناصر در جای اصلی خود fix خواهد شد.

لیست زیر را با استفاده از الگوریتم Quick Sort مرتب کنید.

65 70 75 85 55 50 45

جواب:

اولین فراخوانی به صورت QuickSort(1 , 7) می باشد. عنصر محور پیدا کرده و لیست را به دو قسمت تقسیم می کنیم:

1	2	3	4	5	6	7
65	70	75	85	55	50	45

اولین اجرای تابع partition برای مشخص کردن محل دقیق عنصر 65در لیست می‌باشد، در این فراخوانی جابجایی های زیر انجام میشود:

۱) تعویض ۷۵ و ۵۰

1	2	3	4	5	6	7
65	45	**50**	85	55	**75**	70
		i			j	

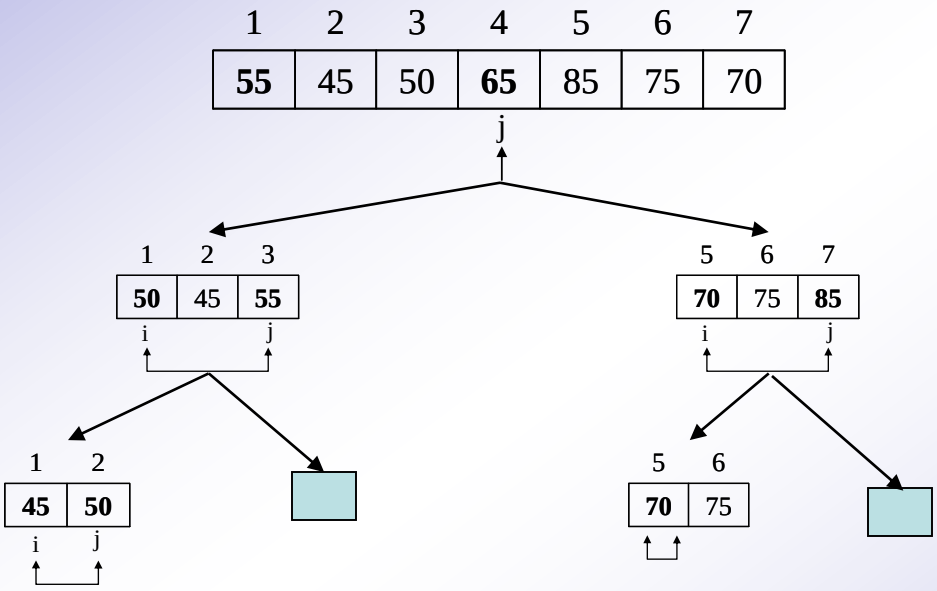
۲) تعویض ۸۵ و ۵۵

1	2	3	4	5	6	7
65	45	50	**55**	**85**	75	70
			i	j		

۳) تعویض ۵۵ و عنصر محور

1	2	3	4	5	6	7
55	45	50	**65**	85	75	70
			j			

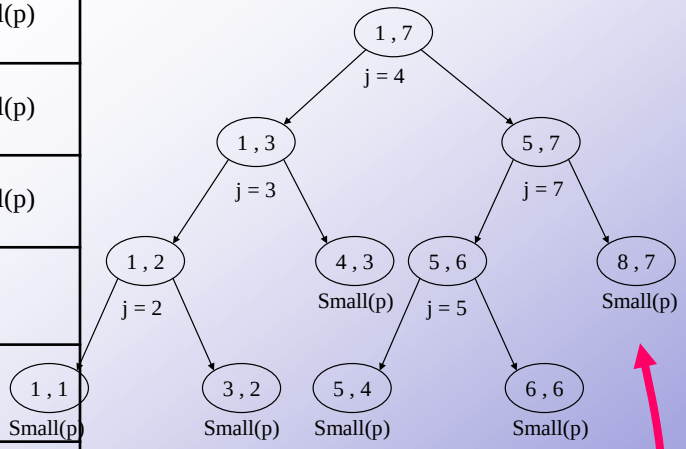
لیست سمت چپ : Quicksort (p , j−1) برابر است با Quicksort (1 , 3) و لیست سمت راست Quicksort (5 , 7) می‌باشد.



1	2	3	4	5	6	7
65	70	75	85	55	50	45

جدول فراخوانيهاي مرحله اي Quick sort طبق درخت فراخواني براي مثال قبل

ترتيب فراخواني	پارامترهاي فراخواني Quick sort	شرط p < q	فراخواني partition	خروجي partition	فراخوانيهاي بازگشتي بعدي (چپ و راست)
1	QuickSort(1 , 7)	True	Partition (1 , 8)	j = 4	QuickSort(1 , 3) QuickSort(5 , 7)
2	QuickSort(1 , 3)	True	Partition (1 , 4)	j = 3	QuickSort(1 , 2) QuickSort(4 , 3)
3	QuickSort(1 , 2)	True	Partition (1 , 3)	j = 2	QuickSort(1 , 1) QuickSort(3 , 2)
4	QuickSort(1 , 1)	False	نداريم	-	small(p) برقرار است.(return)
5	QuickSort(3 , 2)	False	نداريم	-	small(p) برقرار است.(return)
6	QuickSort(4 , 3)	False	نداريم	-	small(p) برقرار است.(return)
7	QuickSort(5 , 7)	True	Partition (5 , 8)	j = 7	QuickSort(5 , 6) QuickSort(8 , 7)
8	QuickSort(5 , 6)	True	Partition (5 , 7)	j = 5	QuickSort(5 , 4) QuickSort(6 , 6)
9	QuickSort(5 , 4)	False	نداريم	-	small(p) برقرار است.(return)
10	QuickSort(6 , 6)	False	نداريم	-	small(p) برقرار است.(return)
11	QuickSort(8 , 7)	False	نداريم	-	small(p) برقرار است.(return)



مرتبه زمانی الگوریتم Quick Sort در حالت متوسط

فرض می‌کنیم عناصر آرایه به طور تصادفی تولید شوند،
بنابراین احتمال آنکه عنصر محور بازگردانده شده، توسط تابع partition ، هر یک از اعداد ۱ تا n باشد، یکسان است .

متوسط = n/2

$$T(n) = T(Left) + T(Right) + T(Partition)$$

$$T(n) = T(n/2) + T(n/2) + O(n)$$

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

مرتبه زمانی الگوریتمQuick Sort در بدترین شرایط

در الگوریتمQuick Sort ، بدترین حالت هنگامی رخ می‌دهد که آرایه از قبل مرتب شده باشد. (به صورت صعودی)

$$T(n) = T(0) + T(n-1) + n - 1$$

T(0) : زمان لازم برای مرتب‌سازی زیرآرایه طرف چپ

T(n – 1) : زمان لازم برای مرتب‌سازی زیرآرایه طرف راست

n – 1 : زمان لازم برای افراز (Partition) . در تابع پارتیشن حداکثر n – 1 مقایسه انجام می شود.

$$T(n) = T(n-1) + n - 1 \qquad \text{for } n > 0$$
$$T(0) = 0$$

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

ضرب سریع ماتریس استراسن به روش تقسیم و حل

- روش استراسن در مورد ضرب ماتریس های ۲×2 ارزش چندانی ندارد.
- تعیین حاصلضرب دو ماتریس n ×n که در آن n توانی از 2 است
- پیچیدگی این الگوریتم از لحاظ تعداد ضرب، جمع و تفریق مورد نیاز کمتر از روش استاندارد است.

جدول مقایسه دو الگوریتم ضرب ماتریس‌های n×n		
نام الگوریتم	تعداد ضرب ها	تعداد جمع و تفریق ها
الگوریتم استاندارد ضرب	n^3	$n^3 - n^2$
الگوریتم استراسن	$n^{2.81}$	$6n^{2.81} - 6n^2$

حل مسأله ضرب سریع ماتریس استراسن به روش تقسیم و حل

حاصل ضرب دو ماتریس 2 × 2 یک ماتریس 2 × 2 است که به صورت زیر نشان داده می‌شود:

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} * \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

مؤلفه‌های حاصل عبارتند از:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \qquad i = 1,2 \qquad j = 1,2 \quad , \quad n = 2$$

در حالت ضرب ماتریس به روش استاندارد با n= 3 به تعداد n^3 عمل ضرب عددی نیاز می باشد.

$$M_1 = (a_{11} + a_{22})(b_{11} + b_{22})$$
$$M_2 = (a_{21} + a_{22})b_{11}$$
$$M_3 = a_{11}(b_{12} - b_{22})$$
$$M_4 = a_{22}(b_{21} - b_{11})$$
$$M_5 = (a_{11} + a_{12})b_{22}$$
$$M_6 = (a_{21} - a_{11})(b_{11} + b_{12})$$
$$M_7 = (a_{12} - a_{22})(b_{21} + b_{22})$$

استراسن معین کرد که اگر فرض کنیم:

صل ضرب C به صورت زیر داده می‌شود: دیده می شود که تعداد ضرب ها ۷ است

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

$$\begin{array}{c} \leftarrow n/2 \rightarrow \\ n/2 \updownarrow \end{array} \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$A_{11} = \begin{bmatrix} a_{11} & a_{11} & \dots & a_{1,\frac{n}{2}} \\ a_{21} & a_{22} & \dots & a_{2,\frac{n}{2}} \\ & \cdot & & \\ & \cdot & & \\ & \cdot & & \\ a_{\frac{n}{2},1} & & \dots & a_{\frac{n}{2},\frac{n}{2}} \end{bmatrix}$$

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

به همین شیوه M2 تا M7 را محاسبه می‌کنیم، سپس رابطه زیر را بدست می‌آوریم و با استفاده از فرمول استراسن خواهیم داشت :

$$\boxed{c_{11} = M_1 + M_4 - M_5 + M_7}$$

الگوریتم ضرب ماتر... strassen

شکل کامل‌تر الگوریتم استراسن

```
n × n-matrix  strassen (int  n ,  n × n-matrix A ,  n × n-matrix B )
{
   if ( n <= threshold)  compute C = A × B using the standard algorithm;
   else  {
Partition A,B into four  submatrices  A11 , A12 , A21 , A22 ; B11 , B12 , B21 , B22 ;
```

M_1=**strassen** $(\frac{n}{2}, A_{11}+A_{22}, B_{11}+B_{22})$;

M_2=**strassen** $(\frac{n}{2}, A_{21}+A_{22}, B_{11})$;

M_3=**strassen** $(\frac{n}{2}, A_{11}, B_{12}-B_{22})$;

M_4=**strassen** $(\frac{n}{2}, A_{22}, B_{21}-B_{11})$;

M_5=**strassen** $(\frac{n}{2}, A_{11}+A_{12}, B_{22})$;

M_6=**strassen** $(\frac{n}{2}, A_{21}-A_{11}, B_{11}+B_{12})$;

M_7=**strassen** $(\frac{n}{2}, A_{12}-A_{22}, B_{21}+B_{22})$;

$C_{11} = M_1 + M_4 - M_5 + M_7$;
$C_{12} = M_3 + M_5$;
$C_{21} = M_2 + M_4$;
$C_{22} = M_1 + M_3 - M_2 + M_6$;

```
   }
   return C;
}
```

پیچیدگی زمانی تعداد ضرب‌ها در الگوریتم استراسن

$$T(n)=\begin{cases}1 & \text{if } n\le 1\\ 7T(\frac{n}{2}) & \text{else}\end{cases} \longrightarrow T(n)=n^{\log 7}\approx n^{2.81}\longrightarrow T(n)\in\theta(n^{2.81})$$

پیچیدگی زمانی تعداد جمع‌ها و تفریق‌ها در الگوریتم strassen

$$T(n)=\begin{cases}1 & \text{if } n\le 1\\ 7T(\frac{n}{2})+18(\frac{n}{2})^2 & \text{else}\end{cases}$$

$$T(n)=6n^{\log 7}-6n^2\approx 6n^{2.81}-6n^2\longrightarrow T(n)\in\theta(n^{2.81})$$

جدول مقایسه دو الگوریتم ضرب ماتریس‌های $n\times n$		
نام الگوریتم	تعداد ضرب ها	تعداد جمع و تفریق ها
الگوریتم استاندارد ضرب	n^3	n^3-n^2
الگوریتم استراسن	$n^{2.81}$	$6n^{2.81}-6n^2$

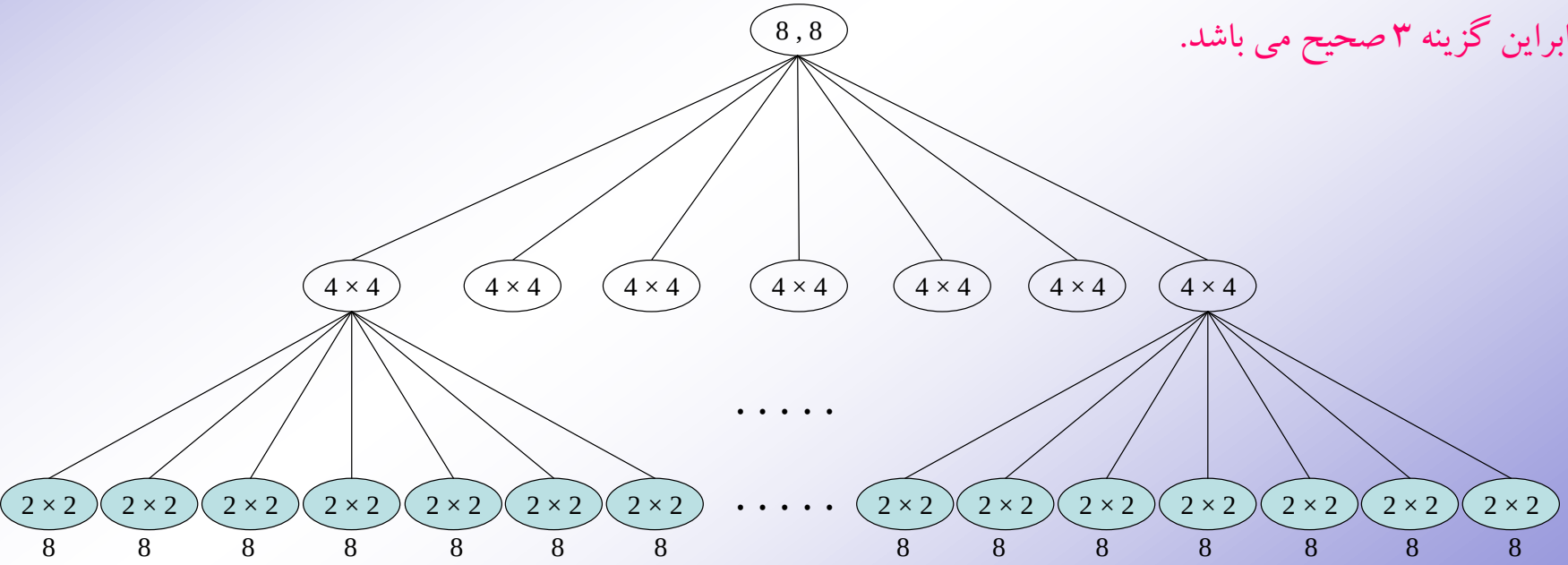
مثال : در ضرب ماتریس‌ها به روش استراسن اگر مسأله کوچک، ضرب ماتریس 2×2 باشد، برای ضرب دو ماتریس 8×8 چند ضرب عددی صورت می‌پذیرد؟ (مهندسی IT– دولتی۸۵)

۱) ۵۷ ۲) ۳٤۳ ۳) ۳۹۲ ٤) ۵۱۲

جواب : اگر فرض کنیم ضرب ماتریس 2×2 به روش غیراستراسن (استاندارد) انجام شود:

$$\begin{cases} T(n) = 7T(\frac{n}{2}) \\ T(2) = 8 \end{cases} \longrightarrow T(8) = 7 \times 7 \times 8 = 392$$

بنابراین گزینه ۳ صحیح می باشد.



نمونه مسائلی که نمی توان از روش تقسیم وحل استفاده کرد

1. مسائلی با اندازه n به چند زیر مساله با اندازه تقریبا برابر n تبدیل می شود ← مرتبه اجرای نمایی خواهد داشت
2. مسائلی با اندازه n به n زیر مساله با اندازه n/c تقسیم می شود ← مرتبه اجرای n^{logn} خواهد داشت

مثال:

1. فیبوناتچی

الگوریتم2-9: ضرب اعداد صحیح بزرگ

مسئله: ضرب دو عدد صحیح بزرگ u و v

```
large _ integer prod ( large_integer u, large_integer v)
{
  large_inreger x , y , w , z ;
  int n , m ;
 n = maximum(number of digits in u,number of digits in v)
 if  (u = = 0 || v = = 0)
      return 0 ;
```

```
else if    (n < = threshold)
    return   u × v obtained in the usual way;
else {
      m = ⌊ n / 2 ⌋;
       x = u divide 10 ^ m ;  y = rem 10 ^ m;
      w = v divide 10 ^ m ;  z = rem 10 ^ m;

       return prod (x ,w) × 10 ^2m +  ( prod ( x, z) + prod
(w, y )) × 10 ^ m + prod ( y, z);
    }
}
```

تحلیل پیچیدگی زمانی در بدترین حالت برای ا لگوریتم(ضرب اعداد صحیح)

عمل اصلی: دستکاری یک رقم دهدهی در یک عدد صحیح بزرگ در هنگام جمع کردن ، تفریق کردن، یا انجام اعمال **divide 10 ^ m**، **rem 10 ^m** یا **×10 ^ m**. هر یک از این اعمال را m بار انجام می دهد.

اندازه ورودی: n ، تعداد ارقام هر یک از دو عدد صحیح.

به ازای $n > s$ که n توانی از 2است $T(n) = 4T(n/2) + cn$

$T(s) = 0$

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

در چه مسائلی نمی توان از روش تقسیم وحل استفاده کرد

- 1- مسایلی با اندازه n به چند زیر مسئله تقسیم می شود که اندازه زیر مسئله ها نیز تقریبا برابر n است.

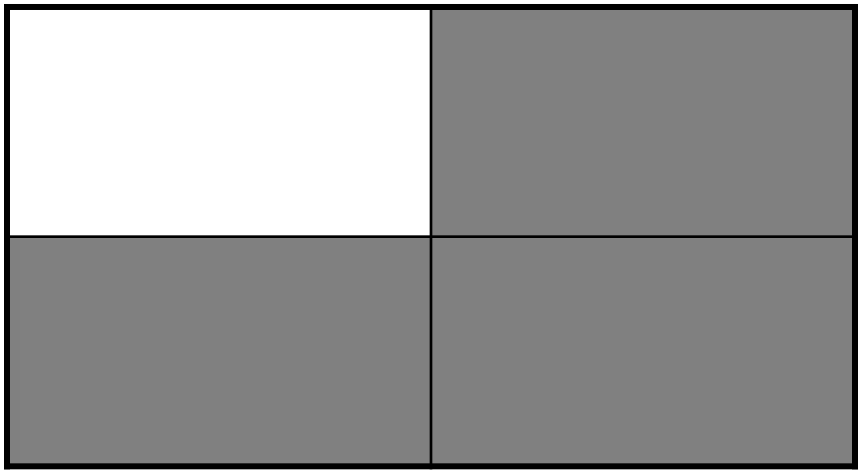
زمان نمایی ایجاد می کند.

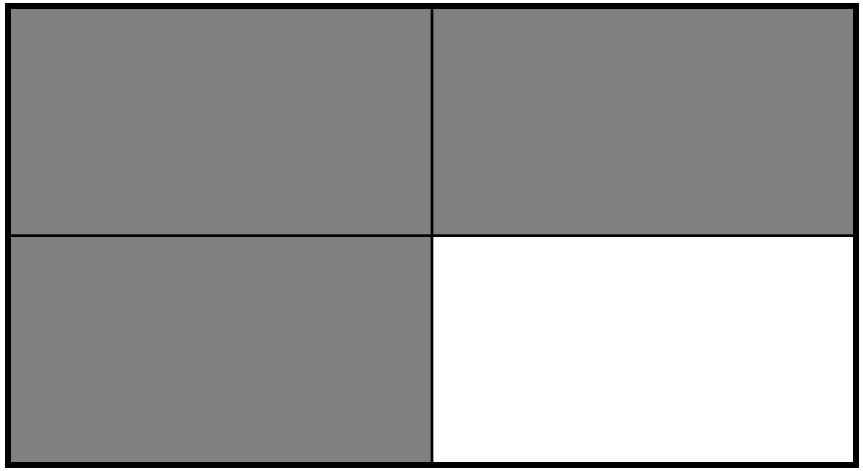
- 2- مساله ای با اندازه n تقریبا به اندازه n زیر مسئله با اندازه n/c که در آن c ثابت است تقسیم می شود.

زمان $n\log^n$ ایجاد می کند.

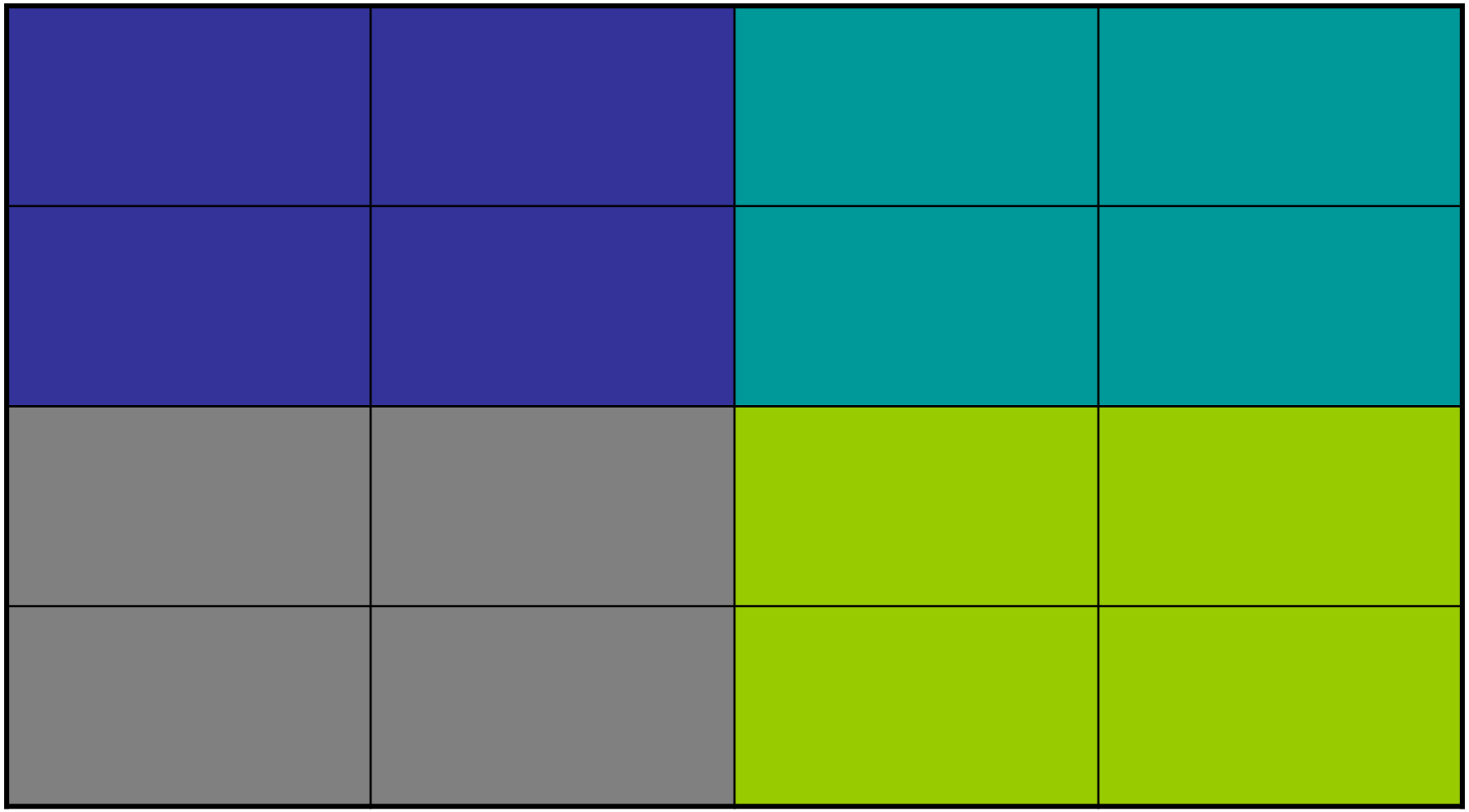
فرش کردن صفحه شطرنجی

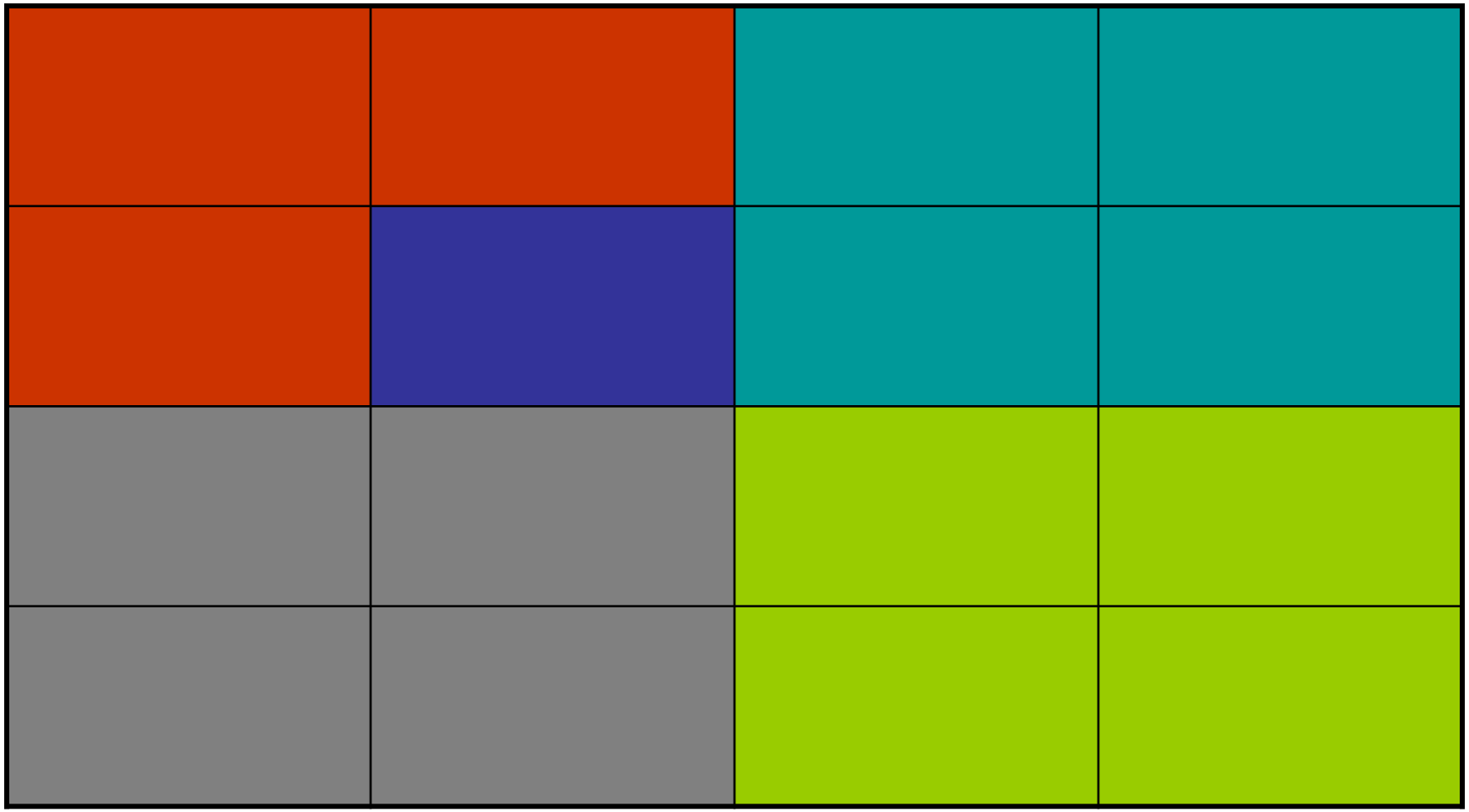
- ابعاد صفحه 2^k*2^k است.
- موزاییک به شکل **L** است.
- در بدترین حالت یکی از خانه ها خالی می ماند.
-

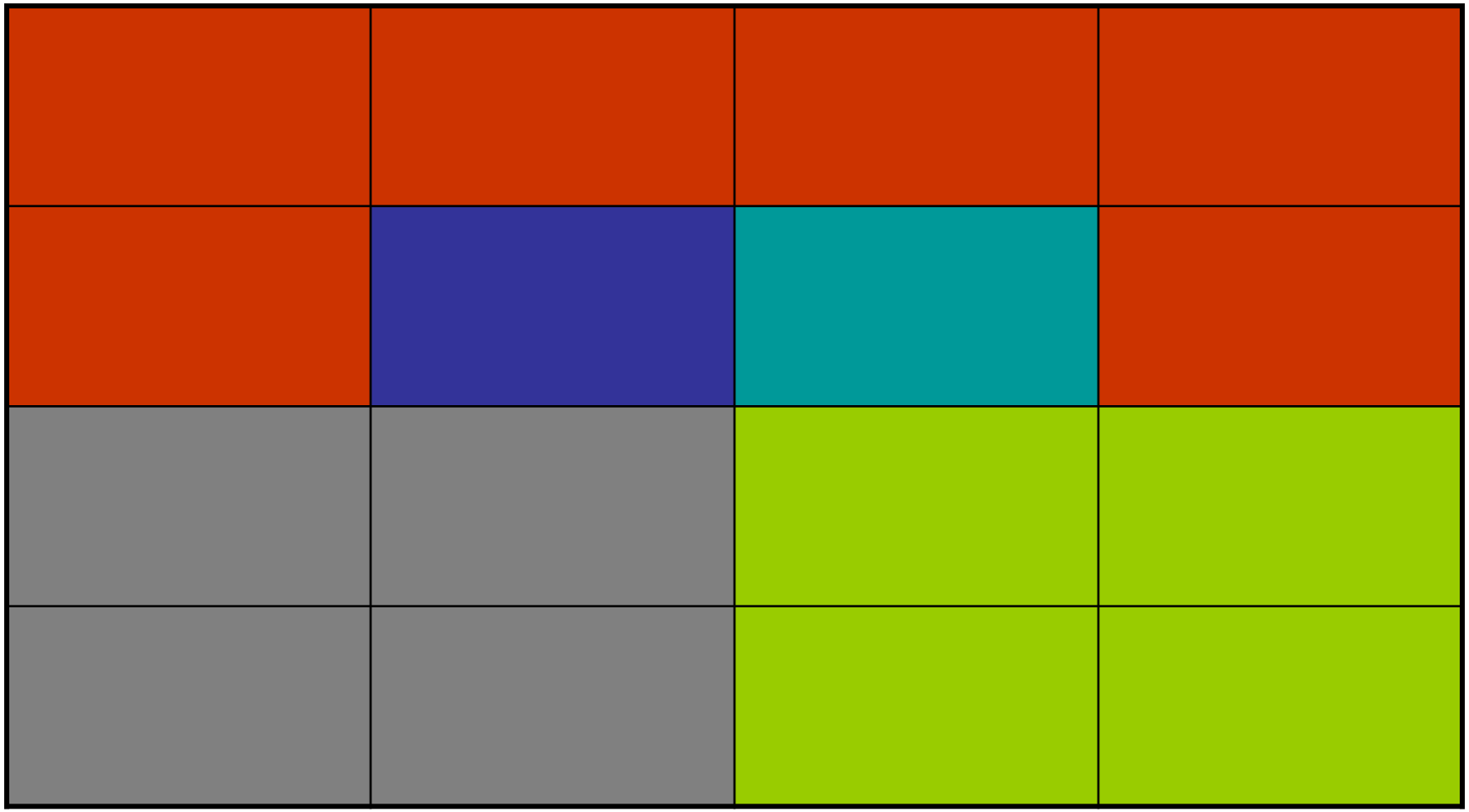


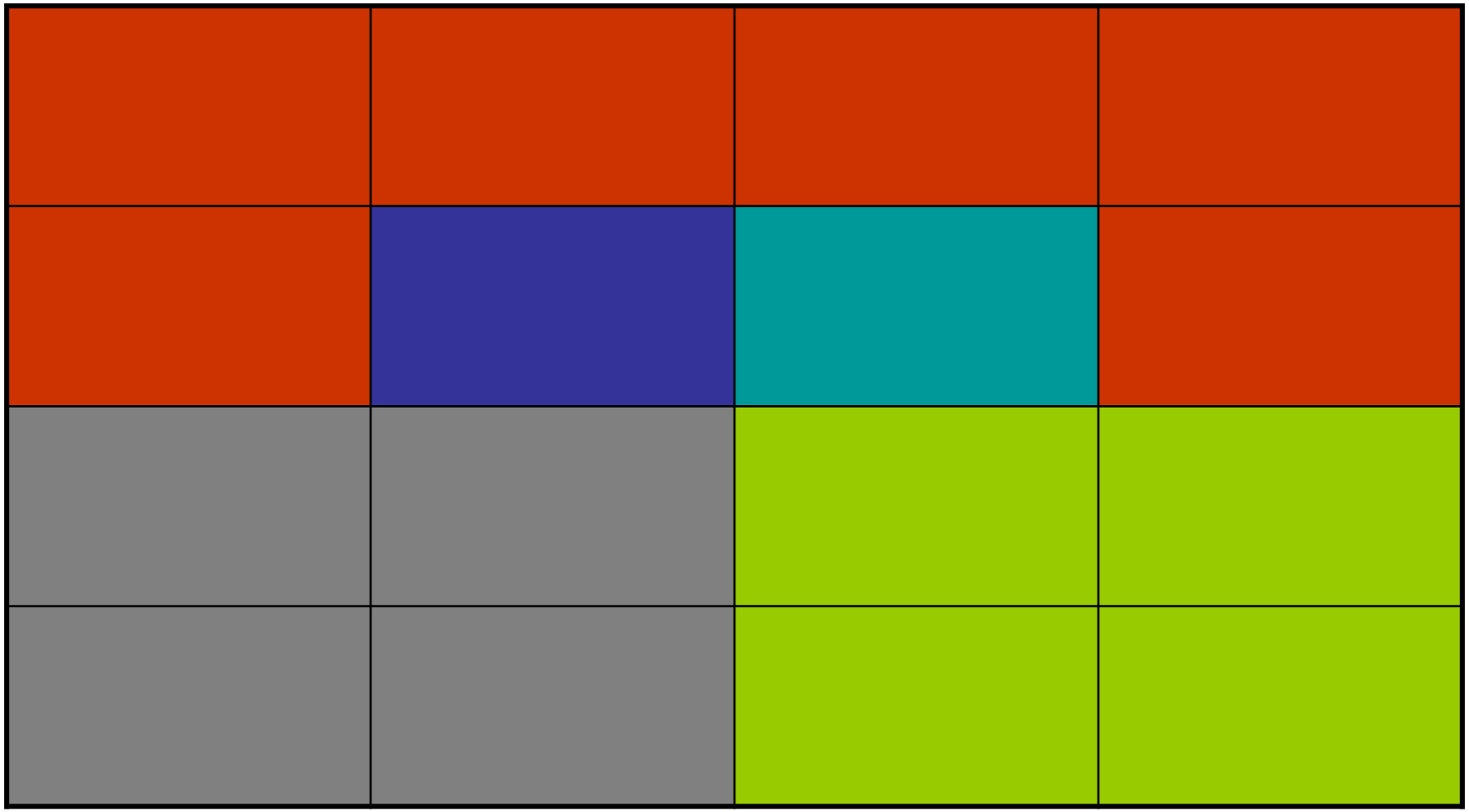


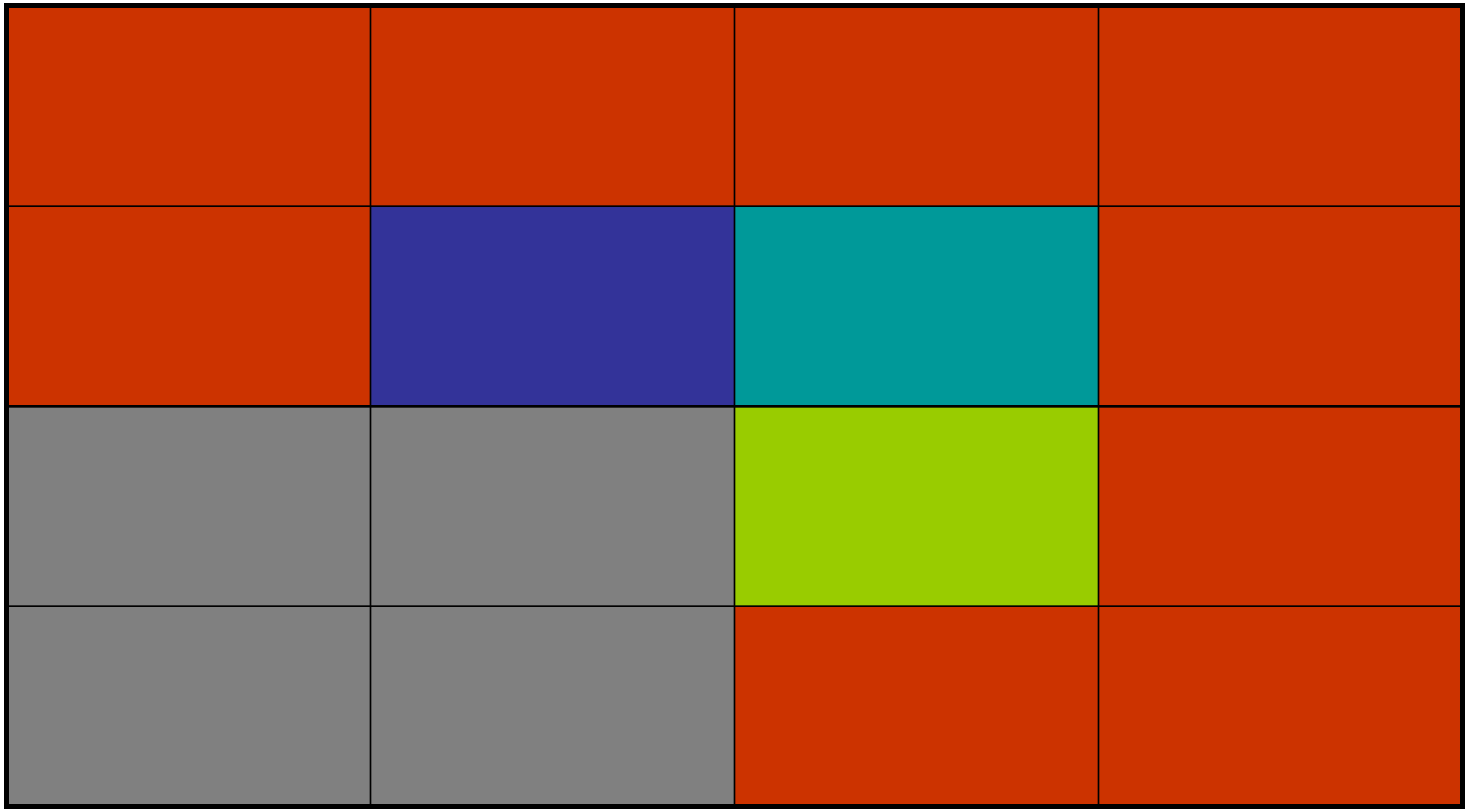
4*4

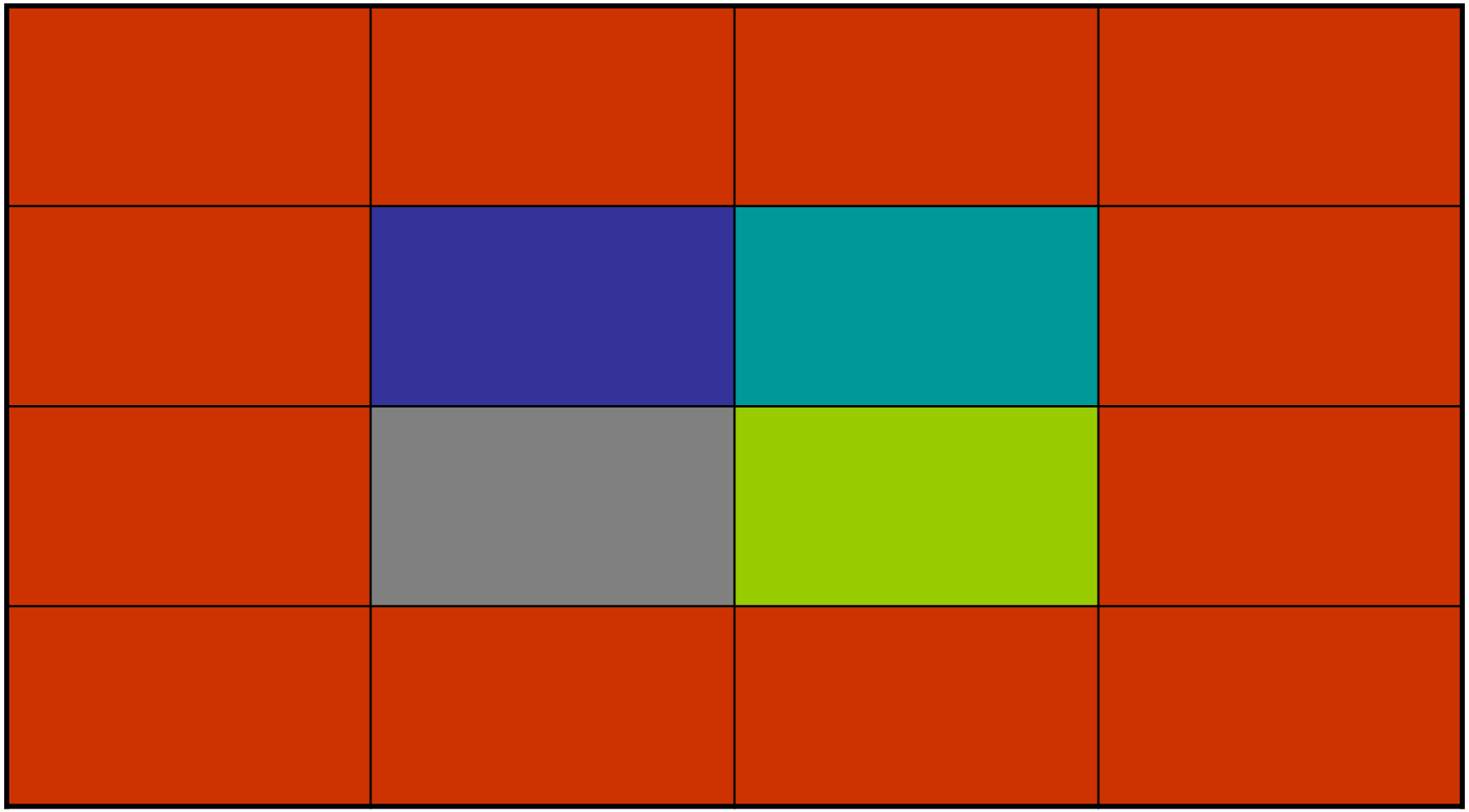


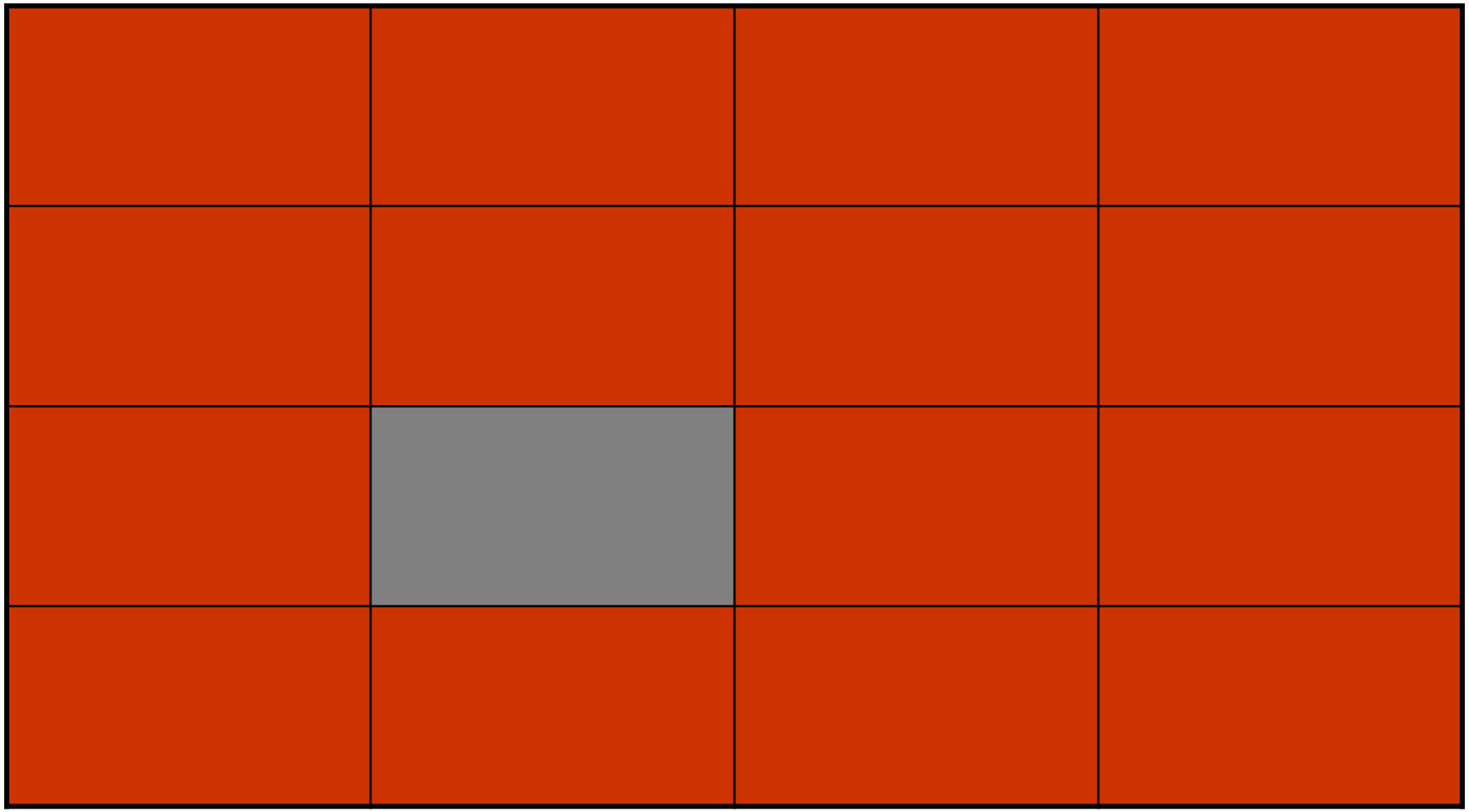












ضرب چند جمله ای ها

- $P(x)=a_n x^n+a_{n-1}x^{n-1}+a^{n-2}x^{n-2} \ . . . \ a_2x^2 + a_1x^1+a_0$
- $Q(x)=b_n x^n+b_{n-1}x^{n-1}+b^{n-2}x^{n-2} \ . . . \ b_2x^2 + b_1x^1+b_0$

- $R(x)= P(x)Q(x)$

- $P(x)=a_n x^n+a_{n-1}x^{n-1}+a^{n-2}x^{n-2} \ . . . \ a_2x^2 + a_1x^1+a_0$
- $Q(x)=b_n x^n+b_{n-1}x^{n-1}+b^{n-2}x^{n-2} \ . . . \ b_2x^2 + b_1x^1+b_0$
- $R(x)= P(x)Q(x)=$

$a_0b_0 + a_0b_1x^1+a_0b_2x^2 + . .+ a_0b_{n-2}x^{n-2} + a_0b_{n-1}x^{n-1} + a_0b_nx^n$

$a_1b_0x+ a_1b_1x^2+a_1b_2x^3 + . .+ a_1b_{n-2}x^{n-1} + a_1b_{n-1}x^n + a_1b_nx^{n+1}$

$a_2b_0 x^2+ a_2b_1x^3+a_2b_2x^4 + . .+ a_2b_{n-2}x^n + a_2b_{n-1}x^{n+1} + a_2b_nx^{n+2}$

.

.

.

- $R(x)= P(x)Q(x)=$

$a_0b_0 + a_0b_1x^1+a_0b_2x^2 + . .+ a_0b_{n-2}x^{n-2} + a_0b_{n-1}x^{n-1} + a_0b_nx^n$

$a_1b_0x+ a_1b_1x^2+a_1b_2x^3 + . .+ a_1b_{n-2}x^{n-1} + a_1b_{n-1}x^n + a_1b_nx^{n+1}$

$a_2b_0 x^2+ a_2b_1x^3+a_2b_2x^4 + . .+ a_2b_{n-2}x^n + a_2b_{n-1}x^{n+1} + a_2b_nx^{n+2}$

.

.

.

a_0b_0

- $R(x)= P(x)Q(x)=$

$a_0b_0 + a_0b_1x^1+a_0b_2x^2 + . .+ a_0b_{n-2}x^{n-2} + a_0b_{n-1}x^{n-1} + a_0b_nx^n$

$a_1b_0x+ a_1b_1x^2+a_1b_2x^3 + . .+ a_1b_{n-2}x^{n-1} + a_1b_{n-1}x^n + a_1b_nx^{n+1}$

$a_2b_0 x^2+ a_2b_1x^3+a_2b_2x^4 + . .+ a_2b_{n-2}x^n + a_2b_{n-1}x^{n+1} + a_2b_nx^{n+2}$

.

.

.

$a_0b_0 + (a_1b_0+a_0b_1)x +$

- $R(x)= P(x)Q(x)=$

$a_0b_0 + a_0b_1x^1+a_0b_2x^2 + . .+ a_0b_{n-2}x^{n-2} + a_0b_{n-1}x^{n-1} + a_0b_nx^n$

$a_1b_0x+ a_1b_1x^2+a_1b_2x^3 + . .+ a_1b_{n-2}x^{n-1} + a_1b_{n-1}x^n + a_1b_nx^{n+1}$

$a_2b_0 x^2+ a_2b_1x^3+a_2b_2x^4 + . .+ a_2b_{n-2}x^n + a_2b_{n-1}x^{n+1} + a_2b_nx^{n+2}$

.

.

.

$a_0b_0 + (a_1b_0+a_0b_1)x + (a_0b_2 + a_1b_1 + a_2b_0)x^2 +$

- $R(x)= P(x)Q(x)=$

$a_0b_0 + a_0b_1x^1+a_0b_2x^2 + . .+ a_0b_{n-2}x^{n-2} + a_0b_{n-1}x^{n-1} + a_0b_nx^n$

$a_1b_0x+ a_1b_1x^2+a_1b_2x^3 + . .+ a_1b_{n-2}x^{n-1} + a_1b_{n-1}x^n + a_1b_nx^{n+1}$

$a_2b_0 x^2+ a_2b_1x^3+a_2b_2x^4 + . .+ a_2b_{n-2}x^n + a_2b_{n-1}x^{n+1} + a_2b_nx^{n+2}$

.

.

.

$a_0b_0 + (a_1b_0+a_0b_1)x + (a_0b_2 + a_1b_1 + a_2b_0)x^2 + . . .$

- k جمله ام $=c_i= \sum_{i=0}^{k} a_ib_{k-i}$

- $R(x)= P(x)Q(x)= \sum_{i=0}^{2n} c_ix^i$

- $R(x)= P(x)Q(x)= \sum_{i=0}^{2n} c_i x^i$

نیاز به دو for برای محاسبه بنابراین : $O(n^2)$

روش تقسیم وحل
برای ضرب چند جمله ایها

- $R(x)= P(x)Q(x)$

- $P(x) =Ax^{(n/2)} + B$
- $Q(x) = C\ x^{(n/2)} + D$
- A , B ,C ,D جند جمله ای از مرتبه n/2 است.
- $R(x)= P(x)Q(x)=(Ax^{(n/2)}+B)(Cx^{(n/2)} +D)$
- $= ACx^{n} +(AD+BC)x^{(n/2)}+BD$
- تعداد ضرب ها 4 تا است.
- عمل ضرب $T(n)=4T(n/2)+cn$
- $T(n)=\theta(n^2)$
- الگوریتم بهتر نشده است.

- $R(x)= P(x)Q(x)=(Ax^{(n/2)}+B)(Cx^{(n/2)} +D)$
- $= ACx^n +(AD+BC)x^{(n/2)}+BD$

- $AD+BC=(A-B)(D-C)+AC+BD$

- $R(x)= ACx^n +((A-B)(D-C)+AC+BD)x^{(n/2)}+BD$
- $R(x)= \underline{AC}x^n +((A-B)(D-C)+\underline{AC}+\underline{BD})x^{(n/2)}+\underline{BD}$
- سه ضرب لازم است.

- $T(n)=3T(n/2)+cn$
- $T(n)= \theta(n^{\log 3})$
- عملکرد بهتر است.