روش تقسیم‌وحل در طراحی الگوریتم‌ها

- **تقسیم** (divide): **تقسیم مسئله به تعدادی زیرمسئله‌ی مشابه با اندازه‌ی کوچک‌تر**
- **حل** (conquer): **حل زیر مسئله‌ها به‌صورت بازگشتب**
- **نرکیب نتایج** (combine)

شما با این الگوریتم‌های تقسیم‌وحل آشنا هستید

- مرتب‌سازی ادغامی
- مرتب‌سازی سریع
- جست‌وجوی دودویی

با این مسئله‌های و راه‌حل‌های تقسیم‌وحل آشنا خواهید شد

- زمان‌بندی بازی‌ها
- نمای برج‌ها
- نزدیک‌ترین نقطه‌ها
- یافتن همه‌ی وارونگی‌ها
- ضرب دو چند جمله‌ای
- ضرب ماتریس‌ها (الگوریتم استراسون)

زمان‌بندی دوره‌ی بازی‌ها

n تیم در یک بازی دوره‌ای
برنامه‌ی بازی‌ها با حداقل مدت را طوری طراحی کنید که در آن

- هر تیم با بقیه‌ی تیم‌ها بازی کند،
- هر تیم در هر روز بیش از یک بازی انجام ندهد.

تعداد کل بازی‌ها: $n(n-۱)/۲$

در هر روز حداکثر تعداد بازی‌ها: $\lfloor \frac{n}{۲} \rfloor$

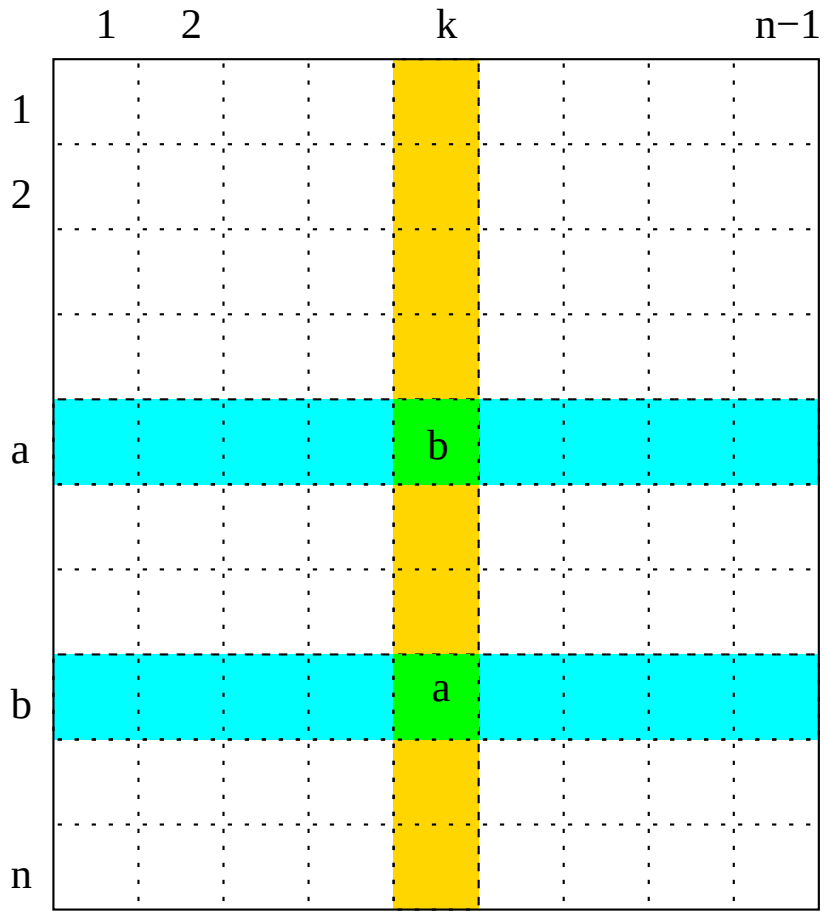
$\Longleftarrow$ **کران پایین تعداد روزهای بازی:**

اگر n زوج باشد برابر $n-۱$ و اگر n فرد باشد برابر n روز است.

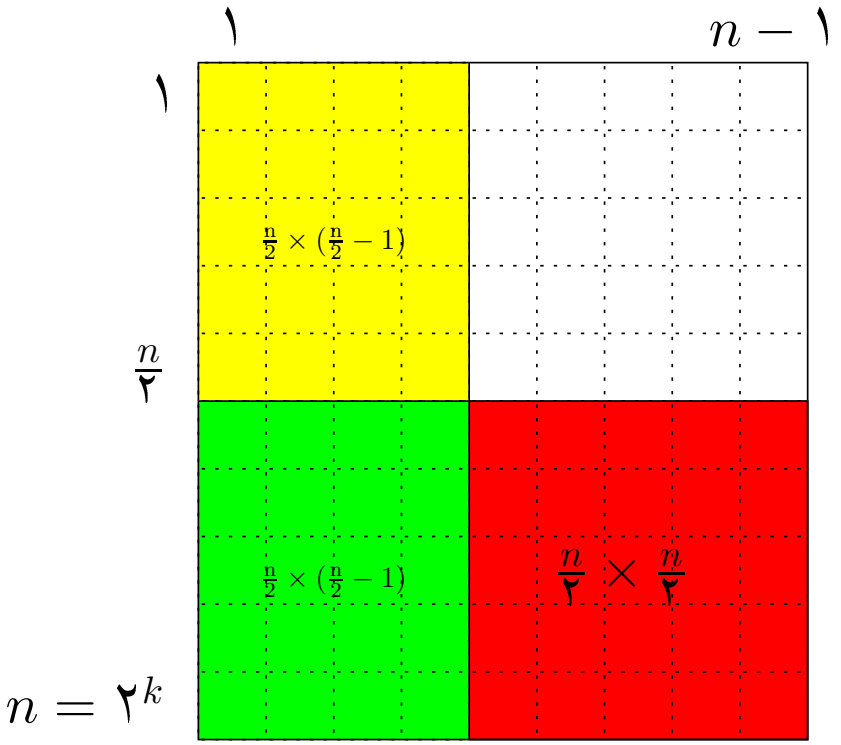
نحوه‌ی نمایش: با یک ماتریس

- در سطر iام اعداد ۱ تا n به‌جز i قرار دارند.
- درهیچ سطر و ستونی عدد تکراری نداریم.
- رابطه‌ی زیر صادق است:

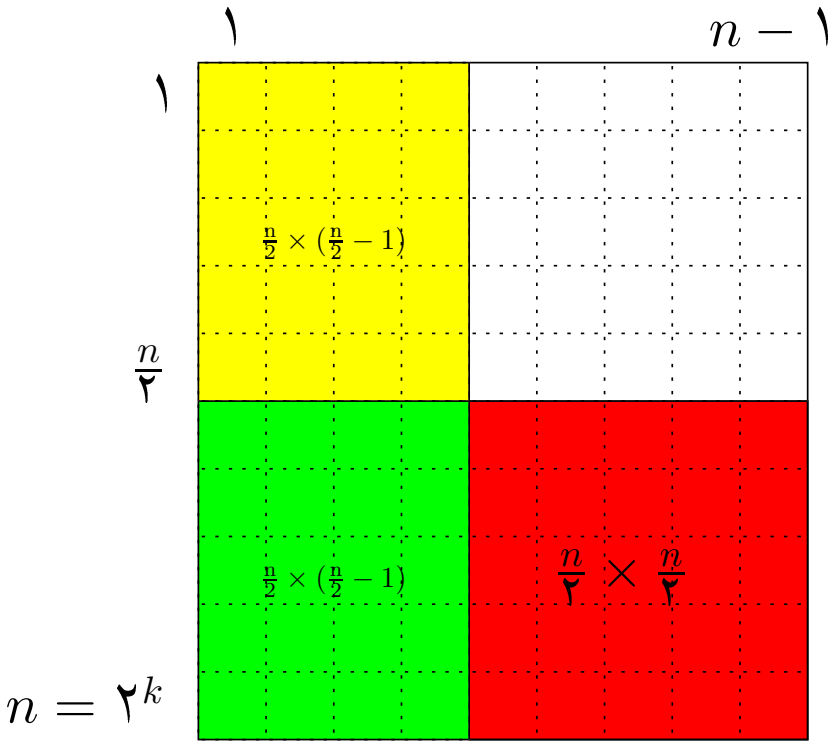
$$A[b,i] = a \Leftrightarrow A[a,i] = b$$


1
2
k
n−1
1
2
a
b
b
a
n

زمان‌بندی دوره‌ی بازی‌ها برای $n = ۲^k$ تیم

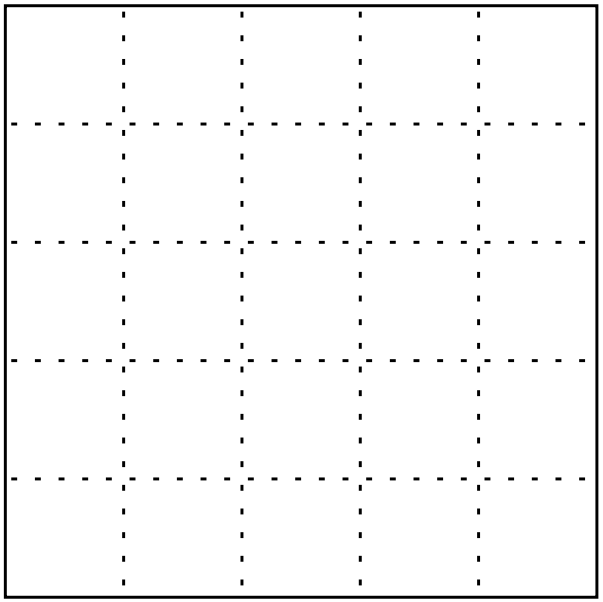


زمان‌بندی دوره‌ی بازی‌ها برای $n = ۲^k$ تیم

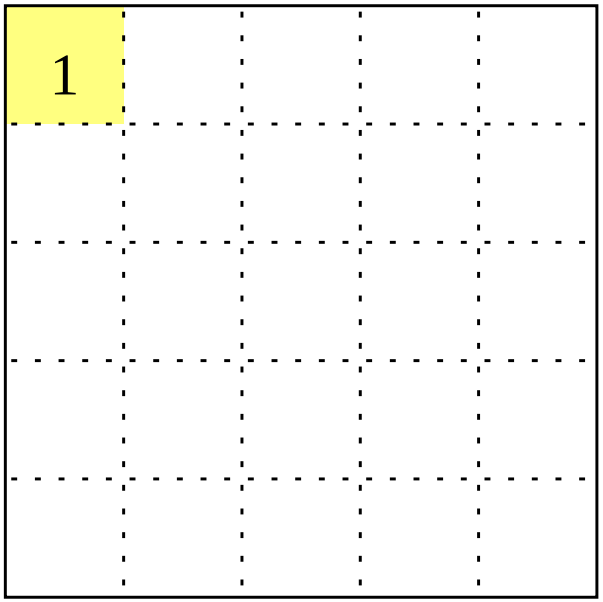


مربع لاتین

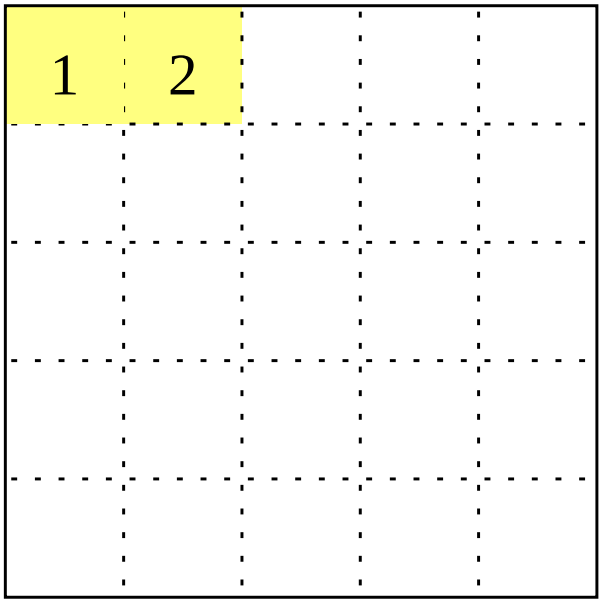
الگوریتمی برای ساختن مربع لاتین



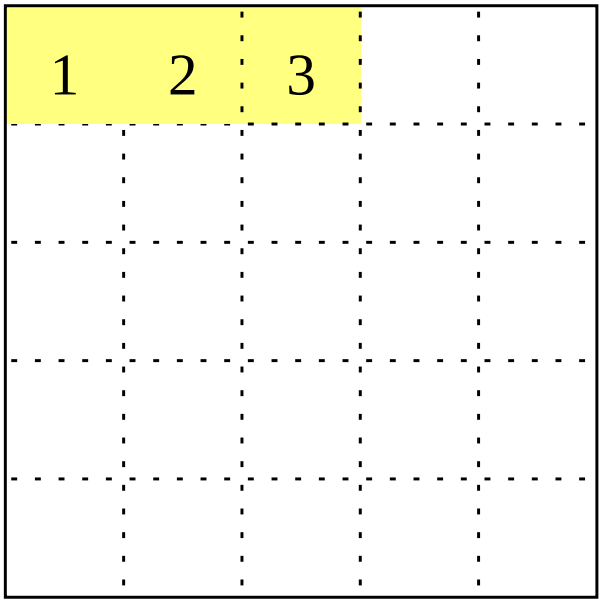
الگوریتمی برای ساختن مربع لاتین



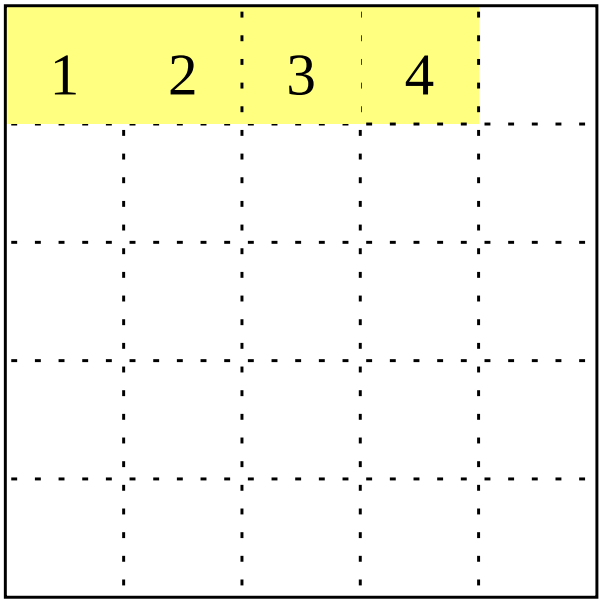
الگوریتمی برای ساختن مربع لاتین



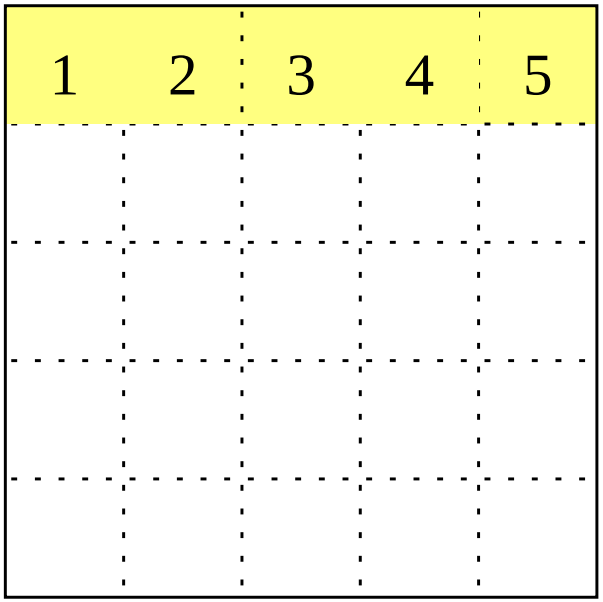
الگوریتمی برای ساختن مربع لاتین



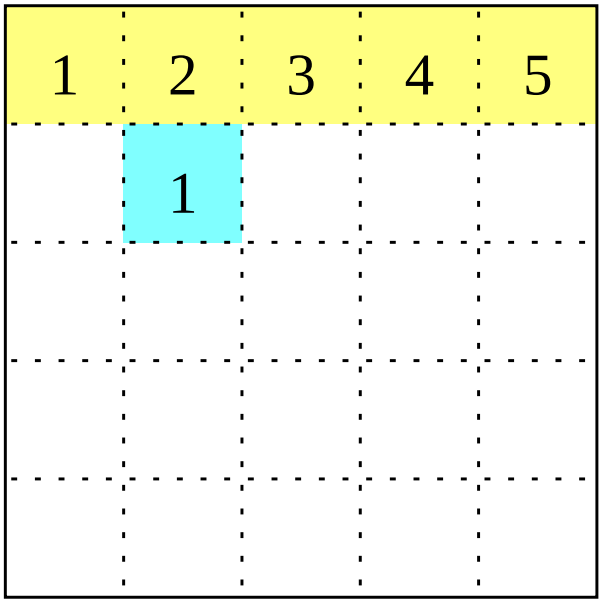
الگوریتمی برای ساختن مربع لاتین



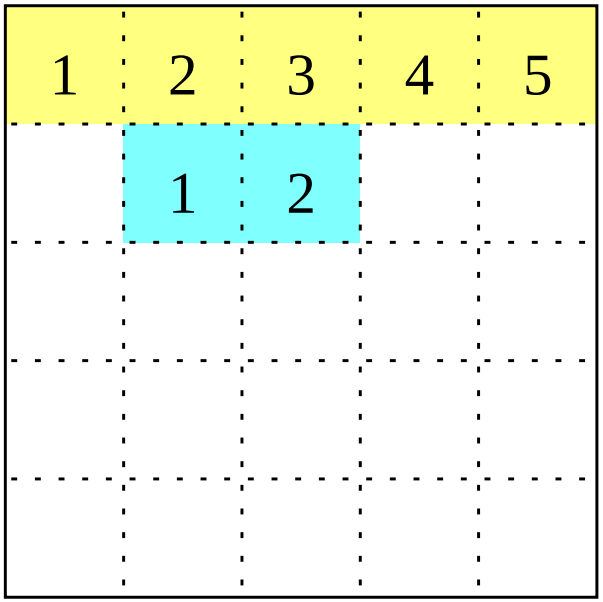
الگوریتمی برای ساختن مربع لاتین



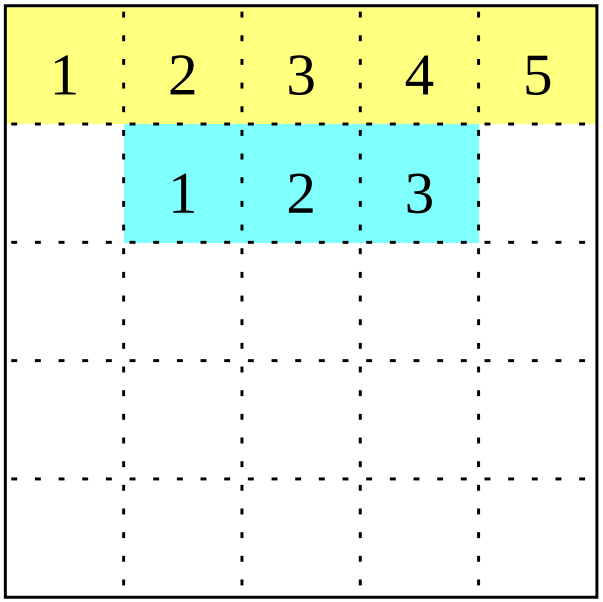
الگوریتمی برای ساختن مربع لاتین



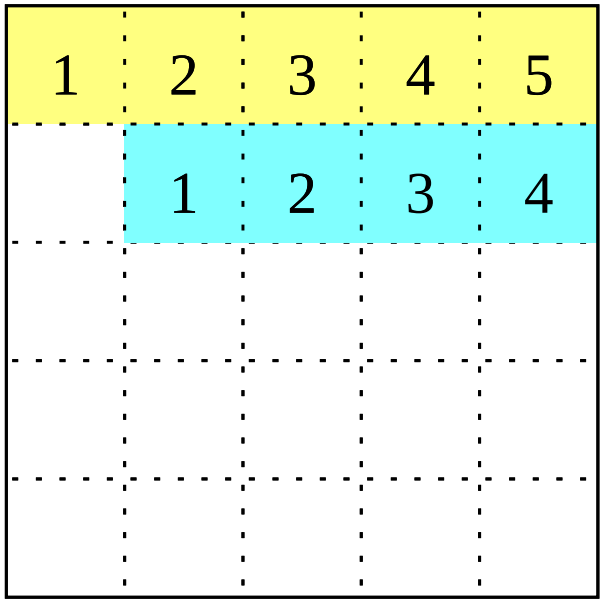
الگوریتمی برای ساختن مربع لاتین



الگوریتمی برای ساختن مربع لاتین



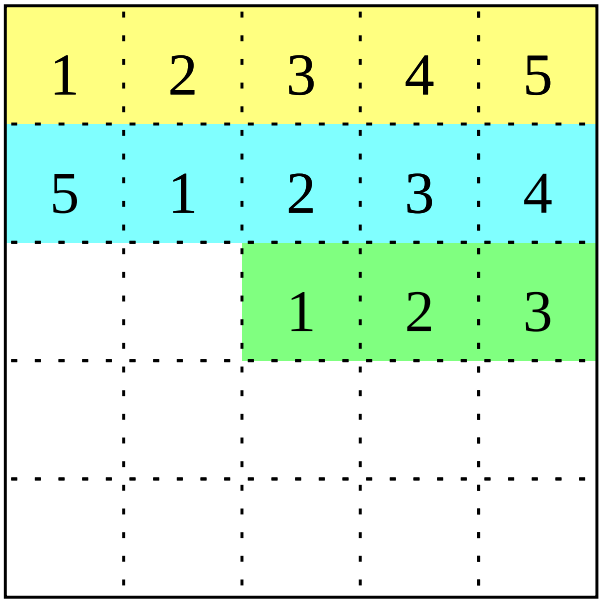
الگوریتمی برای ساختن مربع لاتین



الگوریتمی برای ساختن مربع لاتین

1	2	3	4	5
5	1	2	3	4

الگوریتمی برای ساختن مربع لاتین



الگوریتمی برای ساختن مربع لاتین

1	2	3	4	5
5	1	2	3	4
4	5	1	2	3

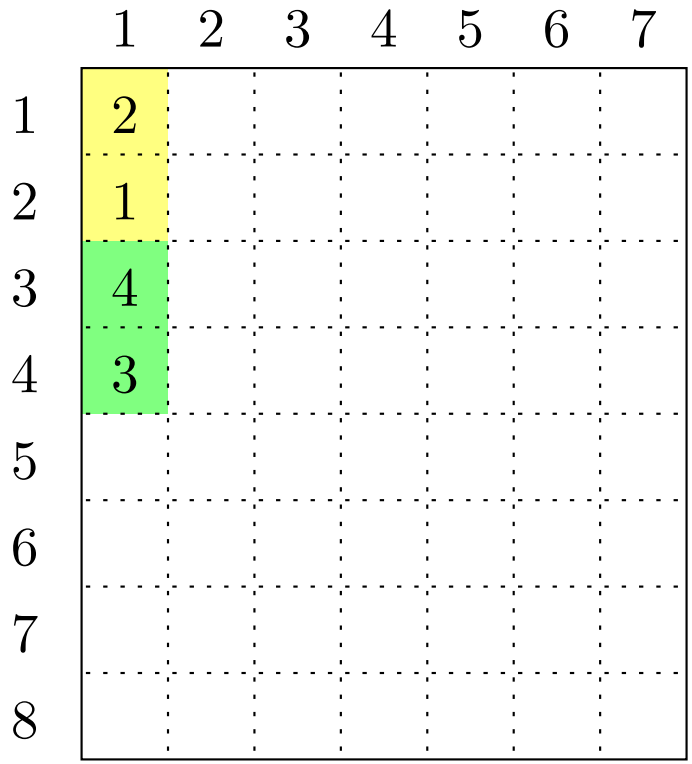
الگوریتمی برای ساختن مربع لاتین

1	2	3	4	5
5	1	2	3	4
4	5	1	2	3
3	4	5	1	2

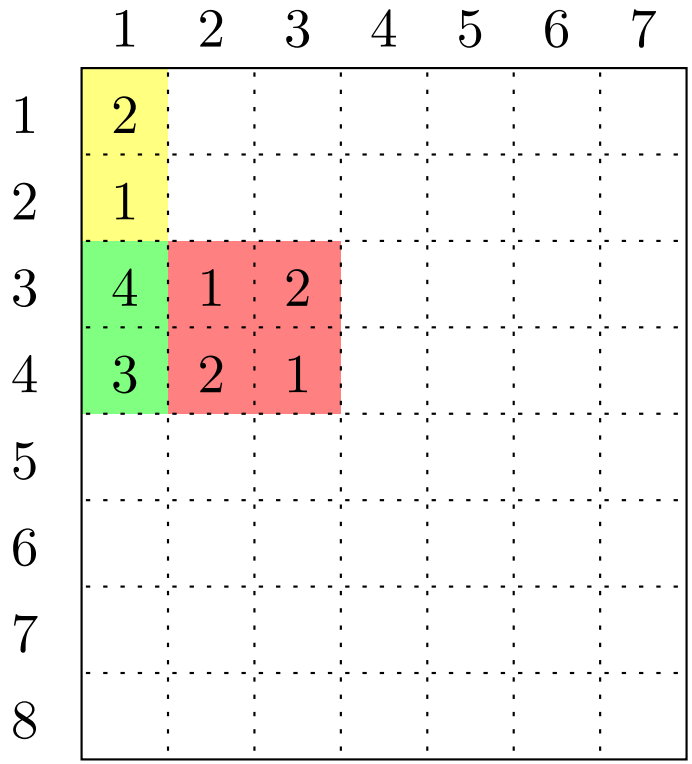
الگوریتمی برای ساختن مربع لاتین

1	2	3	4	5
5	1	2	3	4
4	5	1	2	
3	4	5	1	2
2	3	4	5	1

مثال برای $n = 8$



مثال برای $n = 8$



مثال برای $n = 8$

	1	2	3	4	5	6	7
1	2	3	4				
2	1	4	3				
3	4	1	2				
4	3	2	1				
5							
6							
7							
8							

مثال برای $n = 8$

	1	2	3	4	5	6	7
1	2	3	4				
2	1	4	3				
3	4	1	2				
4	3	2	1				
5	6	7	8				
6	5	8	7				
7	8	5	6				
8	7	5	4				

مثال برای $n = 8$

	1	2	3	4	5	6	7
1	2	3	4				
2	1	4	3				
3	4	1	2				
4	3	2	1				
5	6	7	8	1	2	3	4
6	5	8	7	4	1	2	3
7	8	5	6	3	4	1	2
8	7	5	4	2	3	4	1

مثال برای $n = 8$

	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	1	4	3	8	5	6	7
3	4	1	2	7	8	5	6
4	3	2	1	6	7	8	5
5	6	7	8	1	2	3	4
6	5	8	7	4	1	2	3
7	8	5	6	3	4	1	2
8	7	5	4	2	3	4	1

حالت کلی

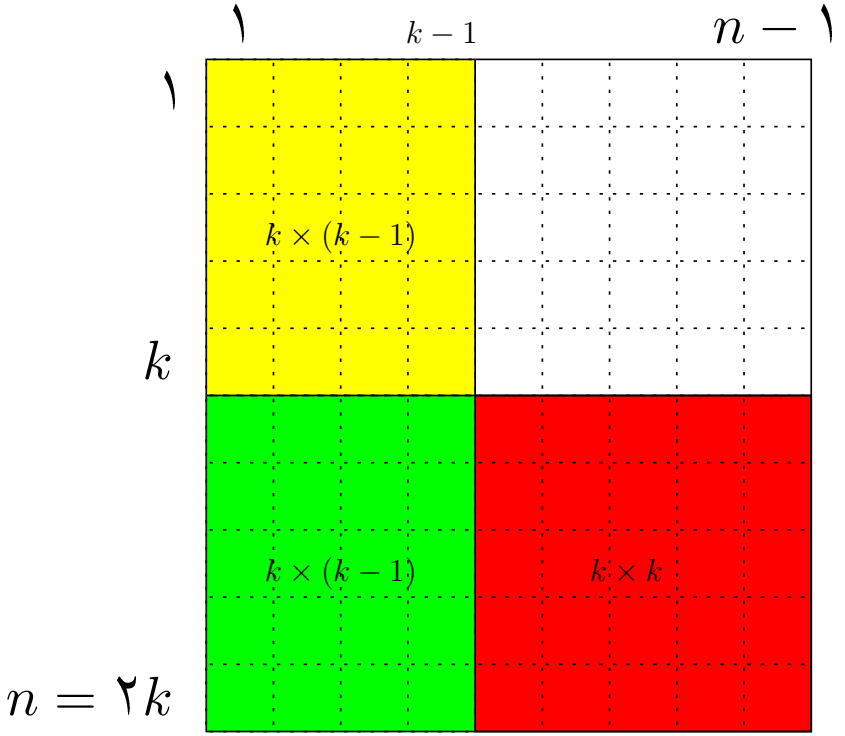
با استقرا اثبات می‌کنیم که:

۱) برای n زوج $n-۱$ روز لازم و کافی است.

۲) برای n فرد n روز لازم و کافی است.

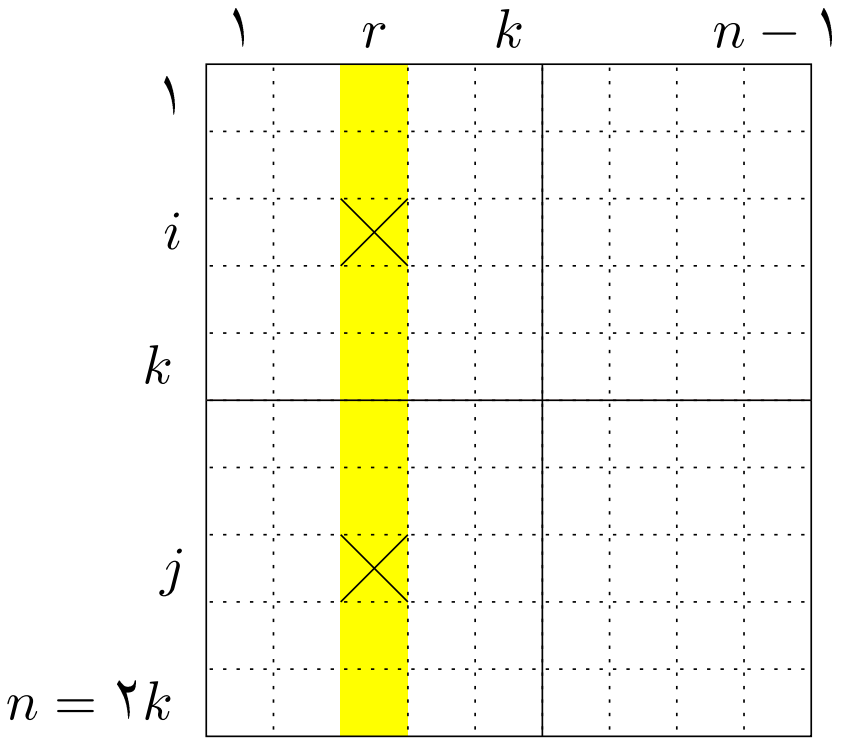
بند ۲ از ۱ نتیجه می‌شود.

$n = ۲k$ و k زوج

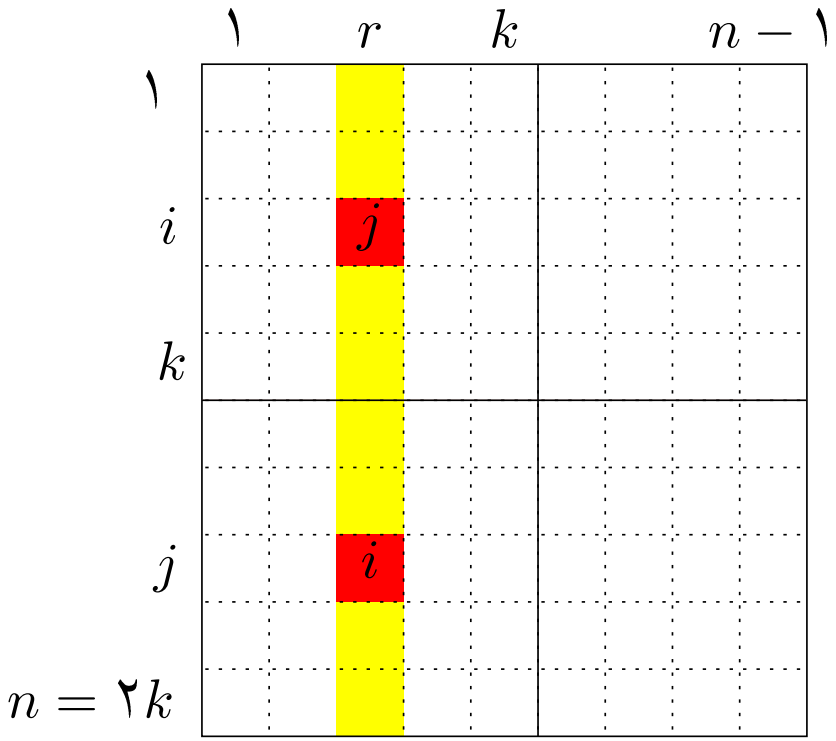


مربع لاتین

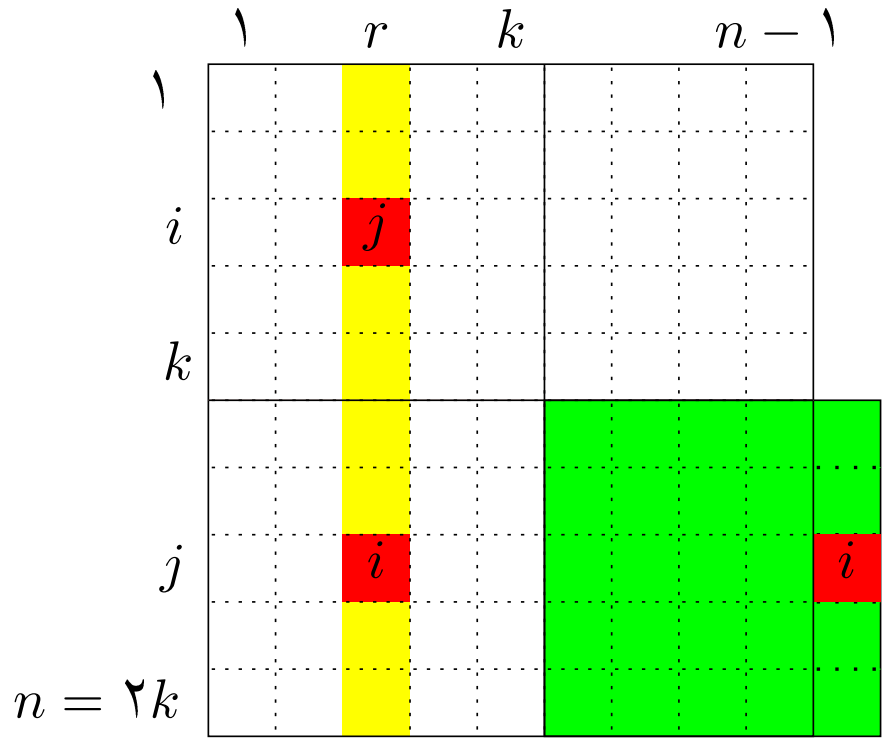
$n = ۲k$ و k فرد



$n = ۲k$ و k فرد

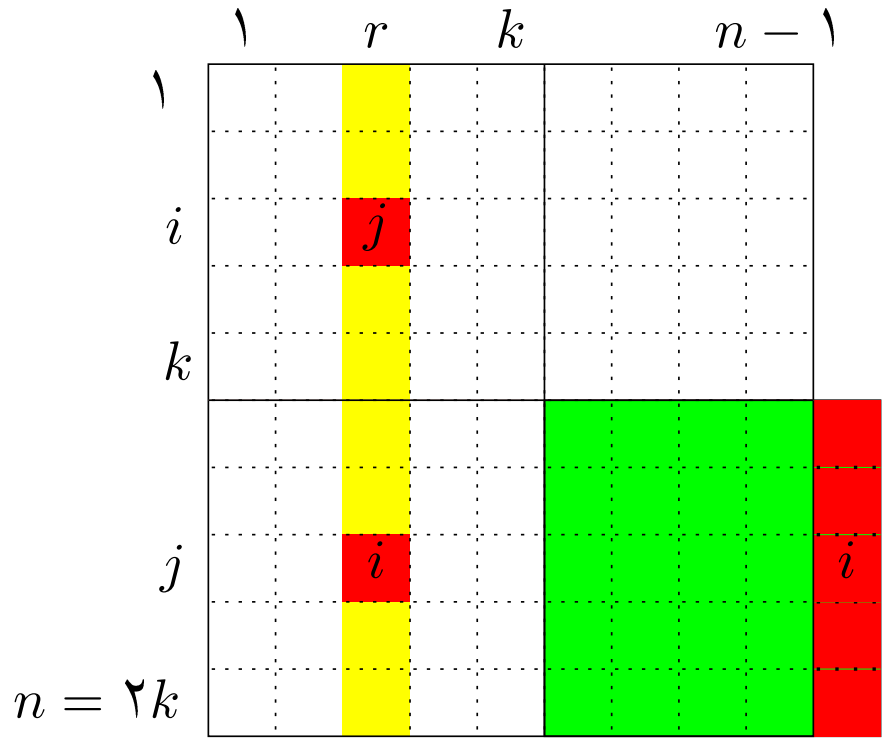


$n = ۲k$ و k فرد



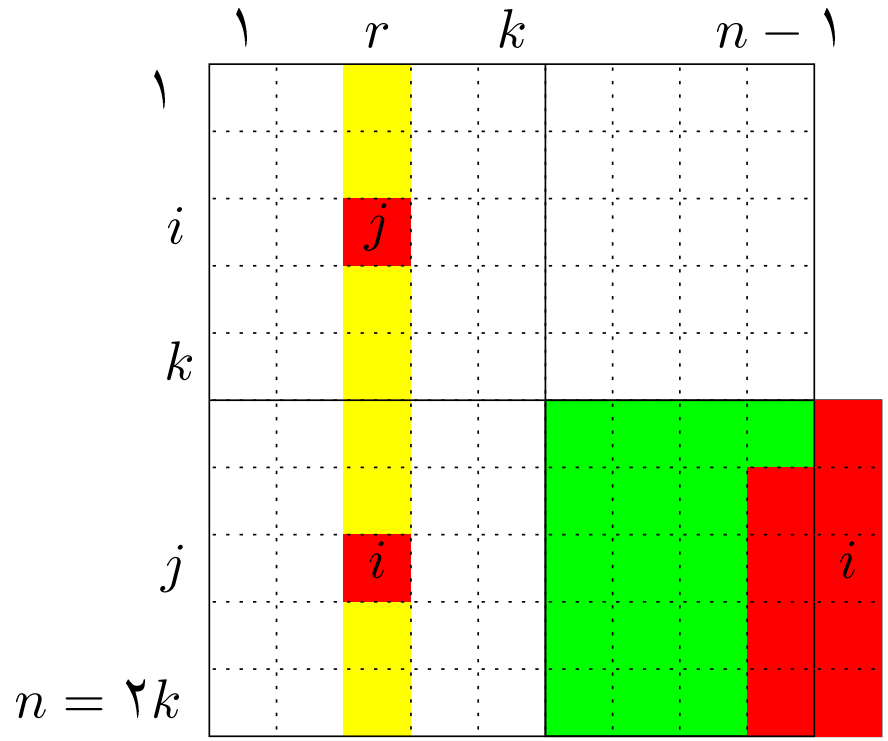
مربع لاتین

$n = ۲k$ و k فرد

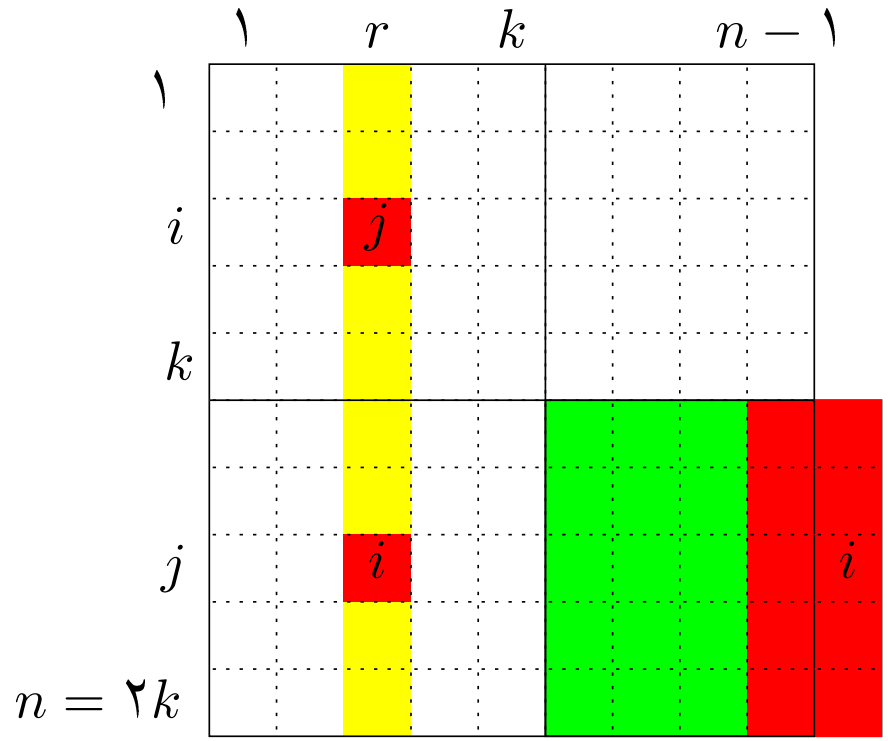


مربع لاتین

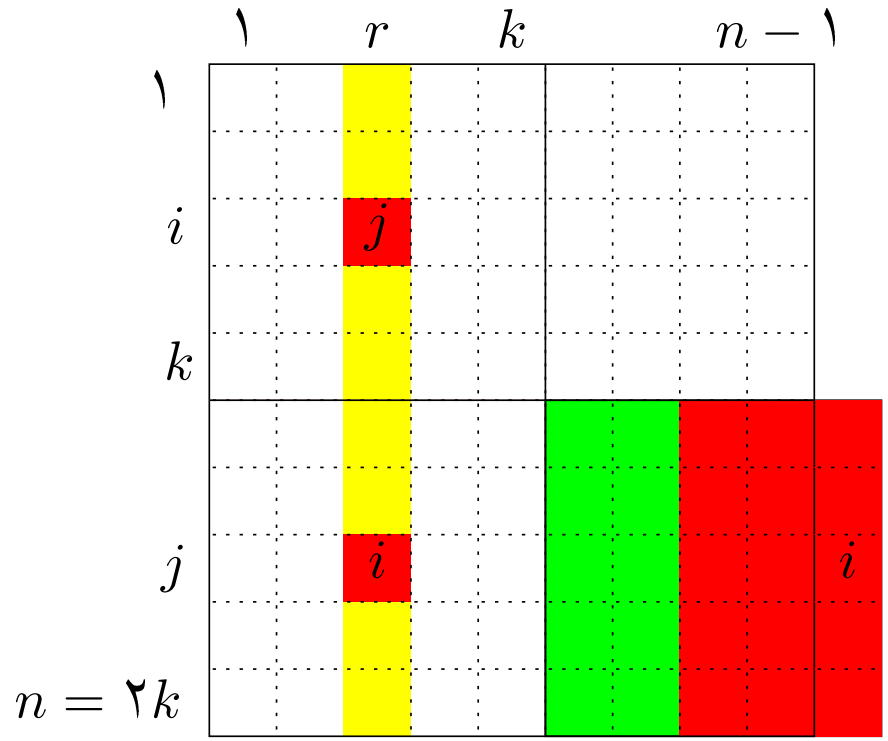
$n = ۲k$ و k فرد



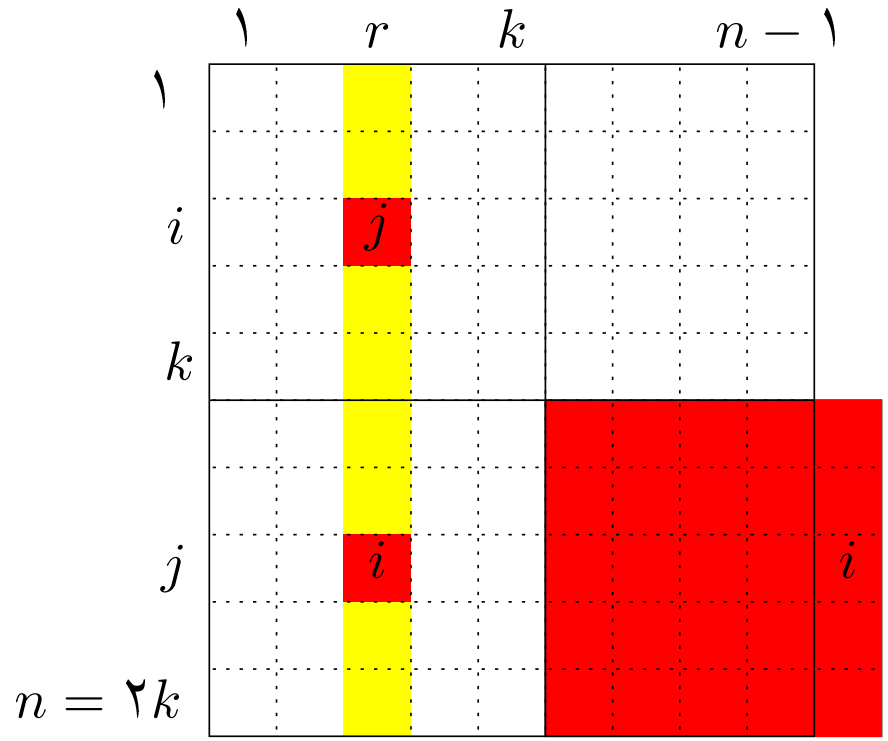
$n = ۲k$ و k فرد



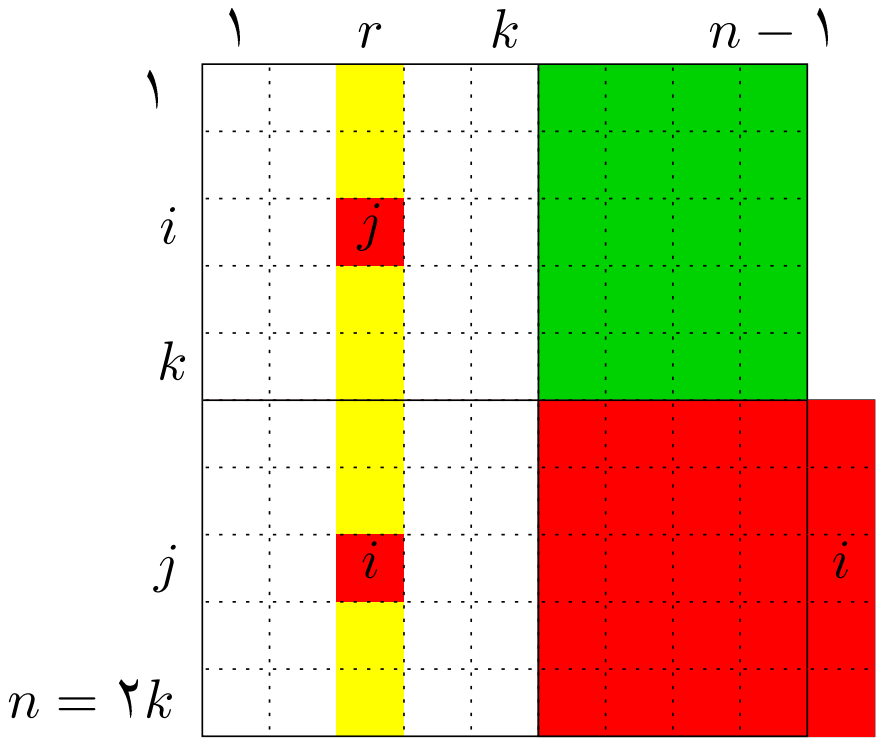
$n = ۲k$ و k فرد



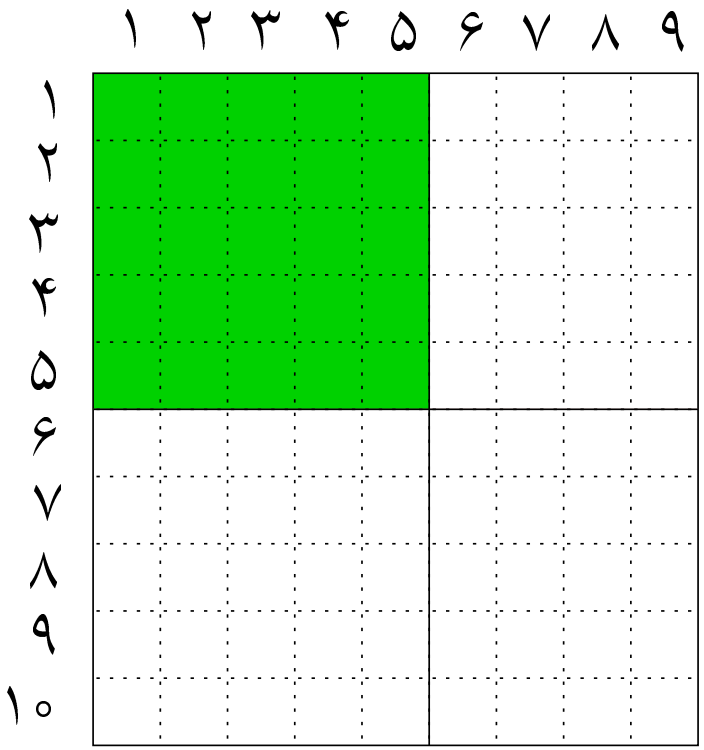
$n = ۲k$ و k فرد



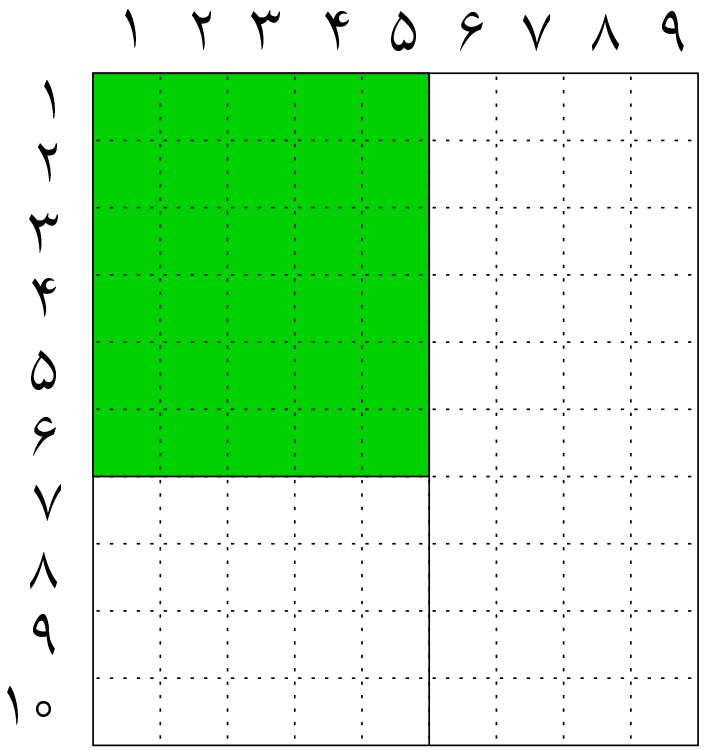
$n = ۲k$ و k فرد



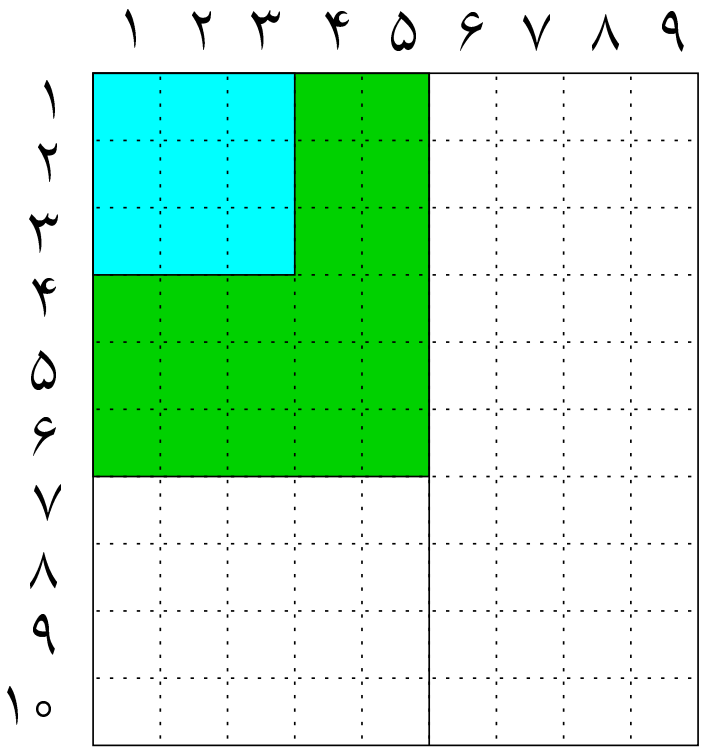
مثال برای $n = ۱۰$



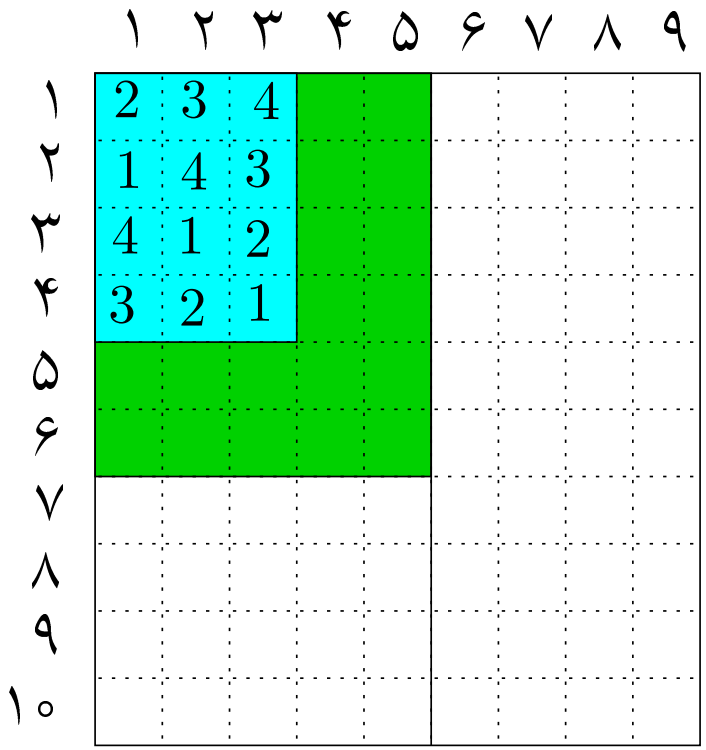
مثال برای $n = ۱۰$



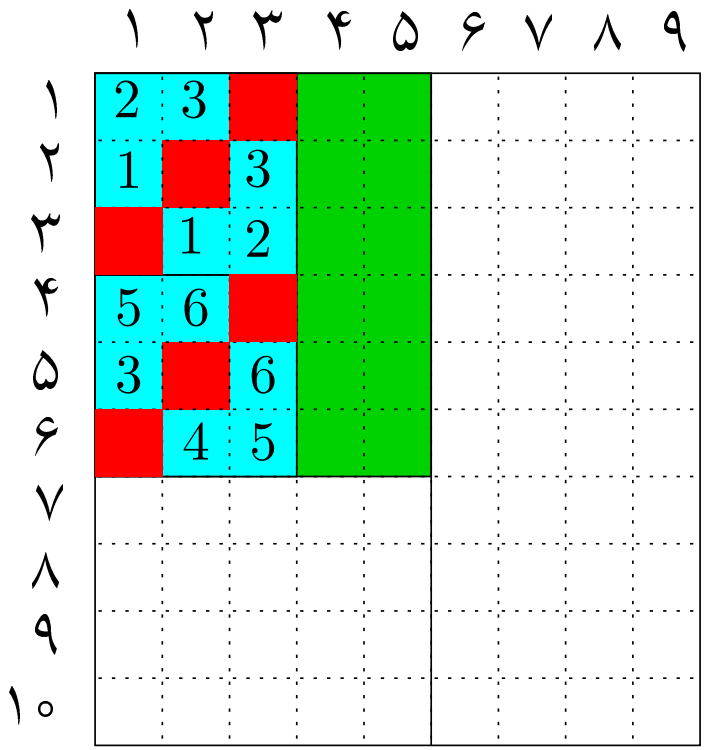
مثال برای $n = ۱۰$



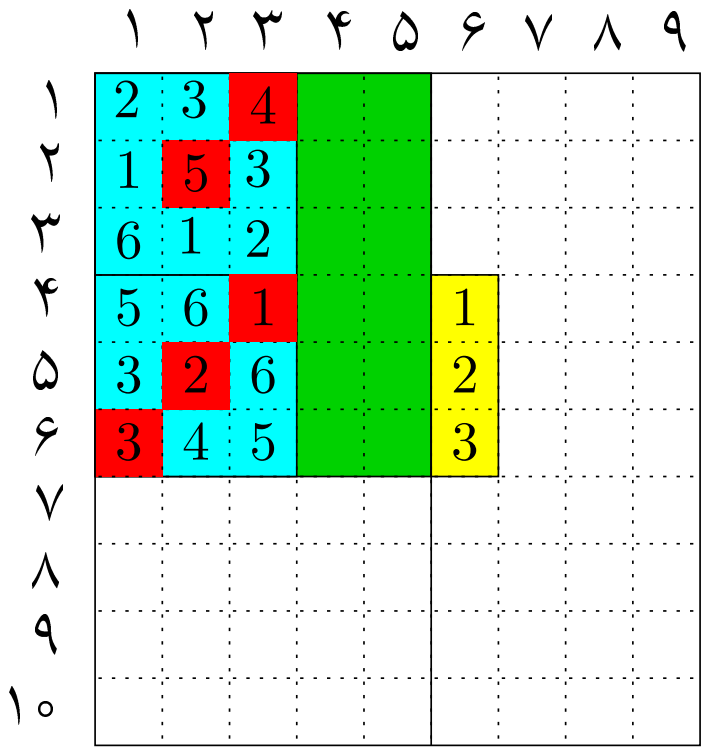
مثال برای $n = ۱۰$



مثال برای $n = ۱۰$



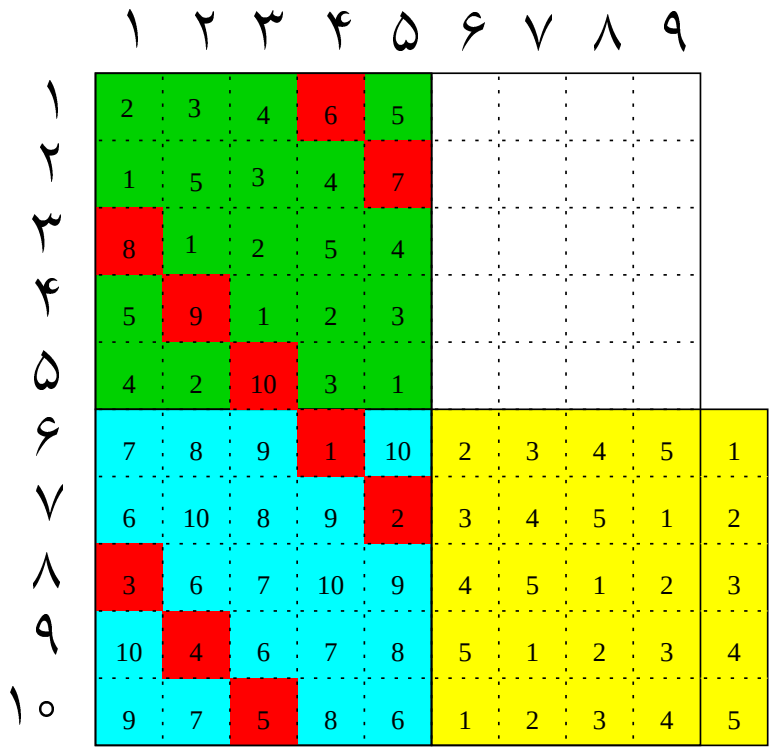
مثال برای $n = ۱۰$



مثال برای $n = ۱۰$

	۱	۲	۳	۴	۵	۶	۷	۸	۹
۱	2	3	4	6	5				
۲	1	5	3	4	6				
۳	6	1	2	5	4				
۴	5	6	1	2	3	1			
۵	4	2	6	3	1	2			
۶	3	4	5	1	2	3			
۷									
۸									
۹									
۱۰									

مثال برای $n = ۱۰$



مثال برای $n = ۱۰$

	۱	۲	۳	۴	۵	۶	۷	۸	۹
۱	2	3	4	6	5	10	9	8	7
۲	1	5	3	4	7	6	10	9	8
۳	8	1	2	5	4	7	6	10	9
۴	5	9	1	2	3	8	7	6	10
۵	4	2	10	3	1	9	8	7	6
۶	7	8	9	1	10	2	3	4	5
۷	6	10	8	9	2	3	4	5	1
۸	3	6	7	10	9	4	5	1	2
۹	10	4	6	7	8	5	1	2	3
۱۰	9	7	5	8	6	1	2	3	4

روش حریصانه

	1	2	3	4	5	6	7	8	9
1		2	3	4	5	6	7	8	9
2	9	1		3	4	5	6	7	8
3	8	9	1	2		4	5	6	7
4	7	8	9	1	2	3		5	6
5	6	7	8	9	1	2	3	4	
6	5		7	8	9	1	2	3	4
7	4	5	6		8	9	1	2	3
8	3	4	5	6	7		9	1	2
9	2	3	4	5	6	7	8		1

	1	2	3	4	5	6	7	8	9
1	10	2	3	4	5	6	7	8	9
2	9	1	10	3	4	5	6	7	8
3	8	9	1	2	10	4	5	6	7
4	7	8	9	1	2	3	10	5	6
5	6	7	8	9	1	2	3	4	10
6	5	10	7	8	9	1	2	3	4
7	4	5	6	10	8	9	1	2	3
8	3	4	5	6	7	10	9	1	2
9	2	3	4	5	6	7	8	10	1
10	1	6	2	7	3	8	4	9	5

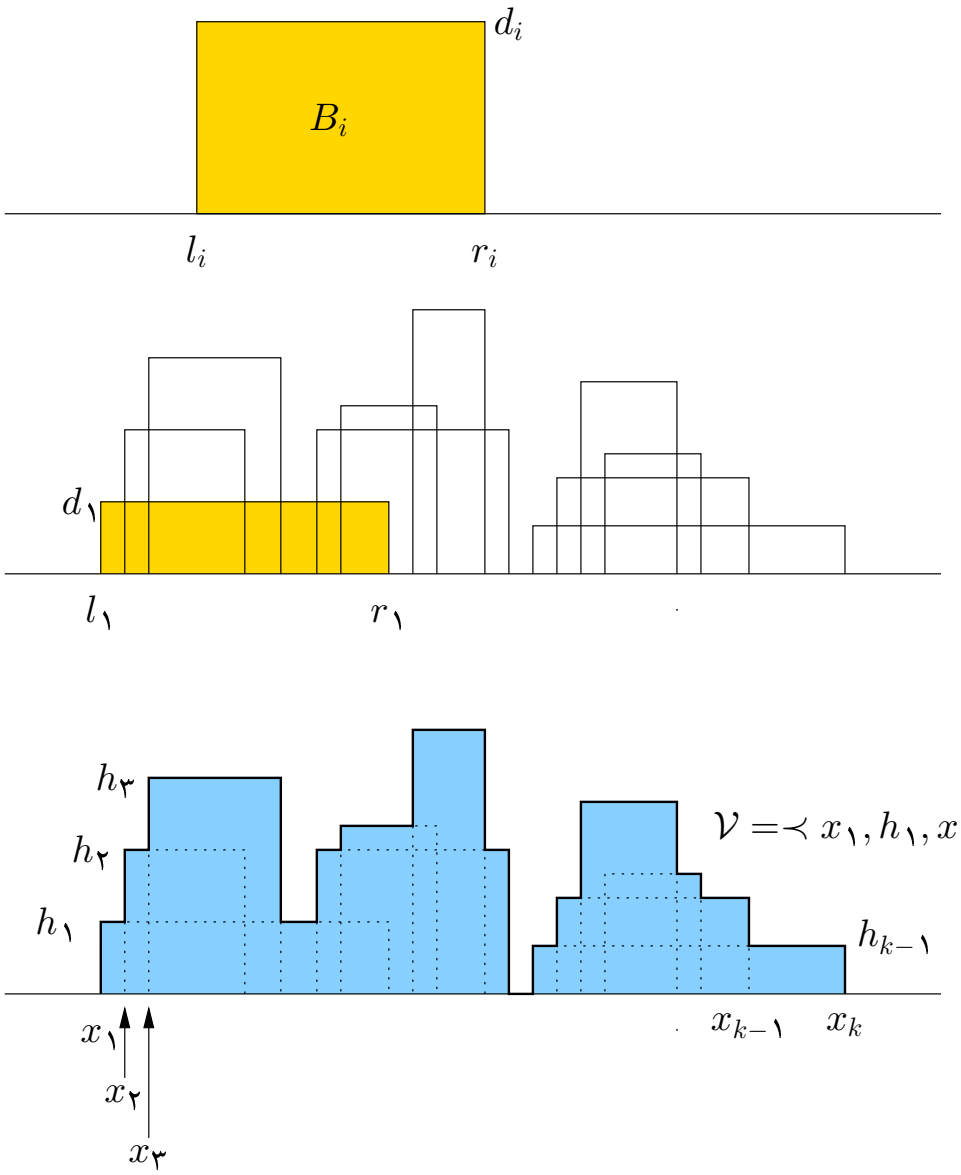
مسئله‌ی برج‌ها

n تا برج بر روی محور x

برج B_i با سه عدد $\langle l_i, r_i, d_i \rangle$

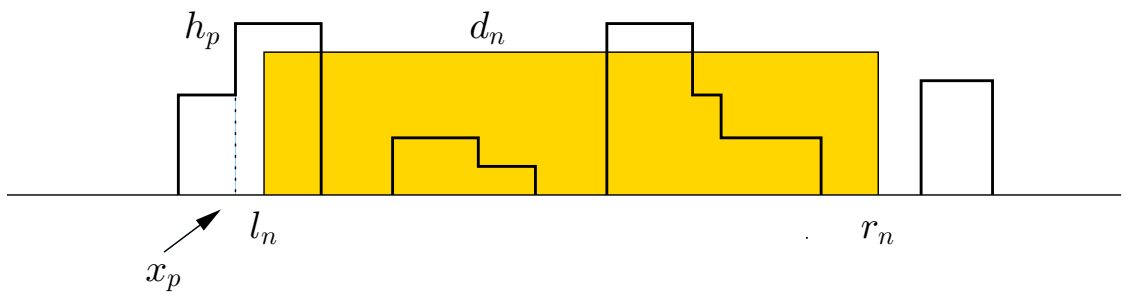
می‌خواهیم «نمای برج‌ها» را به‌دست بیاوریم.

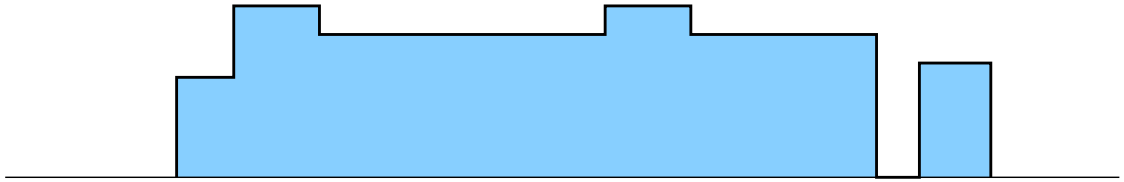
نما: $\mathcal{V} = \langle x_1, h_1, x_2, h_2, \ldots x_{k-1}, h_{k-1}, x_k, 0 \rangle$

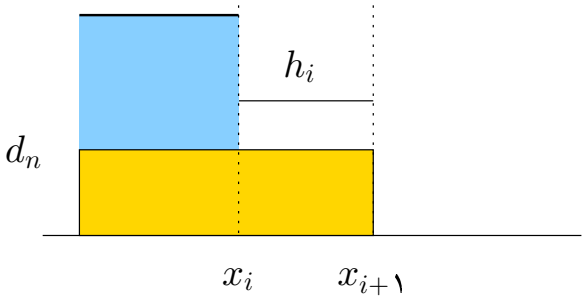

d_i
B_i
l_i
r_i
$d_۱$
$l_۱$
$r_۱$
$h_۳$
$h_۲$
$h_۱$
$h_{k-۱}$
$x_۱$
$x_۲$
$x_۳$
$x_{k-۱}$
x_k
$\mathcal{V} = \prec x_۱, h_۱, x_۲, h_۲, \ldots, x_{k-۱}, h_{k-۱}, x_k, 0 \succ$

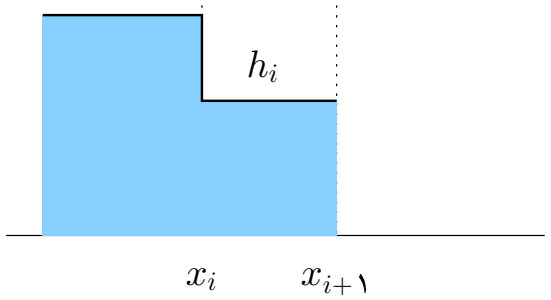
راه‌حل استقرایی

- برای $n = ۱$، نما همان برج
- نما را برای $n - ۱$ برج به‌صورت بازگشتی به‌دست می‌آوریم
- برج B_n را اضافه می‌کنیم و نمای جدید را به‌دست می‌آوریم.

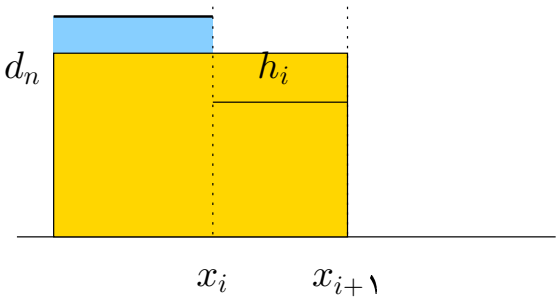

h_p
d_n
l_n
r_n
x_p

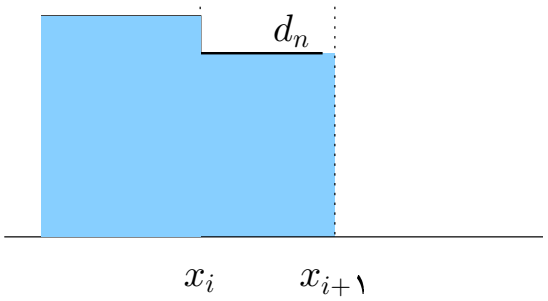




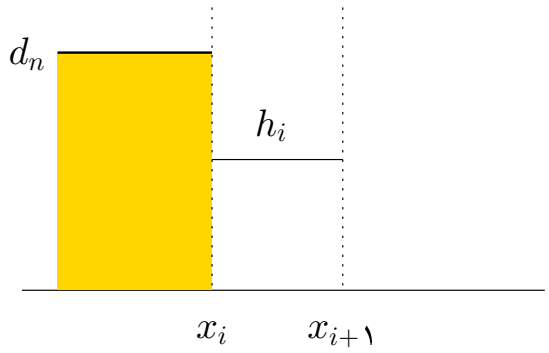


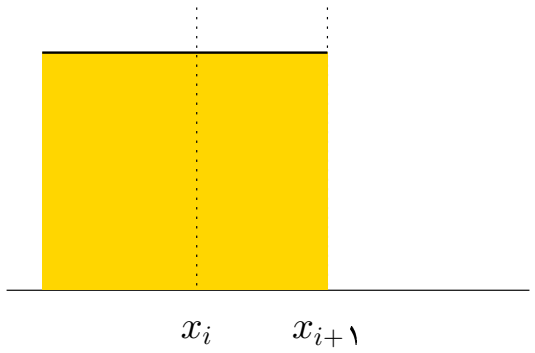
last-height $> d_n$, $h_i \geq d_n$



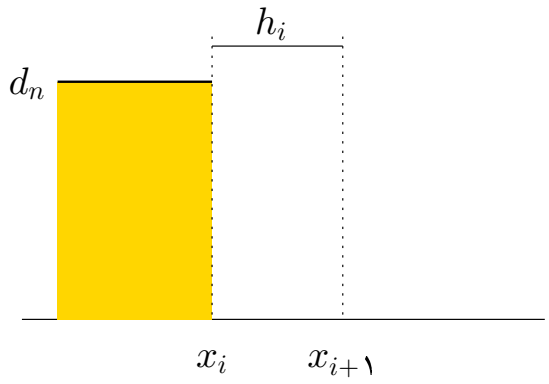


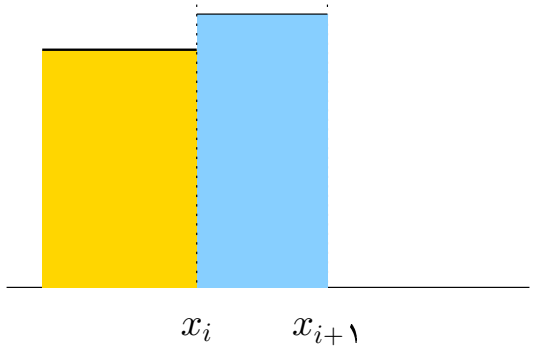
last-height $> d_n$, $h_i \leq d_n$





last-height $= d_n$, $h_i < d_n$





last-height $= d_n$, $h_i < d_n$

```
AddBorj (V_{n-1}, B_n)
    ▷ add B_n to view V_{i-1}
    ▷ Let V_{n-1} = ⟨x_1, h_1, x_2, h_2, ... x_{k-1}, h_{k-1}, x_k, 0⟩
    ▷ Let B_n = ⟨l_n, r_n, d_n⟩
 1  Find x_p < l_n < x_{p+1} using binary search
 2  V ← ⟨x_1, h_1, ..., x_p, h_p⟩
 3  if d_n > h_p
 4    then Add ⟨l_n, d_n⟩ to V;   last-height ← d_n
 5    else last-height ← h_p
 6  i ← p + 1 while x_i < r_n
 7          do if last-height > d_n and h_i ≥ d_n
 8               then Add ⟨x_i, d_i⟩ to V
 9                    last-height ← d_i
10             if last-height > d_n and h_i ≤ d_n
11               then Add ⟨x_i, d_n⟩ to V
12                    last-height ← d_n
13             if last-height = d_n and h_i > d_n
14               then Add ⟨x_i, d_i⟩ to V
15                    last-height ← d_i
16             i ← i + 1
17  if last-height = d_n
18    then Add ⟨r_n, h_{i-1}⟩ to V
19  return V
```

روش استقرایی: پیچیدگی

یافتن l_n: $\Theta(\log k)$

تصحیح ارتفاع‌ها می‌تواند حداکثر $\Theta(k)$ **باشد. چون** $k \leq n$ **داریم:**

$$T(n) = T(n-۱) + \Theta(n) \Longrightarrow T(n) = \Theta(n^۲)$$

روش تقسیم‌وحل

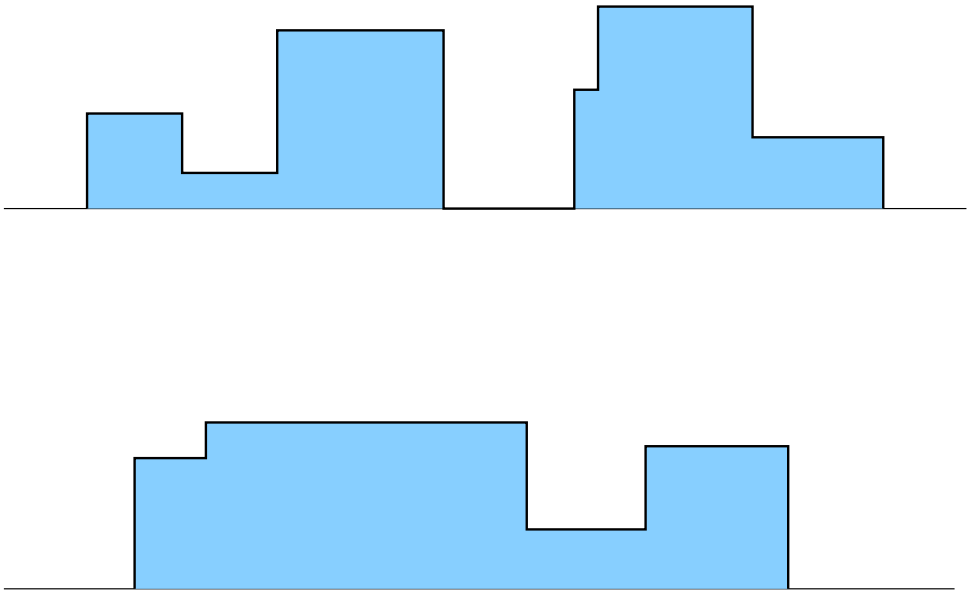
تقسیم مسئله‌ی n را به دو مسئله با اندازه‌های $\lceil n/۲ \rceil$ $\lfloor n/۲ \rfloor$

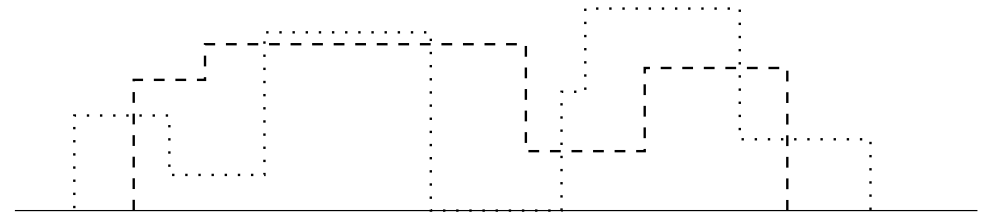
هر دو زیرمسئله را به همین روش حل می‌کنیم.

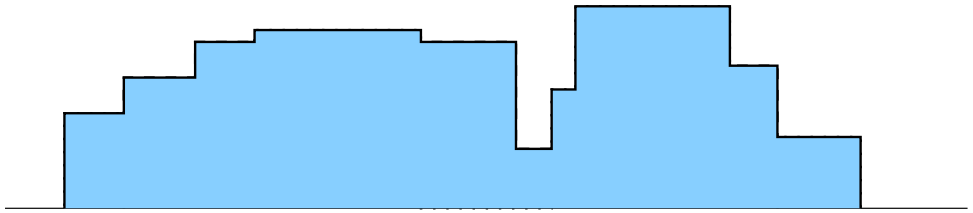
نمای اول: $V_۱ = \langle x_۱, h_۱, x_۲, h_۲, \ldots x_{k-۱}, h_{k-۱}, x_k, ۰ \rangle$

و نمای دوم: $V_۲ = \langle x_r, h_r, x_{r+۱}, h_{r+۱}, \ldots x_{p-۱}, h_{p-۱}, x_p, ۰ \rangle$ **باشد.**

ادغام $V_۱$ و $V_۲$ درست مانند عمل ادغام دو آرایه‌ی مرتب در MergeSort **است:**







پیچیدگی: $\Theta(\lceil \frac{n}{۲} \rceil + \lfloor \frac{n}{۲} \rfloor = n)$

$$T(n) = ۲T(\lceil \frac{n}{۲} \rceil)l + T(\lfloor \frac{n}{۲} \rfloor) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$$

کران پایین این مسئله $n \log n$ است. **چرا؟**

یافتن تعداد وارونگی‌های یک آرایه

وارونگی چیست؟

برای $i < j$ اگر $A[i] > A[j]$، زوج i و j یک وارونگی است.

چه فایده‌ای دارد؟

تعداد تعویض (هزینه‌ی) الگوریتم‌های ساده‌سازی که با تعویض عناصر مجاور مرتب می‌کنند (مانند مرتب‌سازی حبابی، درجی و انتخابی) برابر تعداد وارونگی‌های آرایه‌ی ورودی است.

روش تقسیم‌وحل

روش تقسیم‌وحل

```
COUNT-INV(A, n)
 1  if n = 1
 2    then return
 3  Split A into B and C with ⌊n/2⌋ and ⌈n/2⌉ elements resp.
 4  n1 ← COUNT-INV(B, ⌊n/2⌋)
 5  n2 ← COUNT-INV(C, ⌈n/2⌉)
 6  m ← COUNT-SPLIT-INV(B, ⌊n/2⌋, C, ⌈n/2⌉)
 7  return n1 + n2 + m
```

تحلیل؟

تحلیل الگوریتم تقسیم‌وحل

$\mathcal{O}(n^۲)$

$$T(n) = T(\lfloor n/۲ \rfloor) + T(\lceil n/۲ \rceil) + \mathcal{O}(n^۲)$$

چه کنیم؟

کران پایین=؟ $\mathcal{O}(n \lg n)$

راه‌حل بهینه

```
COUNT-INV-AND-SORT (A, n)
 1  if n = 1
 2    then return
 3  Split A into B and C with ⌊n/2⌋ and ⌈n/2⌉ elements resp.
 4  n_1 ← COUNT-INV-AND-SORT(B, ⌊n/2⌋)
 5  n_2 ← COUNT-INV-AND-SORT(C, ⌈n/2⌉)
 6  m ← COUNT-SPLIT-INV-SORTED(B, ⌊n/2⌋, C, ⌈n/2⌉)
 7  return n_1 + n_2 + m
```

بررسی مجدد ادغام دو آرایه‌ی مرتب

```
MERGE(A, B)
 1  A[n+1] ← ∞;   B[m+1] ← ∞
 2  i ← 1;   j ← 1
 3  for k ← 1 to n+m
 4      do if A[i] < B[j]
 5            then C[k] ← A[i]
 6                 i ← i+1
 7            else C[k] ← B[j]
 8                 j ← j+1
 9  length[C] ← n+m
10  return C
```

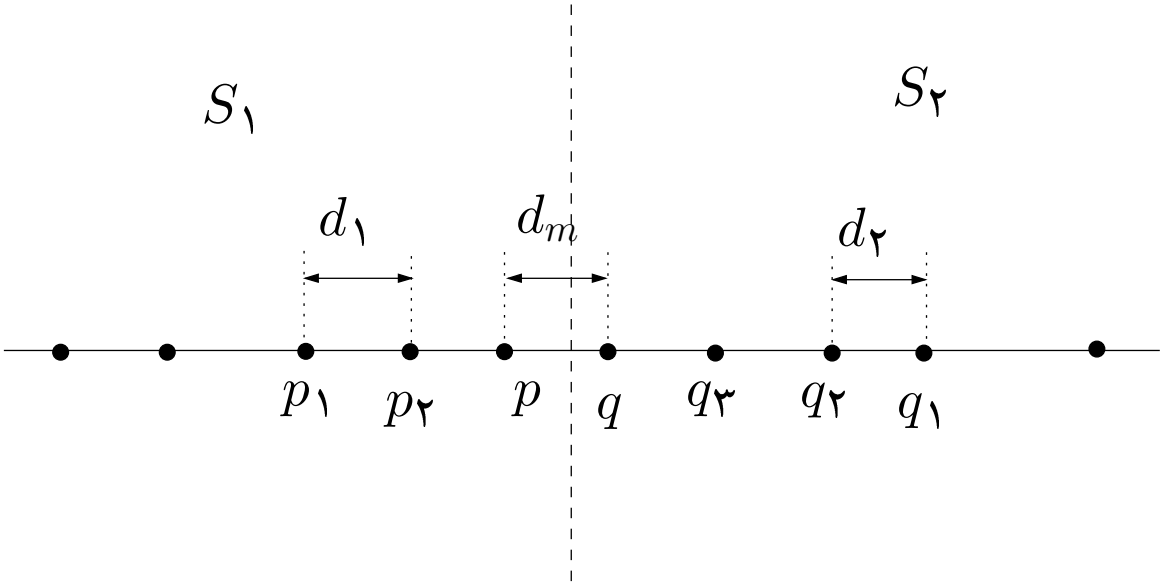
یافتن Split-Inversions

تحلیل نهایی

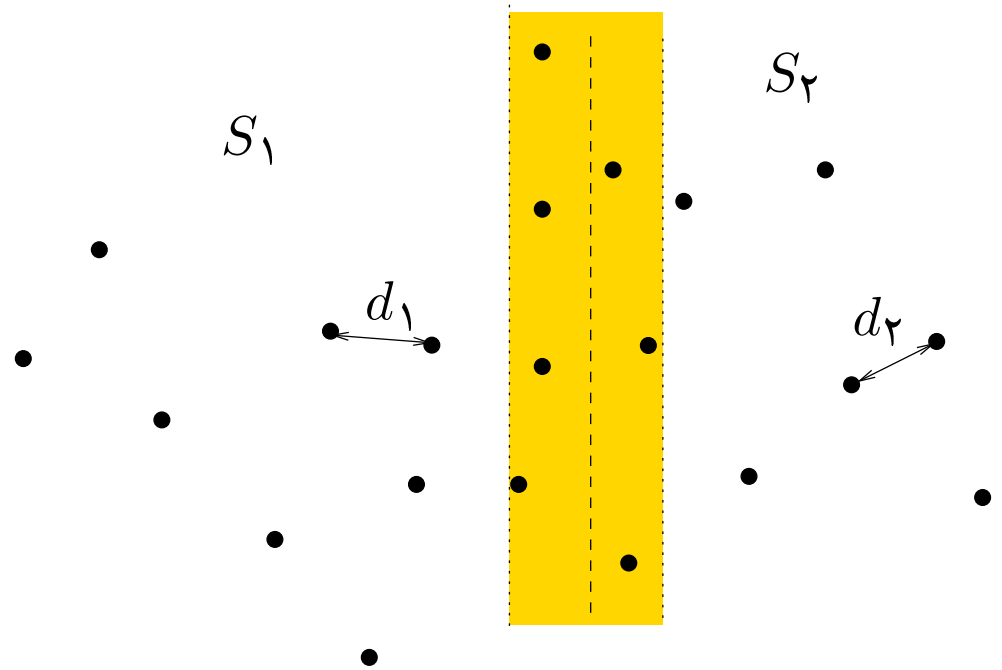
$$T(n) = T(\lfloor n/۲ \rfloor) + T(\lceil n/۲ \rceil) + \mathcal{O}(n) = \mathcal{O}(n \lg n)$$

نزدیک‌ترین زوج نقاط

یک بعدی



```
CLOSET-PAIR-1D (S)
 1  if |S| = 2
 2    then return d = |p_1 − p_2|
 3    else  Sort the points
 4          divide S from the mid-point and Construct S_1 and S_2
 5          d_1 ← CLOSET-PAIR-1D(S_1)
 6          d_2 ← CLOSET-PAIR-1D(S_2)
 7          p ← max(S_1)
 8          q ← min(S_2)
 9          d ← min {d_1, d_2, q − p}
```


$S_۱$
$S_۲$
$d_۱$
$d_۲$

۱) نقاط را هم برحسب x و هم برحسب y شان مرتب کن.

۲) $l_{\bar{x}}$، میانه‌ی x نقاط را به‌دست آوده و بر آن اساس نقاط را به دو مجموعه‌یی S_1 و S_2 افراز کن.

۳) به‌طور بازگشتی نزدیک‌ترین زوج نقطه‌ی (p_1, q_1) از S_1 با فاصله‌ی d_1 را به‌دست آور.

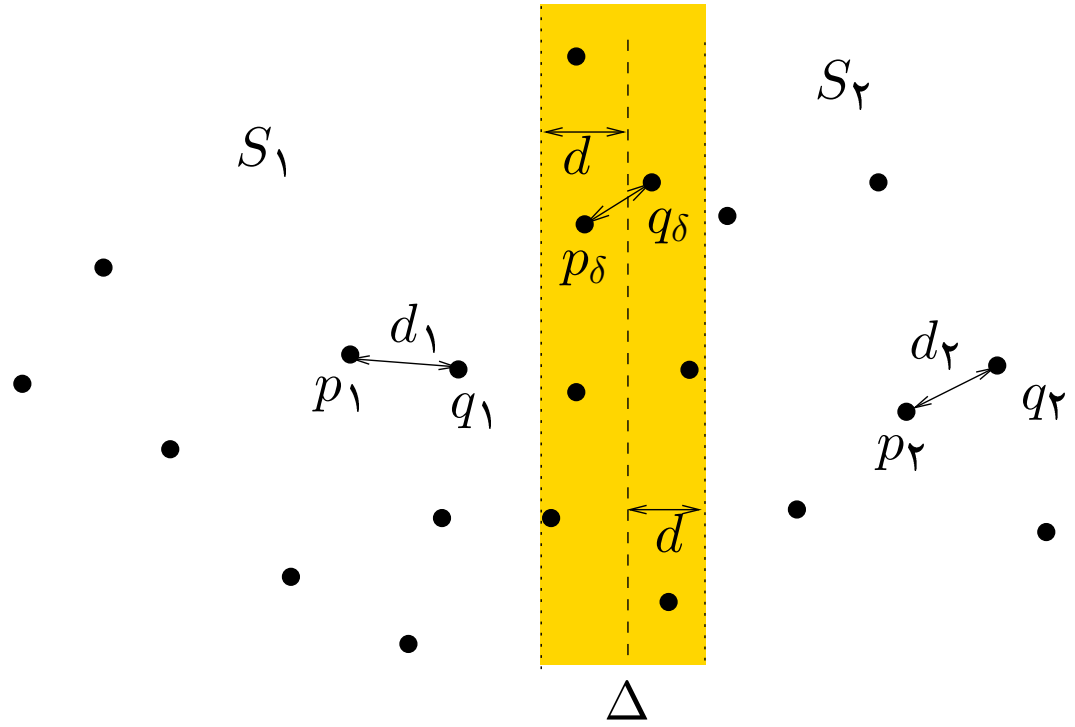
۴) نزدیک‌ترین زوج نقطه‌ی (p_2, q_2) از S_2 با فاصله‌ی d_2 را هم به‌دست آور.

۵) $d = \min\{d_1, d_2\}$

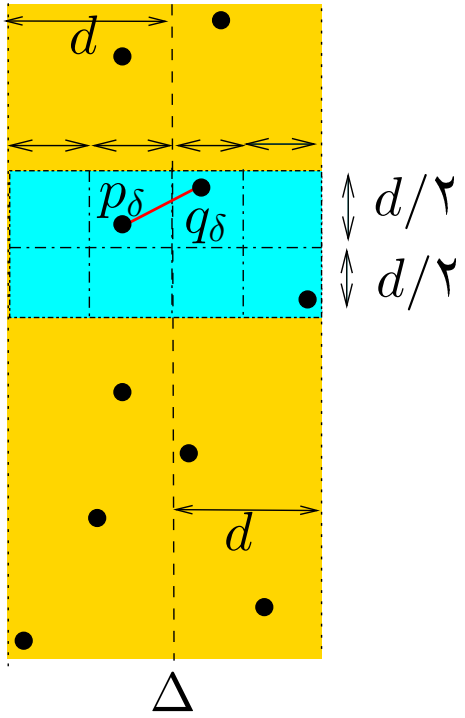
۶) نوار Δ از نقاط با فاصله‌ی d از چپ و راست خط $l_{\bar{x}}$ رامحاسبه کن.

۷) فرض کنید (p_δ, q_δ) نزدیک‌ترین زوج نقطه در Δ که $p_\delta \in S_1$ و $q_\delta \in S_2$ باشد

۸) پاسخ نزدیک‌ترین زوج‌نقطه‌ی بین (p_1, q_1)، (p_2, q_2) و (p_δ, q_δ) است.



$$d = \min\{d_۱, d_۲\}$$


d
p_δ
q_δ
$d/۲$
$d/۲$
d
Δ

برای به‌دست آوردن (p_δ, q_δ) کافی است

۱) نقاط موجود در Δ (که ممکن است همه‌ی نقاط باشد) به صورت مرتب برحسب y به‌دست آور.

۲) این نقاط را به‌ترتیب نزولی مورد بررسی قرار بده.

۳) کافی است هر نقطه را با ۷ نقطه‌ی بعدی دنباله‌ی مرتب چک کنیم.

تحلیل

- $\mathcal{O}(n \lg n)$ برای مرتب‌سازی اولیه
- $T(n) = ۲T(n/۲) + \mathcal{O}(n) = \mathcal{O}(n \lg n)$

این بهینه است.

ضرب دو چندجمله‌ای

دو چندجمله‌ای P و Q برحسب x و از درجه‌ی n

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0$$

$$Q_n(x) = b_n x^n + b_{n-1} x^{n-1} + \cdots + b_0$$

می خواهیم با یک الگوریتم کارا حاصل‌ضرب

$$R_{2n}(x) = P_n(x) Q_n(x) = \sum_{i=0}^{2n} c_i x^i$$

را محاسبه کنیم.

پیاده‌سازی

ورودی: $n, a_n, a_{n-1}, \ldots, a_0, b_n, \ldots, b_1, b_0$

خروجی: $c_{2n}, c_{2n-1}, \ldots, c_0$

ورودی را دو آرایه‌های $n+1$ عنصری A و B و خروجی را در آرایه‌ی $2n+1$ عنصری C می‌نویسیم

راه حل ۱

برای $0 \le k \le 2n$ **داریم،** $c_k = \sum_{i+j=k} a_i b_j$.

cها را می‌توان با $\Theta(n^2)$ **به‌دست آورد.**

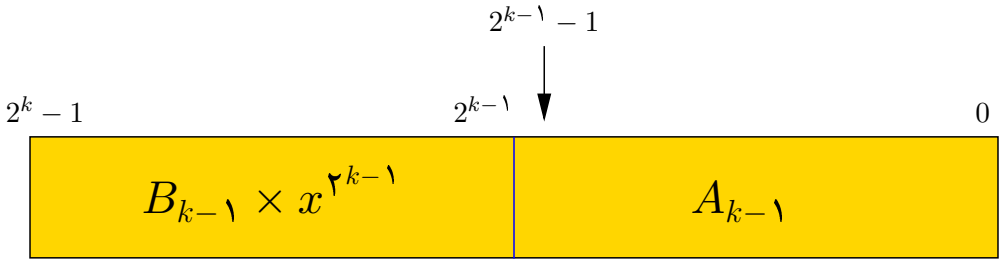
```
MultPoly(A, B, n)
 1  for i ← 0 to 2n
 2      do C[i] ← 0
 3  for i ← 0 to n
 4      do for j ← 0 to n
 5             do C[i + j] ← C[i + j] + A[i] * B[j]
 6  return C
```

راه‌حل ۲: تقسیم‌وحل

برای راحتی فرض می‌کنیم $n+۱ = ۲^k$

در ابتدا P_n و Q_n را به دو نیمه تقسیم می‌کنیم:

$$P_n(x) \equiv AB_k = B_{k-۱}x^{۲^{k-۱}} + A_{k-۱}$$



$$P_n = AB_k$$

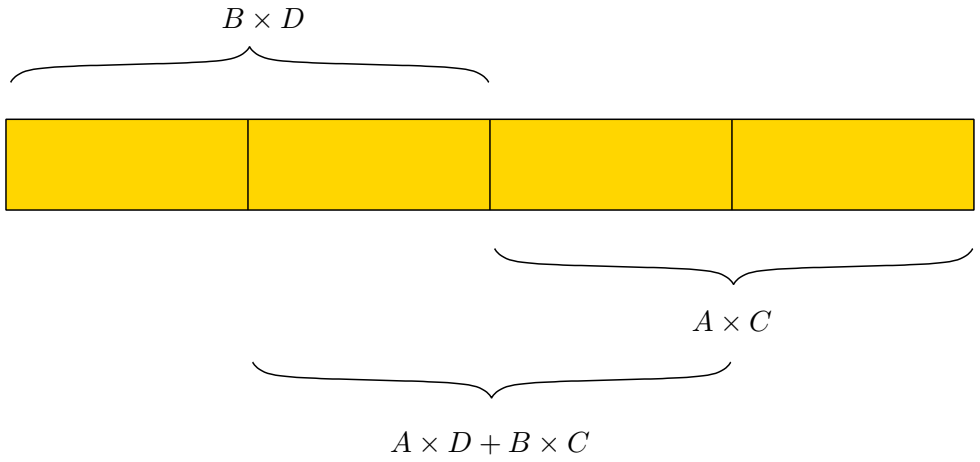
همچنین

$$Q_n(x) = D_{k-1}x^{2^{k-1}} + C_{k-1}$$

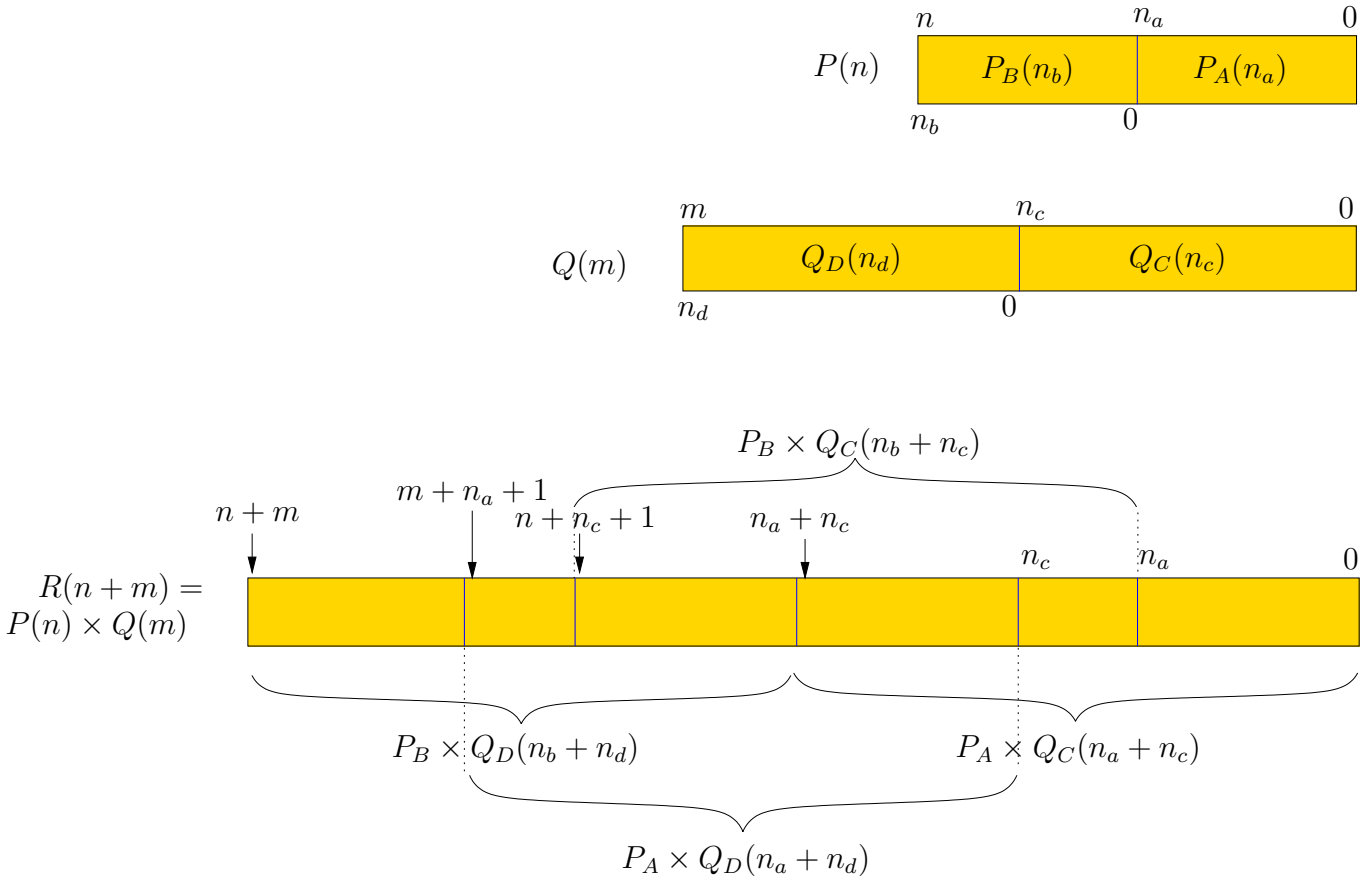
A، B، C و D چندجمله‌ای‌هایی از درجه‌ی $2^{k-1} - 1 = \frac{n+1}{2}$ هستند.

حاصل‌ضرب $P \times Q$ چگونه محاسبه می‌شود؟

$$P \times Q = (B \times D) \times x^{۲k} + (A \times D + B \times C)x^{۲k-۱} + A \times C$$



حالت کلی


$P(n)$
n
n_a
0
$P_B(n_b)$
$P_A(n_a)$
n_b
0
$Q(m)$
m
n_c
0
$Q_D(n_d)$
$Q_C(n_c)$
n_d
0
$P_B \times Q_C(n_b + n_c)$
$n + m$
$m + n_a + 1$
$n + n_c + 1$
$n_a + n_c$
n_c
n_a
0
$R(n + m) =$
$P(n) \times Q(m)$
$P_B \times Q_D(n_b + n_d)$
$P_A \times Q_C(n_a + n_c)$
$P_A \times Q_D(n_a + n_d)$

تحلیل:

$$T(n) = ۴T(\frac{n+۱}{۲}) + \Theta(n) \Longrightarrow T(n) = \Theta(n^۲)$$

راه‌حل ۳: تقسیم‌وحل

$$A \times D + B \times C = (A - B) \times (D - C) + A \times C + B \times D$$

$$T(n) = ۳T(\frac{n + ۱}{۲}) + \Theta(n) \Longrightarrow T(n) = \Theta(n^{\log_۲ ۳})$$

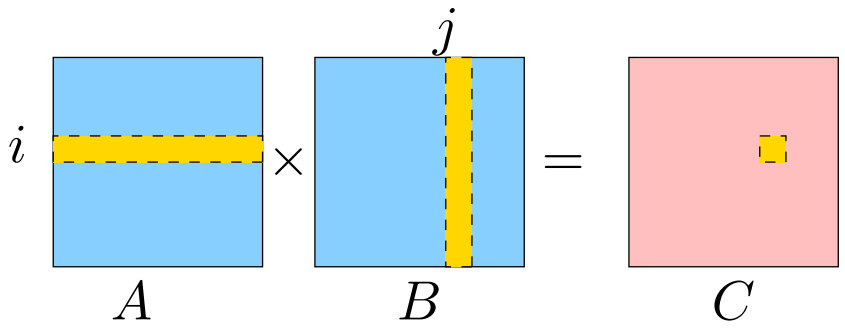
راه‌حل بهینه

برای این مسئله راه‌حل بهینه‌ی $O(n \lg n)$ وجود دارد. این راه‌حل بر اساس روش FFT (تابع تبدیل سریع فوریه) است.

مسئله‌ی دیگر: الگوریتم استراسون برای ضرب ماتریس‌ها

ضرب دو ماتریس A **و** B $(A \times B)$ **به ابعاد** $n \times n$

روش معمول از $O(n^۳)$



$$C[i,j] = \sum_{k=۱}^{n} A[i,k]B[k,j]$$

آقای استراسون[1] در سال ۱۹۶۹ با استفاده از روش تقسیم‌وحل موفق شد این‌کار را در $\Theta(n^{\lg ۷}) = O(n^{۲.۸۱})$ انجام دهد.

ماتریس‌ها را به ماتریس‌هایی به اندازه‌ی نصف تقسیم می‌کنیم.

$$\begin{bmatrix} A_{۱۱} & A_{۱۲} \\ A_{۲۱} & A_{۲۲} \end{bmatrix} \times \begin{bmatrix} B_{۱۱} & B_{۱۲} \\ B_{۲۱} & B_{۲۲} \end{bmatrix} = \begin{bmatrix} C_{۱۱} & C_{۱۲} \\ C_{۲۱} & C_{۲۲} \end{bmatrix}$$

رابطه‌های زیر در مورد این ۱۲ ماتریس کوچک‌تر برقرارند:

$$\begin{aligned}
C_{11} &= A_{11}B_{11} + A_{12}B_{21},\\
C_{12} &= A_{11}B_{12} + A_{12}B_{22},\\
C_{21} &= A_{21}B_{11} + A_{22}B_{21},\\
C_{22} &= A_{21}B_{12} + A_{22}B_{22}.
\end{aligned}$$

راه‌حل اول: **۸ ضرب ماتریس‌های کوچک‌تر**

$$\Theta(n^3) \Longleftarrow \quad T(n) = 8T(n/2) + \Theta(n^2)$$

آقای استرسن ۸ ضرب را به ۷ ضرب تقلیل داد.

$$\Theta(n^{\lg 7}) = O(n^{2.81}) \Longleftarrow \quad T(n) = 7T(n/2) + \Theta(n^2)$$

لم. ضرب دو ماتریس دل‌خواه ۲×۲ را می‌توان با ۷ عمل ضرب و ۱۸ عمل جمع و تفریق انجام داد.

اثبات.

$$\begin{bmatrix} c_{۱۱} & c_{۱۲} \\ c_{۲۱} & c_{۲۲} \end{bmatrix} = \begin{bmatrix} a_{۱۱} & a_{۱۲} \\ a_{۲۱} & a_{۲۲} \end{bmatrix} \times \begin{bmatrix} b_{۱۱} & b_{۱۲} \\ b_{۲۱} & b_{۲۲} \end{bmatrix}$$

و حاصل‌ضرب‌های زیر را به‌دست می‌آوریم،

$$\begin{aligned} m_{۱} &= (a_{۱۱} - a_{۱۲})(b_{۲۱} + b_{۲۲}), \\ m_{۲} &= (a_{۱۱} + a_{۲۲})(b_{۱۱} + b_{۲۲}), \\ m_{۳} &= (a_{۱۱} - a_{۲۱})(b_{۱۱} + b_{۱۲}), \end{aligned}$$

$$
\begin{aligned}
m_۴ &= (a_{۱۱} + a_{۱۲})b_{۲۲}, \\
m_۵ &= a_{۱۱}(b_{۱۲} - b_{۲۲}), \\
m_۶ &= a_{۲۲}(b_{۲۱} - b_{۱۱}), \\
m_۷ &= (a_{۲۱} + a_{۲۲})(b_{۱۱}.
\end{aligned}
$$

حال، می‌توان دید که اعداد ماتریس حاصل به‌صورت زیر قابل محاسبه‌اند،

$$
\begin{aligned}
c_{۱۱} &= m_۱ + m_۲ + m_۴ + m_۶, \\
c_{۱۲} &= m_۴ + m_۵, \\
c_{۲۱} &= m_۶ + m_۷, \\
c_{۲۲} &= m_۲ - m_۳ + m_۵ - m_۷.
\end{aligned}
$$

که ۷ عمل ضرب و ۱۸ عمل جمع و تفریق دارد.

قضیه. ضرب دو ماتریس $n \times n$ را می‌توان در $\Theta(n^{\lg ۷})$ انجام داد.

بهترین الگوریتم برای ضرب ماتریس‌ها

بهترین الگوریتم موجود برای ضرب دو ماتریس $n \times n$ از مرتبه‌ی $O(n^{2.376})$ است که توسط کاپراسمیت و وینوگراد در سال ۱۹۸۷ ارایه شد. البته این نتیجه صرفاً از بعد نظری قابل توجه است و هنوز کاربرد عملی ندارد.

هم‌چنین روشن نیست که آیا به‌تر از این الگوریتم وجود دارد یا خیر. البته بدیهی که حد پایین این الگوریتم $\Theta(n^2)$ است.