طراحی الگوریتم

درس سوم: روش تقسیم و حل

مدرس:
فرشید شیرافکن

دانشجوی **دکتری دانشگاه تهران**
(کارشناسی و کارشناسی ارشد : کامپیوتر نرم افزار) (دکتری: بیو انفورماتیک)

روش تقسیم و حل

(Divide-and-Conquer)

مراحل :

۱- تقسیم نمونه ای از یک مسئله به یک یا چند نمونه کوچکتر

۲- حل نمونه های کوچکتر

۳- ترکیب حل نمونه های کوچکتر برای بدست آوردن حل نمونه اولیه (در صورت نیاز)

دلیل اینکه می گوئیم "در صورت نیاز" این است که در بعضی الگوریتم ها مانند جستجوی دودویی نمونه فقط به یک نمونه کوچکتر کاهش می یابد و نیازی به ترکیب حل ها نیست.

هنگام طراحی الگوریتم های تقسیم و حل معمولاً آن را به صورت یک روال بازگشتی می نویسند.

در صورت امکان باید در مورد زیر از روش تقسیم و حل پرهیز کرد:

نمونه ای با اندازه n به دو یا چند نمونه تقسیم می شود که اندازه آن ها نیز تقریبا n است. افراز در این حالت به یک الگوریتم زمانی نمایی منجر می شود.

مثال

الگوریتمی هر ورودی مسئله به اندازه ی **n** را به ۲ بخش کم و بیش مساوی تقسیم می کند. زیر مسئله ها را به صورت بازگشتی حل و سپس با هزینه خطی حاصل این دو را با هم ترکیب کرده و جواب مسئله را به دست می آورد.

رابطه بازگشتی :

$$T(n) = 2T(\frac{n}{2}) + O(n)$$
$$\Rightarrow T(n) = O(n\lg n)$$

جستجوی دودویی

پیدا کردن عدد ۱۰ در آرایه مرتب:

1	2	3	4	5	6	7	8	9
5	9	10	20	35	50	60	70	75

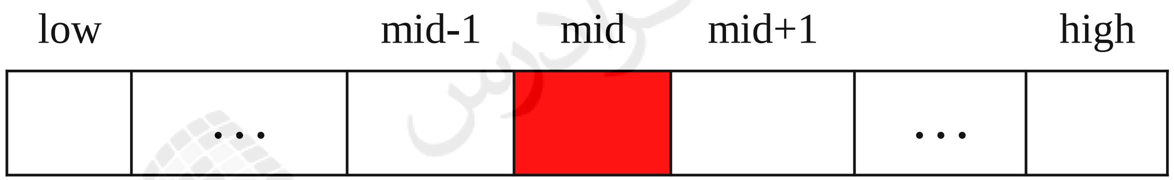
5	9	10	20	35	50	60	70	75

5	9	10	20					

		10	20					

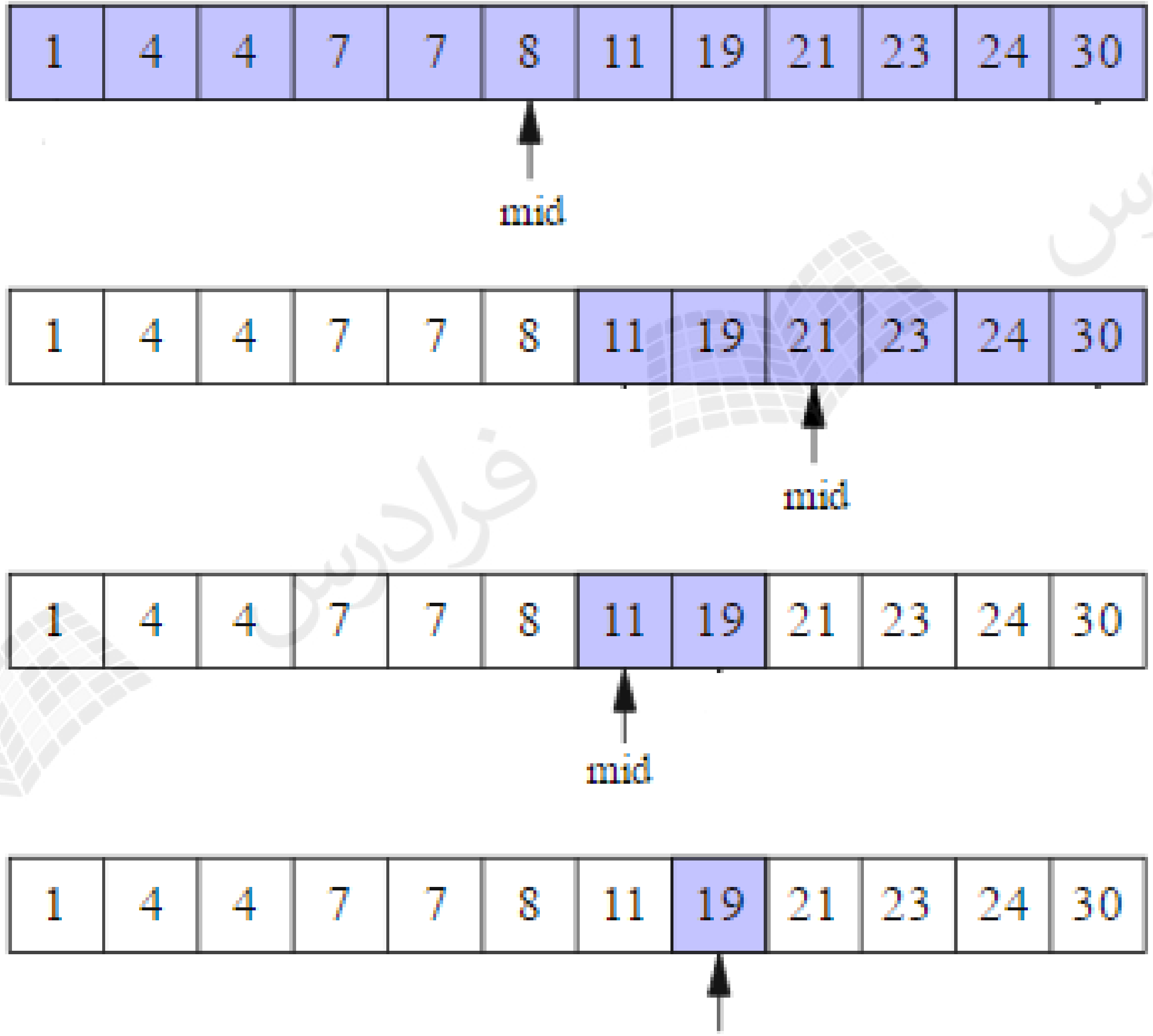
الگوریتم جستجوی دودویی

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

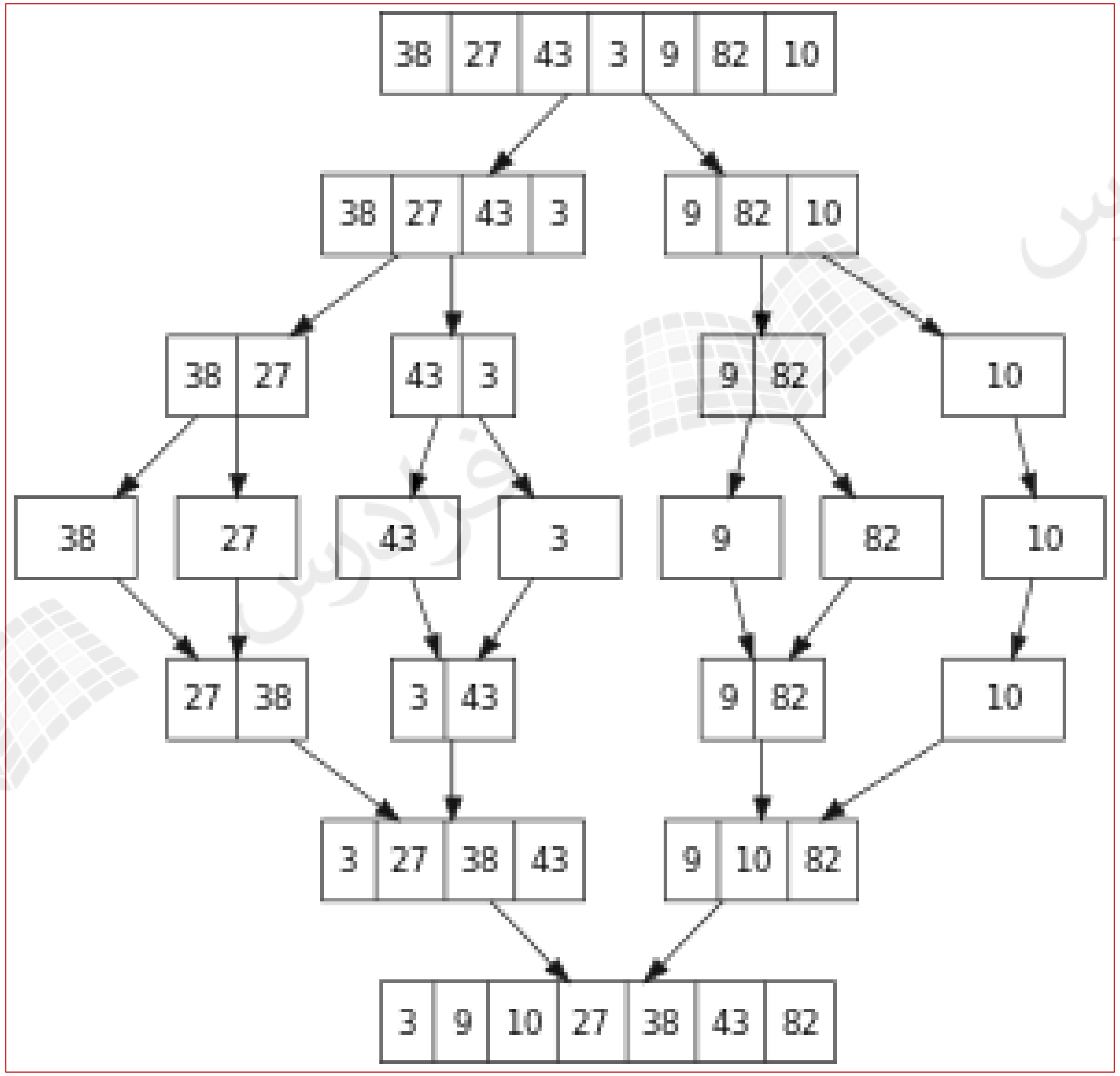


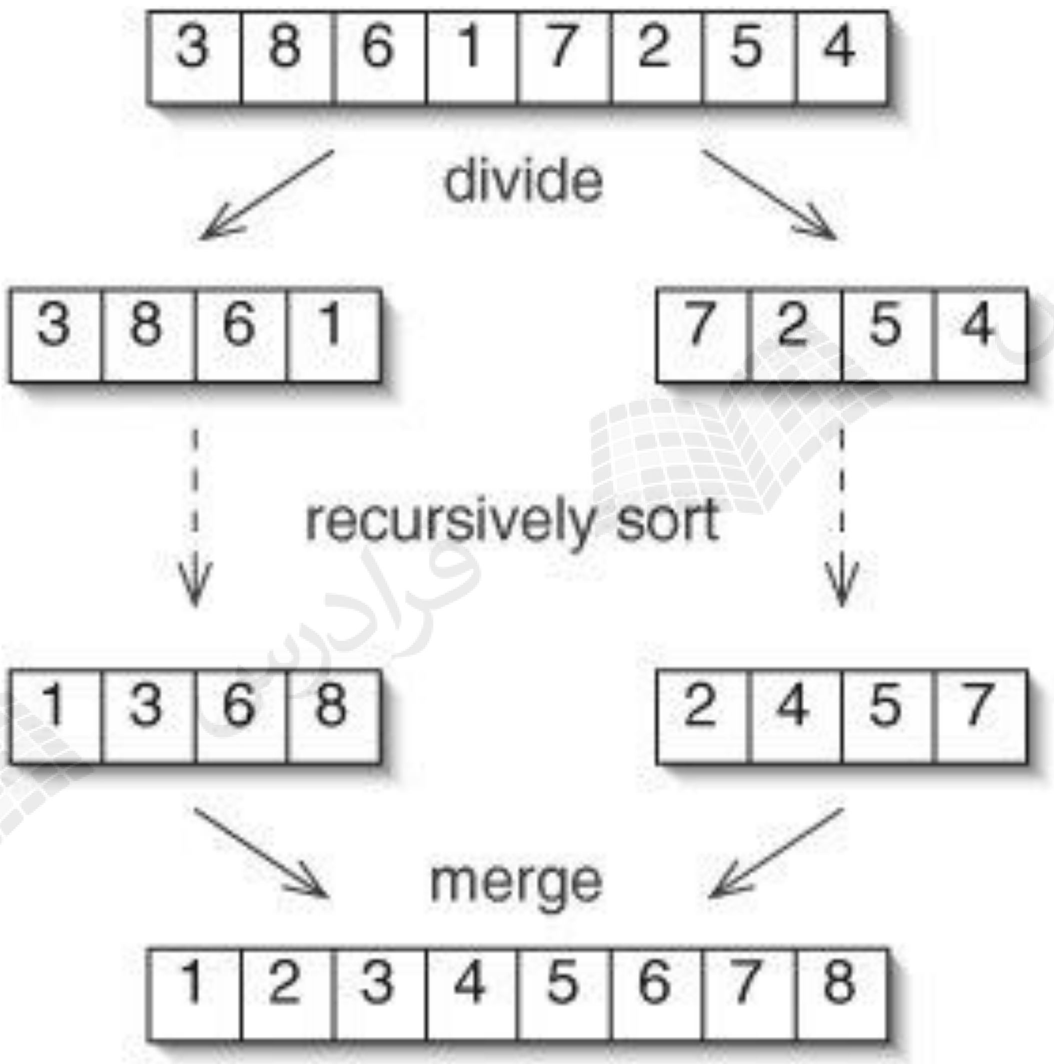
$$T(n) = T(\frac{n}{2}) + 1 \Rightarrow O(\lg n)$$

مثال


1 4 4 7 7 8 11 19 21 23 24 30
mid
1 4 4 7 7 8 11 19 21 23 24 30
mid
1 4 4 7 7 8 11 19 21 23 24 30
mid
1 4 4 7 7 8 11 19 21 23 24 30

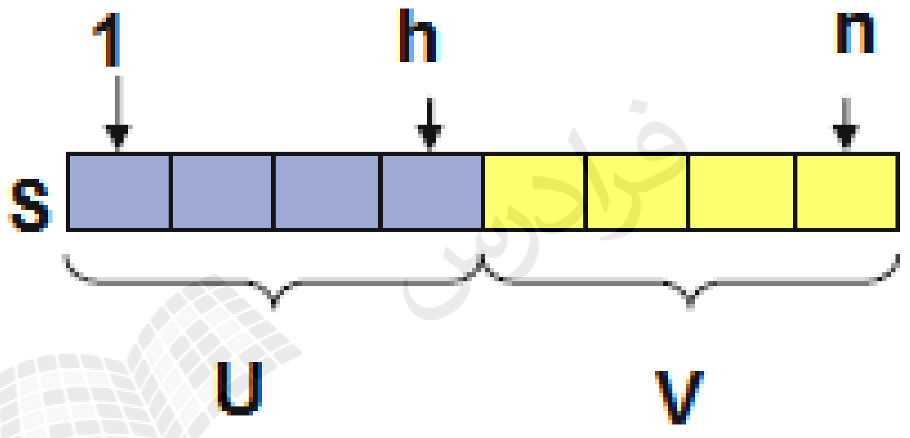
مرتب سازی ادغامی (Merge Sort)




3 8 6 1 7 2 5 4
divide
3 8 6 1
7 2 5 4
recursively sort
1 3 6 8
2 4 5 7
merge
1 2 3 4 5 6 7 8

الگوریتم

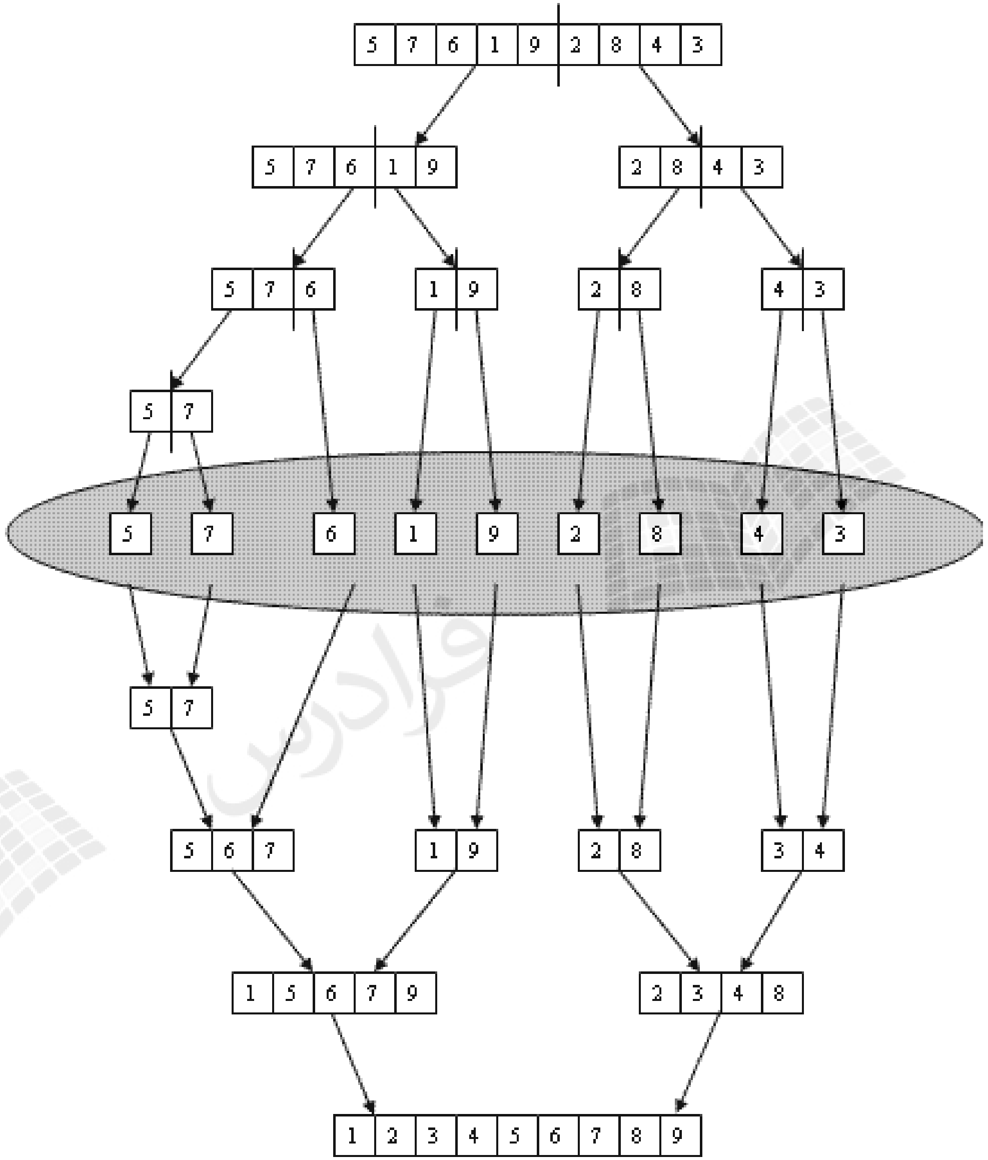
```
mergesort ( S[ ] , n )
{
   h = [n/2];
   m = n – h;
   if (n > 1)
      {
         copy  S[1..h]  to U[1..h];
         copy  S[h +1..n]  to  V[1..m];
         mergesort ( U , h );
         mergesort ( V , m );
          merge (U , h , V , m , S);
      }
 }
```



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

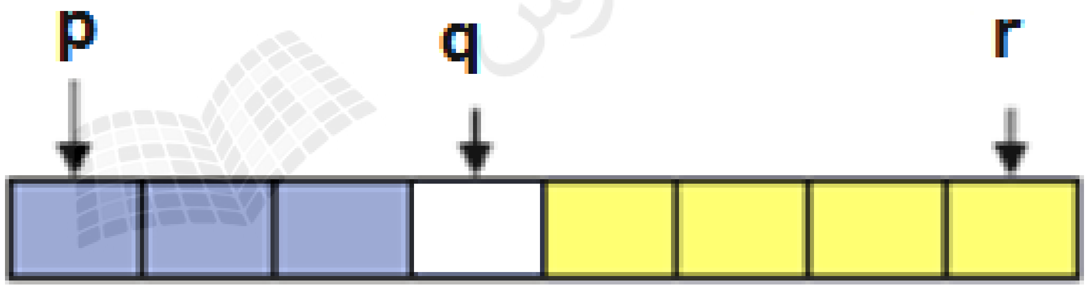
$$\Rightarrow T(n) = n \lg n - (n-1)$$

$$\in \theta(n \lg n)$$


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

مرتب سازی سریع (Quick sort)

```
QuickSort (A , p , r)
{
  if ( p < r )
   {
     q = Partition (A , p , r) ;
     QuickSort (A , p , q-1) ;
     QuickSort (A , q+1, r ) ;
   }
}
```

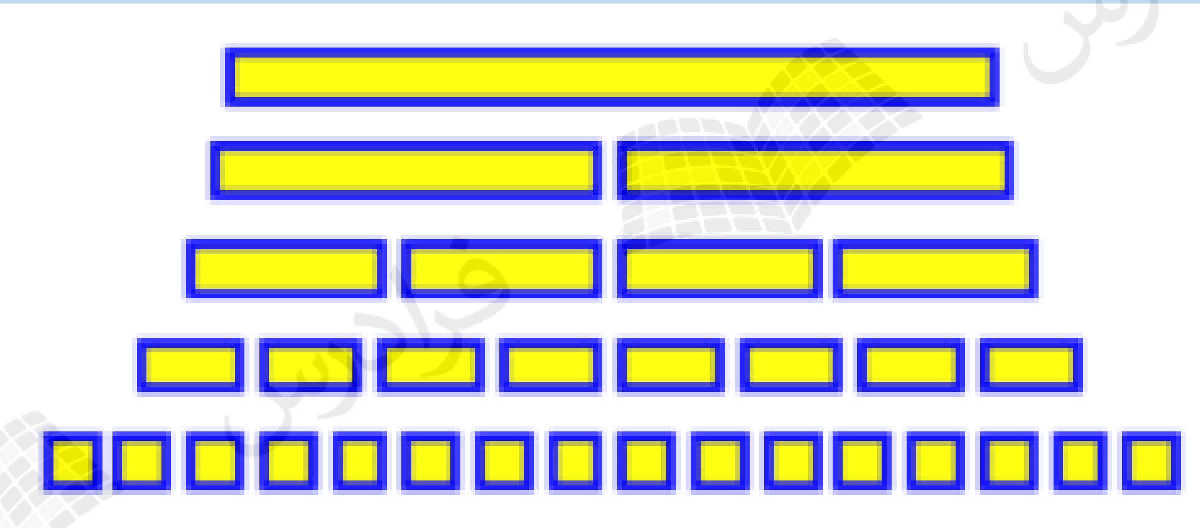


الگوریتم پارتیشن بندی

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

عملکرد مرتب سازی سریع

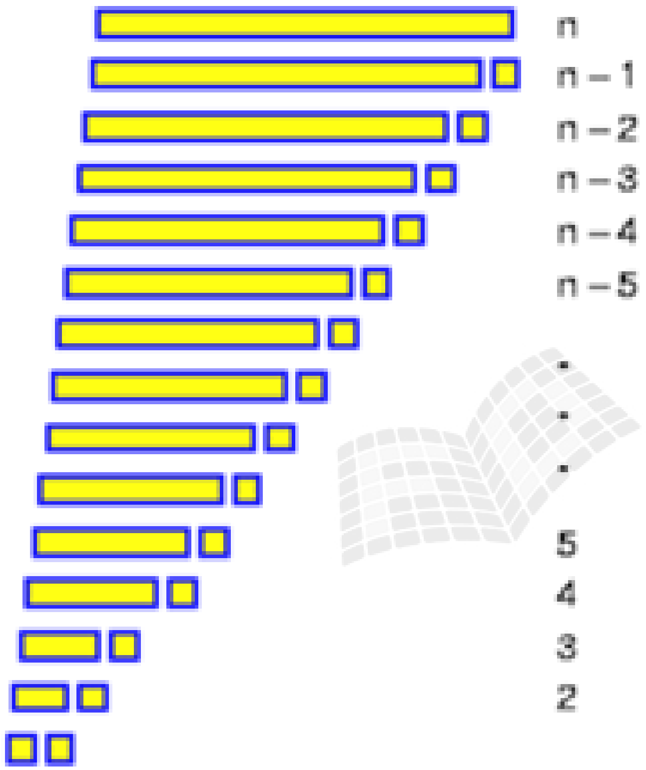
حالت خوب و میانگین :



$$T(n) = T(n/2) + T(n/2)+O(n) \Rightarrow O(nlogn)$$

بدترین حالت مرتب سازی سریع

در صورتی که اولین عنصر یا آخرین عنصر را به عنوان محور انتخاب کنیم، مرتب سازی سریع برای یک آرایه مرتب، بدترین عملکرد را خواهد داشت.



$$T(n) = T(n-1) + O(n) \Rightarrow O(n^2)$$

حالت میانگین مرتب سازی سریع

$$
\begin{array}{lcccc}
T(n) = & T(0) & + & T(n-1) & + & O(n) \\
T(n) = & T(1) & + & T(n-2) & + & O(n) \\
\vdots & \vdots & & \vdots & & \\
T(n) = & T(n-2) & + & T(1) & + & O(n) \\
T(n) = & T(n-1) & + & T(0) & + & O(n) \\
\hline
nT(n) = & \sum_{i=0}^{n-1} T(i) & + & \sum_{i=0}^{n-1} T(i) & + & nO(n)
\end{array}
$$

Average case: $T(n) = \frac{2}{n}\sum_{i=0}^{n-1} T(i) + cn$

Average case

$$T(n) = \frac{2}{n}\sum_{i=0}^{n-1} T(i) + cn$$

T(n) = O(n.Log(n))

Initiation: $n = 2 \Longrightarrow T(2) = T(1) + 2c = O(1)$

Hypothesis: $\forall i < n \Longrightarrow T(i) = O(i.Log(i)) \leq c' i.Log(i)$

Induction step: prove the statement for n:

$$\begin{aligned}
T(n) &\leq \frac{2}{n}\sum_{i=0}^{n-1} c' i.Log(i) + cn \\
&\leq \frac{2c'}{n}\left(\sum_{i=0}^{n/2} i.Log(n/2) + \sum_{i=n/2+1}^{n-1} i.Log(n)\right) + cn \\
&\leq \frac{2c'}{n}\left(\sum_{i=0}^{n/2} i.Log(n) - \sum_{i=0}^{n/2} i + \sum_{i=n/2+1}^{n-1} i.Log(n)\right) + cn \\
&\leq \frac{2c'}{n}\left(Log(n)\frac{n(n-1)}{2} - \frac{(n/2)(n/2-1)}{2}\right) + cn \\
&= O(n.Log(n))
\end{aligned}$$

الگوریتم پارتیشن بندی

```
Partition(A, s, e, m){
    x ← A[s]; i ← s+1; j ← e;
    do{
        while(A[i] < x) i++;
        while(A[j] > x) j--;
        if(i < j) swap(A[i], A[j]);
    }while(i < j);
    swap(A[s], A[j]);
    return(j);
}
```

5	1	6	12	3	9	20	8
x	i						j

Strassen’s matrix multiplication

ضرب ماتریس های استراسن

استراسن الگوریتمی را ارائه داد که پیچیدگی آن، از لحاظ ضرب (و همچنین از لحاظ جمع و تفریق) بهتر از پیچیدگی درجه سوم است.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 2 & 0 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 4 & 4 \\ 10 & 8 \end{bmatrix}$$

$$\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} * \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix} = \begin{bmatrix} A_{00} * B_{00} + A_{01} * B_{10} & A_{00} * B_{01} + A_{01} * B_{11} \\ A_{10} * B_{00} + A_{11} * B_{10} & A_{10} * B_{01} + A_{11} * B_{11} \end{bmatrix}$$

استراسن

$$\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} * \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix} = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{bmatrix}$$

$M_1=(A_{00}+A_{11})*(B_{00}+B_{11})$

$M_2=(A_{10}+A_{11})*B_{00}$

$M_3=A_{00}*(B_{01}-B_{11})$

$M_4=A_{11}*(B_{10}-B_{00})$

$M_5=(A_{00}+A_{01})*B_{11}$

$M_6=(A_{10}-A_{00})*(B_{00}+B_{01})$

$M_7=(A_{01}-A_{11})*(B_{10}+B_{11})$

مثال: محاسبه M1

$$A=\left[\begin{array}{cc|cc}1&2&3&4\\5&6&7&8\\\hline 9&1&2&3\\4&5&6&7\end{array}\right]\qquad B=\left[\begin{array}{cc|cc}8&9&1&2\\3&4&5&6\\\hline 7&8&9&1\\2&3&4&5\end{array}\right]$$

$$M_1=(A_{00}+A_{11})*(B_{00}+B_{11})$$

$$\left(\begin{bmatrix}1&2\\5&6\end{bmatrix}+\begin{bmatrix}2&3\\6&7\end{bmatrix}\right)\times\left(\begin{bmatrix}8&9\\3&4\end{bmatrix}+\begin{bmatrix}9&1\\4&5\end{bmatrix}\right)=\begin{bmatrix}3&5\\11&13\end{bmatrix}\times\begin{bmatrix}17&10\\7&9\end{bmatrix}$$

$$=\begin{bmatrix}3\times17+5\times7 & 3\times10+5\times9\\11\times17+13\times7 & 11\times10+13\times9\end{bmatrix}=\begin{bmatrix}86&75\\278&227\end{bmatrix}$$

$$\begin{bmatrix}A_{00}&A_{01}\\A_{10}&A_{11}\end{bmatrix}*\begin{bmatrix}B_{00}&B_{01}\\B_{10}&B_{11}\end{bmatrix}=\begin{bmatrix}M_1+M_4-M_5+M_7 & M_3+M_5\\M_2+M_4 & M_1+M_3-M_2+M_6\end{bmatrix}$$

تحلیل پیچیدگی زمانی الگوریتم استراسن

دستور بازگشتی (**عمل اصلی ضرب است**):

$$T(n) = 7T(\frac{n}{2})$$

$$T(1) = 1$$

$$T(n) = \theta(n^{\lg 7}) \approx \theta(n^{2.81})$$

مثال

اگر در ضرب ماتریس‌ها به روش استراسن (Strassen) ، مساله کوچک ضرب ماتریس های 2×2 باشد، برای ضرب دو ماتریس 8×8 ، چند **ضرب** عددی صورت می‌پذیرد؟

$$T(n) = 7T(\frac{n}{2})$$

$$T(2) = 8$$

$$T(4) = 7T(2) = 7\times 8 = 56$$

$$T(8) = 7T(4) = 7\times 56 = 392$$

الگوریتم استراسن

```
strassen (n  , A  , B , C)
{
      if (n <= threshold)
            compute  C = A × B using the standard algorithm;
      else{
            partition  A  into four submatrices   A11 , A12 , A21 , A22 ;
            partition  B  into four submatrices   B11 , B12  , B21 , B22 ;
            compute  C = A × B   using   Strassen's Method;
          }
}
```

مقدار threshold نقطه ای است که در آن احساس می شود استفاده از الگوریتم ضرب استاندارد بهتر از فراخوانی بازگشتی روال Strassem باشد.

ضرب دو عدد صحیح بزرگ

برای انجام اعمال محاسباتی روی اعداد صحیحی بزرگتر از حد قابل نمایش توسط سخت افزار کامپیوتر، باید از روش تقسیم و حل استفاده کرد.

اگر n تعداد ارقام عدد صحیح u باشد، آن را به دو عدد صحیح یکی x با $\lceil n/2 \rceil$ رقم و دیگری y با $\lfloor n/2 \rfloor$ تبدیل می‌کنیم به صورت: $u = x \times 10^m + y$ که $m = \lfloor n/2 \rfloor$.

$$12345 = 123 \times 10^2 + 45$$

ضرب u و v

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

پس می توان v و u را با ۴عمل ضرب روی اعداد صحیح (با حدود نیمی از ارقام) و اجرای عملیات زمان خطی، در هم ضرب کنیم.

مثال

ضرب دو عدد 567832 و 9423723 :

$$567,832\times 9,423,723=(567\times 10^3+832)(9423\times 10^3+723)$$
$$=567\times 9,423\times 10^6+(567\times 723+9423\times 832)\times 10^3+832\times 723$$

سپس این اعداد صحیح کوچکتر را با همین روال ضرب می‌کنیم:

$$567\times 9423=(5\times 10^2+67)(94\times 10^2+23)$$
$$=5\times 94\times 10^2+(5\times 23+67\times 94)\times 10^2+67\times 23$$

در این صورت عمل ضرب اعداد حداکثر دو رقمی را به طریق معمولی انجام می دهیم.

الگوریتم ضرب دو عدد صحیح بزرگ

```
prod( u , v){
  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 = u  rem  10^m ;
          w= v  divide  10^m ;
          z = v  rem  10^m ;
          return   prod(x,w) ×10^2m  + (prod(x,z)+ prod(w,y))×10^m  + prod(y,z);
        }
}
```

تحلیل الگوریتم در بدترین حالت

$$T(n) = 4T(\frac{n}{2}) + cn \Rightarrow \theta(n^{\log_2^4}) = \theta(n^2)$$

الگوریتم سریعتر:

در الگوریتم قبلی برای محاسبه ضرب دو عدد به ۴ عمل ضرب نیاز بود که با یک ابتکار جالب تعداد آن را به ۳ عمل ضرب، کاهش می دهیم.

$$T(n) = 3T(\frac{n}{2}) + cn \Rightarrow \theta(n^{\log_2^3}) = \theta(n^{1.58})$$

مرتبه ضرب دو عدد k رقمی : $\mathbf{k^{\lg_2^3}}$

$$u.v = x.w \times 10^{2m} + (x.z + w.y) \times 10^{m} + y.z$$

$$u.v = x.w \times 10^{2m} + ((x+y)(w+z) - x.w - y.z) \times 10^{m} + y.z$$

$$x.w \, , y.z \, , (x+y)(w+z)$$

مسئله انتخاب

انتخاب i امین کوچکترین عنصر در آرایه n عنصری (عنصر با رتبه i)

$i=1$: مینیمم

$i=n$: ماکزیمم

میانه : $i = \lfloor (n+1) / 2 \rfloor$ or $\lceil (n+1)/2 \rceil$

پیدا کردن هفتمین کوچکترین عنصر

6	10	13	5	8	3	2	11

$i = 7$ (2,3,5,6,8,10,11,13)

عنصر اول را محور در نظر می گیریم و پارتیشن بندی می کنیم:

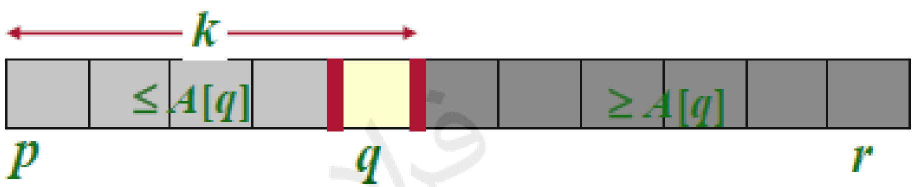
2	5	3	6	8	13	10	11

$k = 4$

به طور بازگشتی در قسمت صورتی رنگ به دنبال **سومین** کوچکترین عنصر می گردیم.

الگوریتم انتخاب

```
select (A[ ] , p , r , i ){
   if (p == r)  return  A[p];
   q = R-partition(A,p,r);
   k = q-p+1; // k =rank(A[q])
   if  ( i == k )  return  A[q];
   else  if ( i < k )
                  return  select ( A , p , q-1 , i )
          else
                  return  select ( A , q+1 , r , i-k )
}
```



این الگوریتم در بدترین حالت از مرتبه توان ۲ است.

الگوریتم انتخاب k امین عنصر (میانه میانه ها)

ابتدا همه عناصر را به دسته های ۵ تایی (به جز احتمالاً یک دسته) تقسیم می‌کنیم.

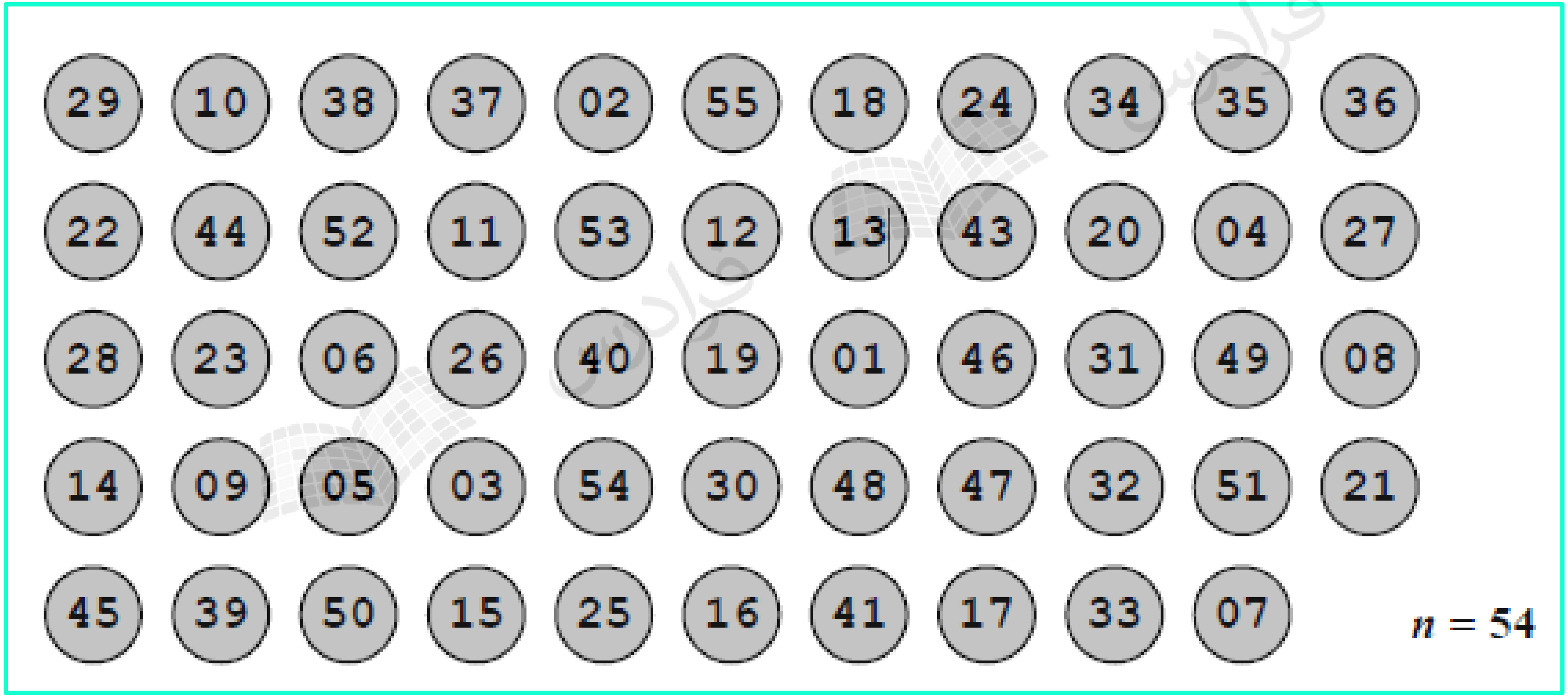
میانه هر دسته را به دست می‌آوریم و سپس میانه میانه ها را به صورت بازگشتی پیدا می‌کنیم.

این عنصر را به عنوان محور انتخاب می‌کنیم و عمل **partition** را بر روی آرایه عناصر انجام می‌دهیم.

همین الگوریتم را بصورت بازگشتی (و برای یک k دیگر) بر روی یکی از بخش‌ها اجرا می‌کنیم تا عنصر مورد نظر پیدا شود.

مثال

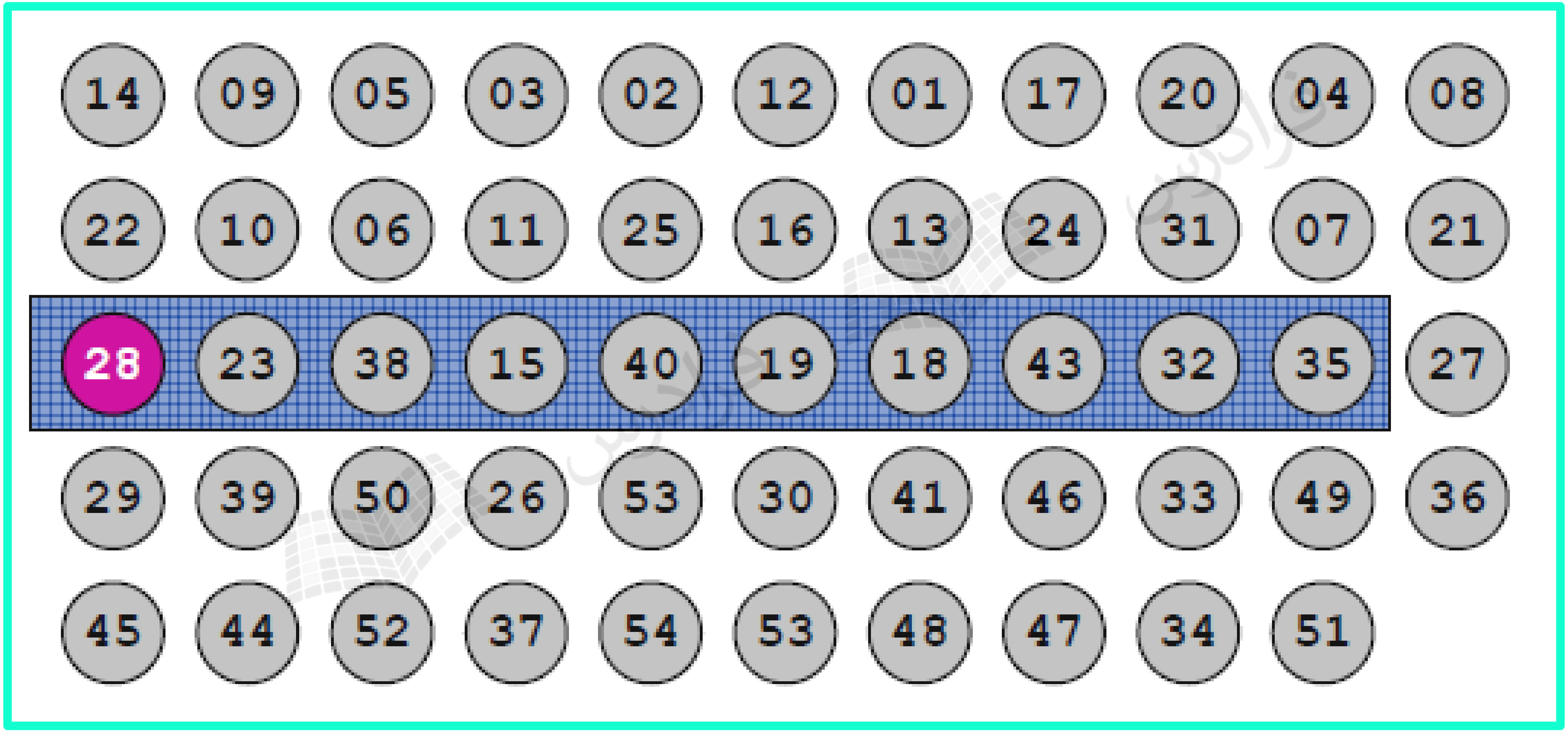
تقسیم ۵۴ عنصر به گروهایی با $n/5$ عنصر (در نتیجه ۱۱ گروه خواهیم داشت)



مرتب کردن هر گروه

14	09	05	03	02	12	01	17	20	04	08
22	10	06	11	25	16	13	24	31	07	21
28	23	38	15	40	19	18	43	32	35	27
29	39	50	26	53	30	41	46	33	49	36
45	44	52	37	54	53	48	47	34	51	

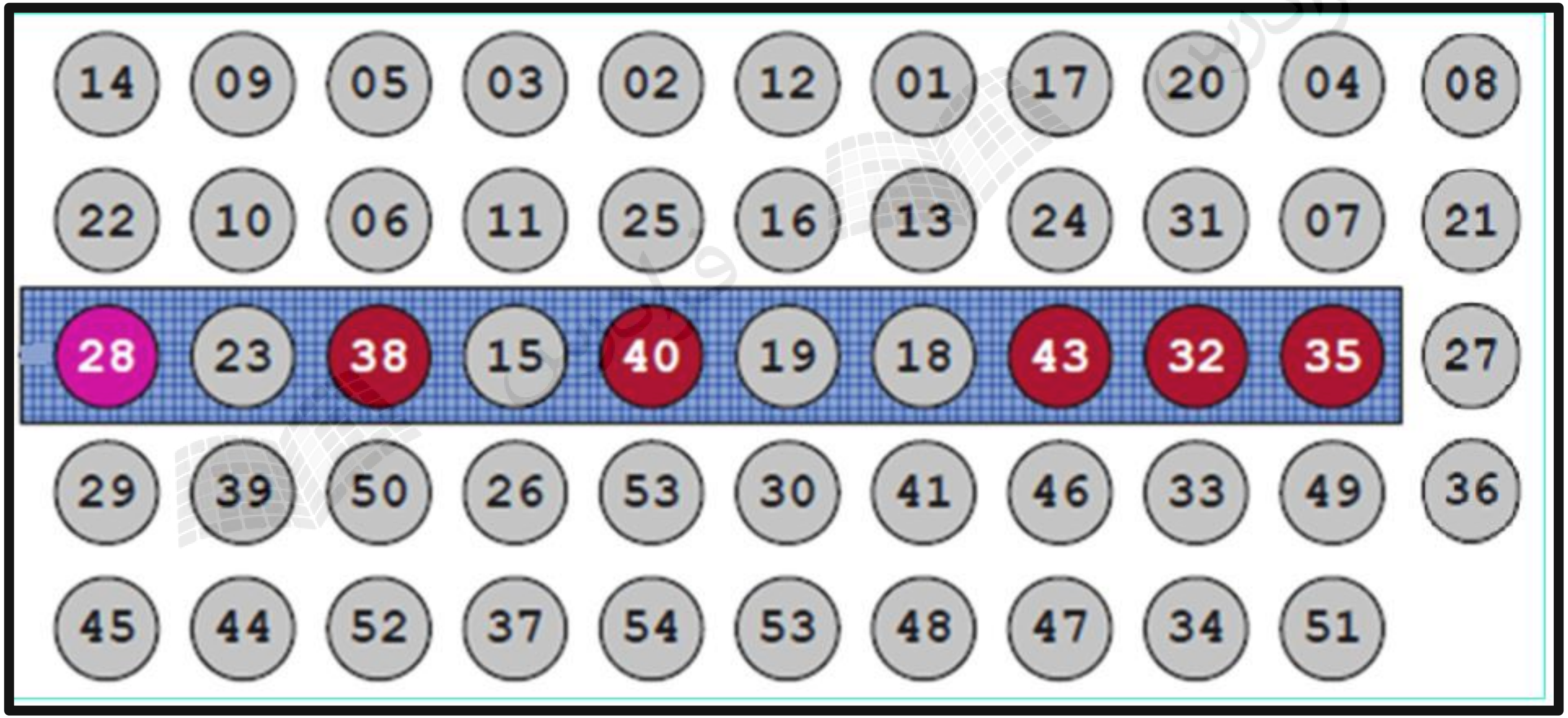
مشخص کردن میانه هر گروه



انتخاب میانه میانه ها(عدد ۲۸)

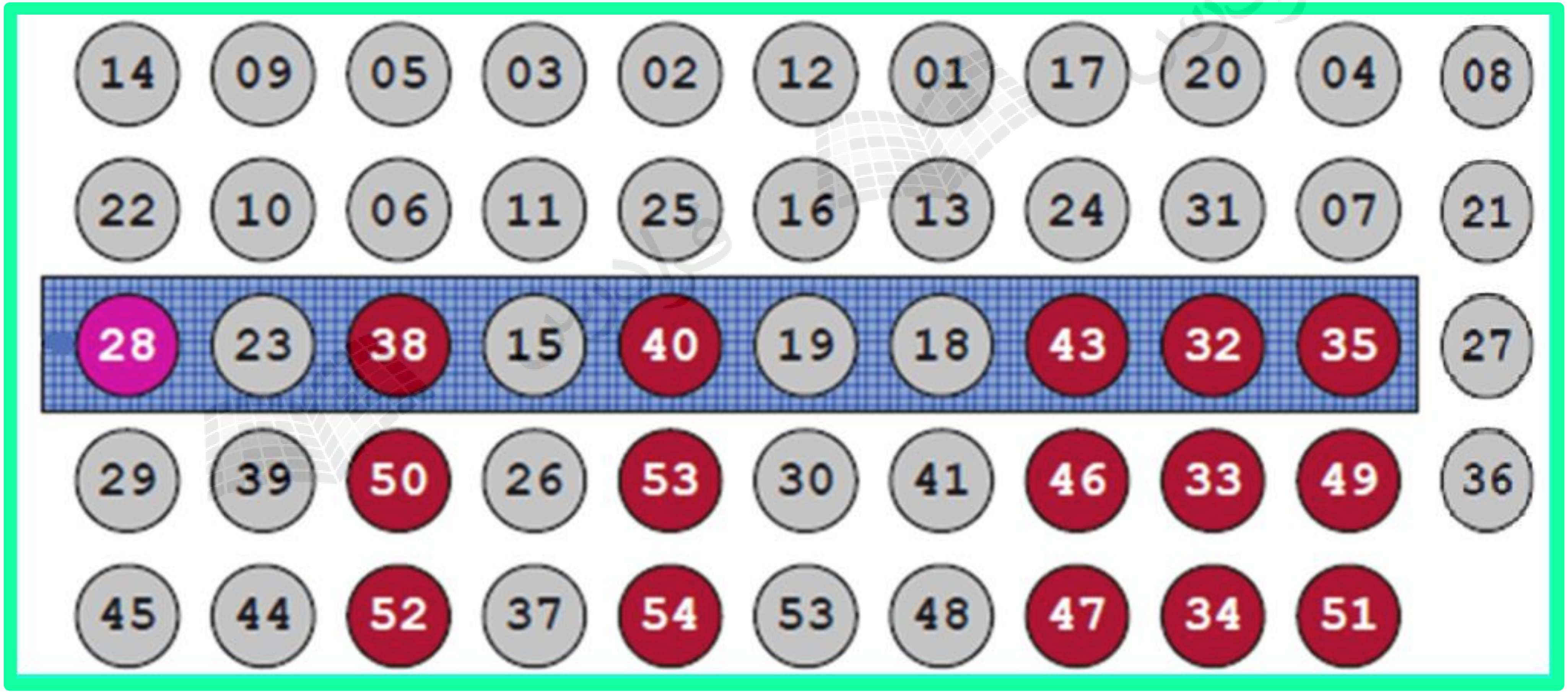
حداقل تعداد میانه های بزرگتر از میانه میانه ها

$$\lceil \lceil n/5 \rceil /2 \rceil - 2 = \lceil n/10 \rceil - 2$$



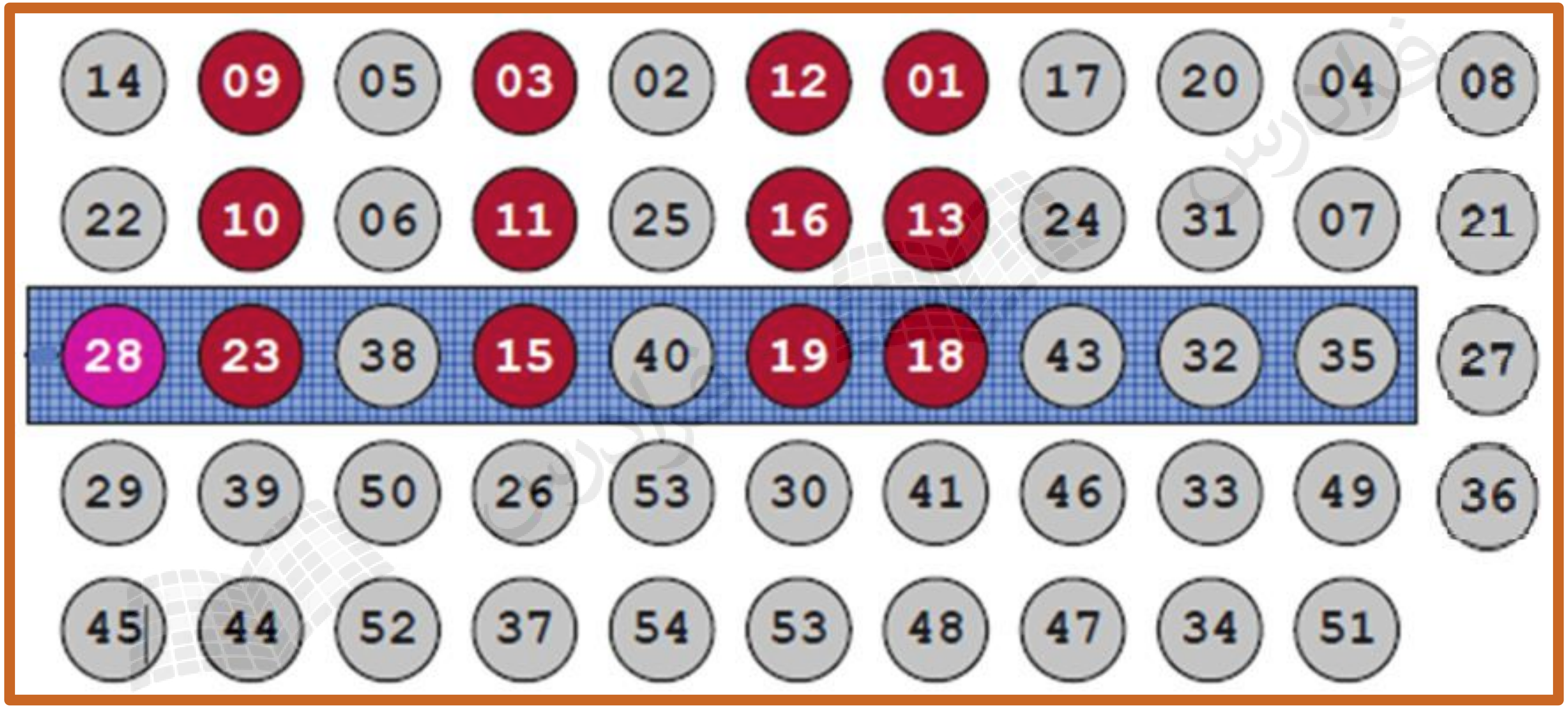
حداقل عناصر بزرگتر از میانه میانه ها

$$3(\lceil n/10 \rceil - 2)$$



حداقل عناصر کوچکتر از میانه ها

$3(\lceil n/10 \rceil - 2)$



تعداد عناصری که تابع انتخاب برای آن ها به طور بازگشتی صدا زده می شود :

$$n-\left(\frac{3n}{10}-6\right)=\frac{7n}{10}+6$$

الگوریتم انتخاب

SELECT(A, i)

1. Divide n elements into $\lfloor n/5 \rfloor$ groups of 5, plus extra — O(n)
2. Find medians of each group by insertion sort. — O(n)
3. Find median x of the $\lfloor n/5 \rfloor$ medians by calling SELECT( ) — T(n/5)
4. Partition the n elements around pivot x . Let k = rank(x) — O(n)
5. if (i = k) then — T(7n/10 + 6)

    return x

  if (i < k) then

    SELECT(left partition , i)

  else

    SELECT (right partition , i-k)

$$T(n) = T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 6\right) + O(n) \quad \Rightarrow O(n)$$

$$T(n)=T\left(\frac{n}{m}\right)+T\left(\frac{(3m-1)n}{4m}+c\right)+O(n)$$

گروه های m عنصری

$$T(n)=T\left(\left\lceil\frac{n}{3}\right\rceil\right)+T\left(\frac{2n}{3}+4\right)+O(n)\Rightarrow O(n.\lg n)$$

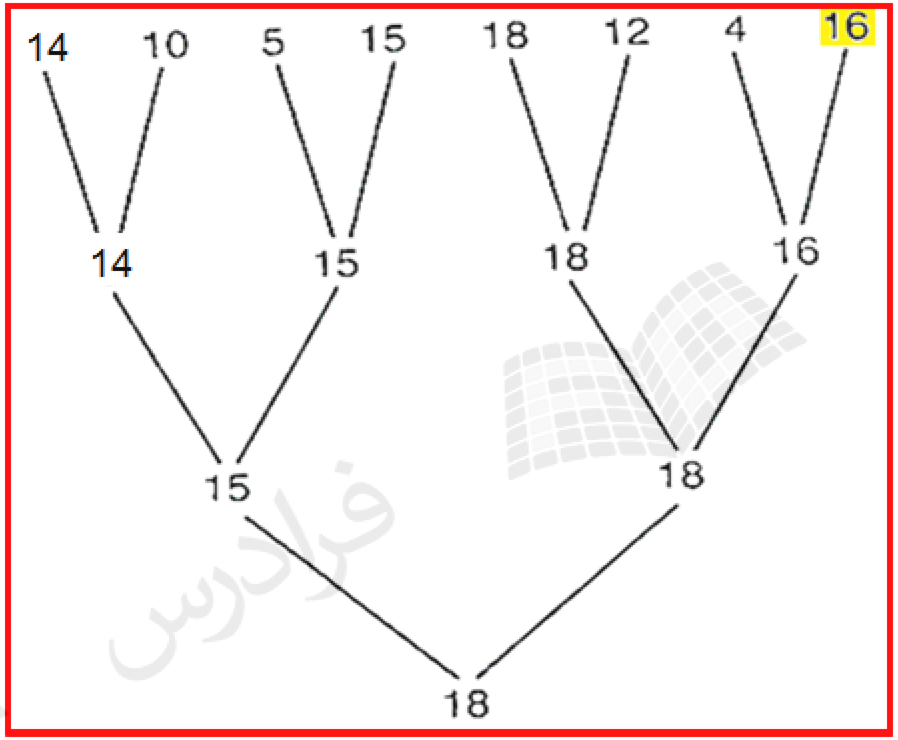
گروه های 3 عنصری

$$T(n)=T\left(\left\lceil\frac{n}{7}\right\rceil\right)+T\left(\frac{5n}{7}+8\right)+O(n)\Rightarrow O(n)$$

گروه های 7عنصری

مقدار $m=5$ ، کوچکترین مقدار فرد است که منجر به کارایی خطی می شود. به همین علت در توضیح الگوریتم انتخاب، عناصر را به دسته های پنج تایی تقسیم کردیم.

یافتن بزرگ ترین کلید دوم (تورنمنت)



کلیدی که یک مقایسه را به 18 ببازد ، به لیست اضافه می شود. **[12,16,15]**

اگر ۸ کلید داشته باشیم، در دور اول، ۴مقایسه، دور دوم، ۲مقایسه و دور آخر ۱ مقایسه خواهیم داشت. (n-1)

برنده دور آخر، بزرگترین کلید است. بازنده دور آخر الزاما بزرگ ترین کلید دوم نیست.

تعداد اعداد لیست : $\log n$ (ارتفاع درخت مقایسه)

تعداد مقایسه برای پیداکردن بزرگترین عدد در این لیست : $\boldsymbol{\log n - 1}$

تعداد کل مقایسه‌های لازم برای یافتن بزرگ‌ترین کلید دوم:

$$\boldsymbol{(n-1)+(\lg n-1)=n+\lg n-2}$$

مثال : تعداد مقایسه مورد نیاز برای پیدا کردن دومین کوچکترین عنصر در آرایه 16 عنصری :

$$\boldsymbol{16+\log 16-2=16+4-2=18}$$

یافتن k امین کوچک‌ترین کلید

حد پایین برای یافتن k امین کوچک‌ترین کلید در مجموعه n کلیدی:

$$n + (k-1)\left\lceil \lg\left(\frac{n}{k-1}\right) \right\rceil - k \quad \Rightarrow \quad \theta(n)$$

$k > 1$

این اسلاید ها بر مبنای نکات مطرح شده در فرادرس

«آموزش طراحی الگوریتم»

تهیه شده است.

برای کسب اطلاعات بیشتر در مورد این آموزش به لینک زیر مراجعه نمایید

faradars.org/fvsft1092