موسسه آموز ش عالی ماد

تحلیل و طراحی الگوریتم ها

فصل دوم:
روش تقسیم و غلبه
Divide & Conquer

سید جبار بارخدا
barkhoda@gmail.com

مهر ۹۲

مطالب فصل

- معرفی روش تقسیم و حل
- حل مسائل نمونه و تحلیل آنها
  - جستجوی دودویی
  - مرتب سازی ادغامی
  - مرتب سازی سریع
  - ضرب استراسن
  - ماکزیمم و مینمم یک آرایه
  - ضرب اعداد صحیح بزرگ
  - ضرب چند جمله ای ها

تقسیم و غلبه (divide and conquer)

- نمونه ای از یک مساله را به صورت بازگشتی به تعدادی نمونه کوچکتر تقسیم می کند (تا زمانی که راه حل نمونه های کوچکتر به سادگی قابل تعیین باشند) و از حل نمونه های کوچکتر به حل نمونه بزرگ می رسد.
  - تقسیم (divide): مسئله به تعدادی زیر مسئله تقسیم می شود.
  - غلبه (conquer): زیرمسئله ها به صورت بازگشتی حل می شوند.
  - ترکیب (combine): در صورت لزوم، ترکیب حل زیرمسئله ها برای یافتن جواب مسئله کلی
- اگر پس از تقسیم مسأله، زیرمسأله‌ها هنوز بزرگ و غیر قابل حل هستند، آنها را دوباره به چند زیرمسأله دیگر تقسیم می‌کنیم.
- باید مسأله ها با تقسیمات متوالی آنقدر کوچک شوند تا دیگر مشکلی برای حل آنها نداشته باشیم.
- روش تقسیم و غلبه یک رهیافت بالا به پایین (top-down) است که توسط روتین های بازگشتی به کار می رود.
- حل یک نمونه سطح بالای مسئله با رفتن به جزء و به دست آوردن حل نمونه های کوچکتر حاصل می شود.

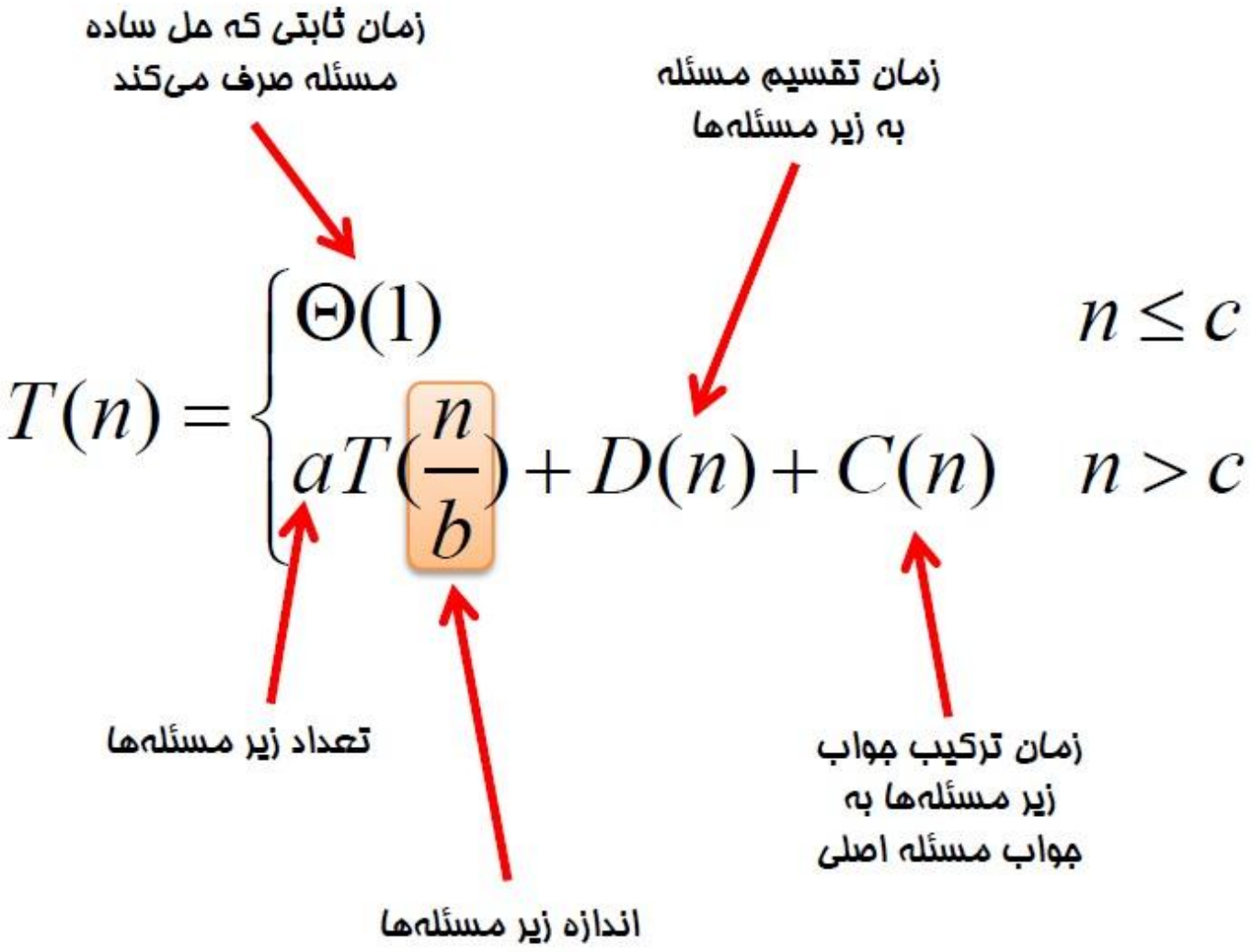
شکل کلی الگوریتم تقسیم و حل

```
Alg   D_and_C ( p )
{
    if small(p) then
        return S(p) ;
    else
    {
        divide p int smaller instances p۱,p۲,…,pk, // k > ۱
        apply D_and_C to each of these subproblems ;
        return combine (D_and_C (p۱) , D_and_C (p۲) , ... , D_and_C (pk)) ;
      }
}
```

شکل کلی الگوریتم تقسیم و حل

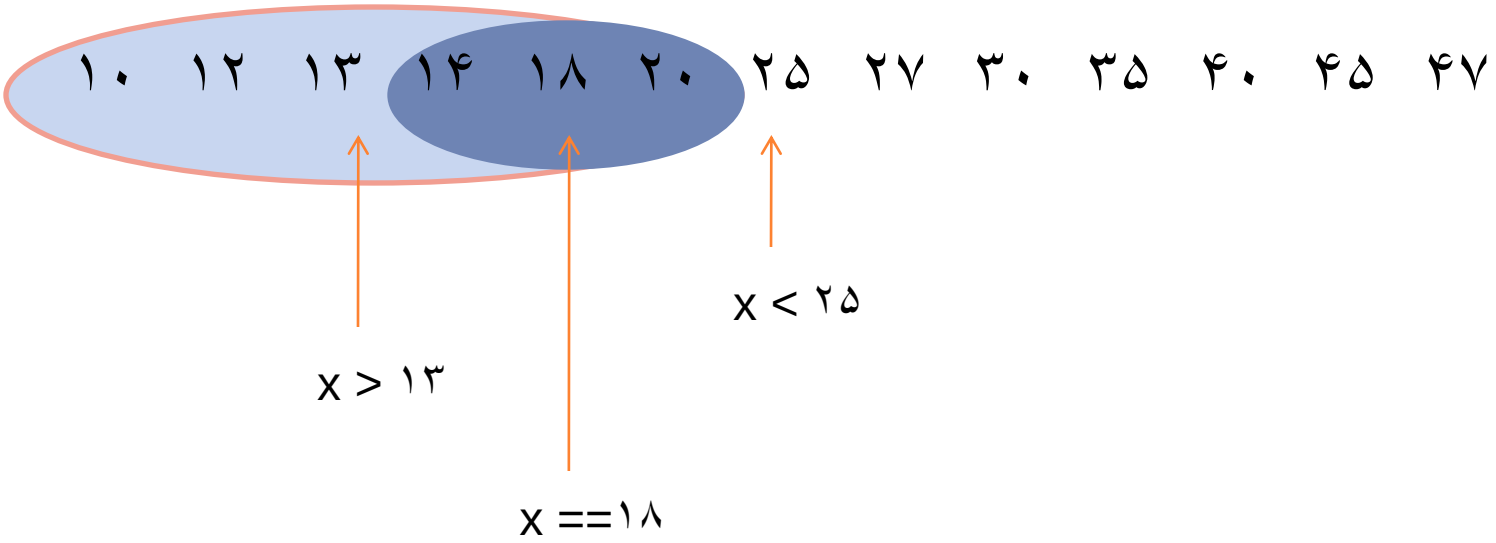
- **small(p)** : یک تابع بولین (boolean) است و مشخص می‌کند که آیا ورودی P (یا مسأله P) به اندازه کافی کوچک شده است یا خیر؟
  - مسأله باید آنقدر کوچک شود که بتوان بدون تقسیم، آن را حل نمود. در صورتی که مسأله اندازه کافی کوچک شده باشد، خروجی تابع small (p) برابر true خواهد بود.
- **s(p)** : تابعی است که مسأله کوچک شده را حل می‌کند. اگر small (p) برابر true باشد، s(p) اجرا خواهد شد.
- **divide** : در هر بار فراخوانی، مسأله را به k زیر مسأله تقسیم می کند.
- **recursive** : تمام زیرمسأله‌های ایجاد شده را به صورت بازگشتی، دوباره با استفاده از الگوریتم تقسیم و حل فوق حل می‌کند.
- **combine** : جواب همه زیرمسأله‌ها را با هم ترکیب می‌کند.

تحلیل الگوریتم های تقسیم و حل



جست و جوی دودویی (binary search)

- مثال: می خواهیم در آرایه مرتب زیر $x=۱۸$ را جست و جو کنیم.



جست و جوی دودویی (binary search)

```
1.  index binSearch(index low, index high)
2.  {
3.      index mid;
4.      if (low > high)
5.          return -1;
6.      else
7.      {
8.          mid = [(low + high)/2];
9.          if (x == S[mid])
10.             return mid;
11.         else if (x  <  S[mid])
12.             return binSearch (low, mid - 1);
13.         else return binSearch (mid + 1, high);
14.     }
15. }
```

پیچیدگی زمانی جستجوی دودویی

- در بدترین حالت W(n)
  - عمل اصلی: تعداد مقایسه
  - اندازه ورودی: تعداد عناصر آرایه

- حل:

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

$$W(n) = \lg n + 1$$

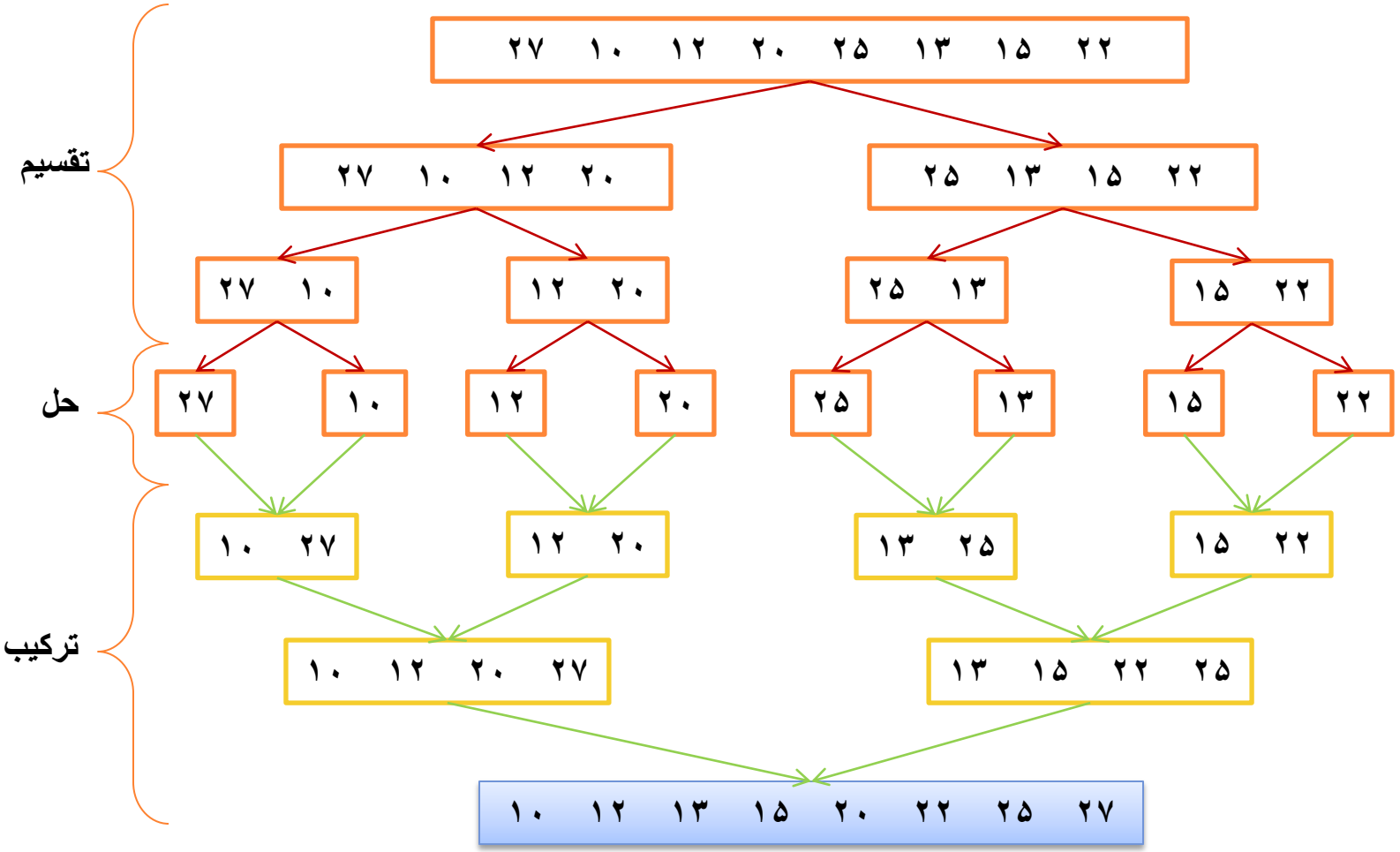
✓اگر n به توانی از دو محدود نباشد:

$$W(n) = \lfloor \lg n \rfloor + 1 \in \Theta(\lg n)$$

مرتب سازی ادغامی (merge sort)

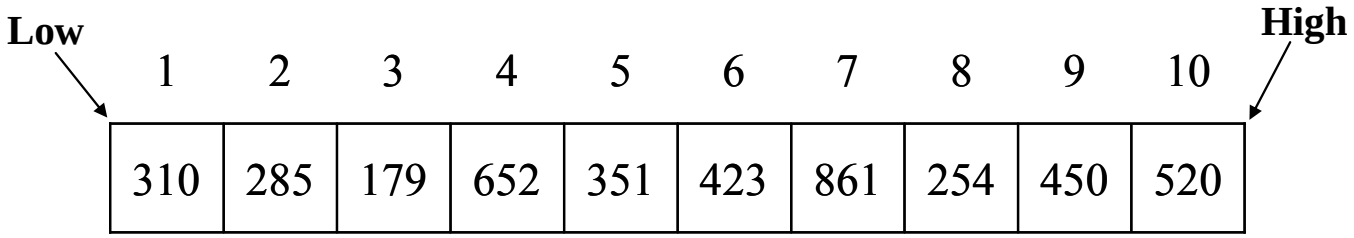
- **تقسیم** آرایه n عنصری به دو زیر آرایه به اندازه $n/۲$
- **حل** هر یک از زیر آرایه ها با مرتب سازی آن. اگر زیرآرایه به اندازه کافی کوچک نبود، برای حل آن به روش بازگشتی عمل می کنیم.
- **ترکیب** حل های زیر آرایه ها با ادغام آن ها در یک آرایه مرتب.
  - ادغام دوطرفه (two-way merge): به معنای ترکیب دو آرایه مرتب شده در یک آرایه مرتب است.

Merge sort


تقسیم
حل
ترکیب
۲۷ ۱۰ ۱۲ ۲۰ ۲۵ ۱۳ ۱۵ ۲۲
۲۷ ۱۰ ۱۲ ۲۰
۲۵ ۱۳ ۱۵ ۲۲
۲۷ ۱۰
۱۲ ۲۰
۲۵ ۱۳
۱۵ ۲۲
۲۷
۱۰
۱۲
۲۰
۲۵
۱۳
۱۵
۲۲
۱۰ ۲۷
۱۲ ۲۰
۱۳ ۲۵
۱۵ ۲۲
۱۰ ۱۲ ۲۰ ۲۷
۱۳ ۱۵ ۲۲ ۲۵
۱۰ ۱۲ ۱۳ ۱۵ ۲۰ ۲۲ ۲۵ ۲۷

الگوریتم مرتب سازی ادغامی (Merge sort)

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



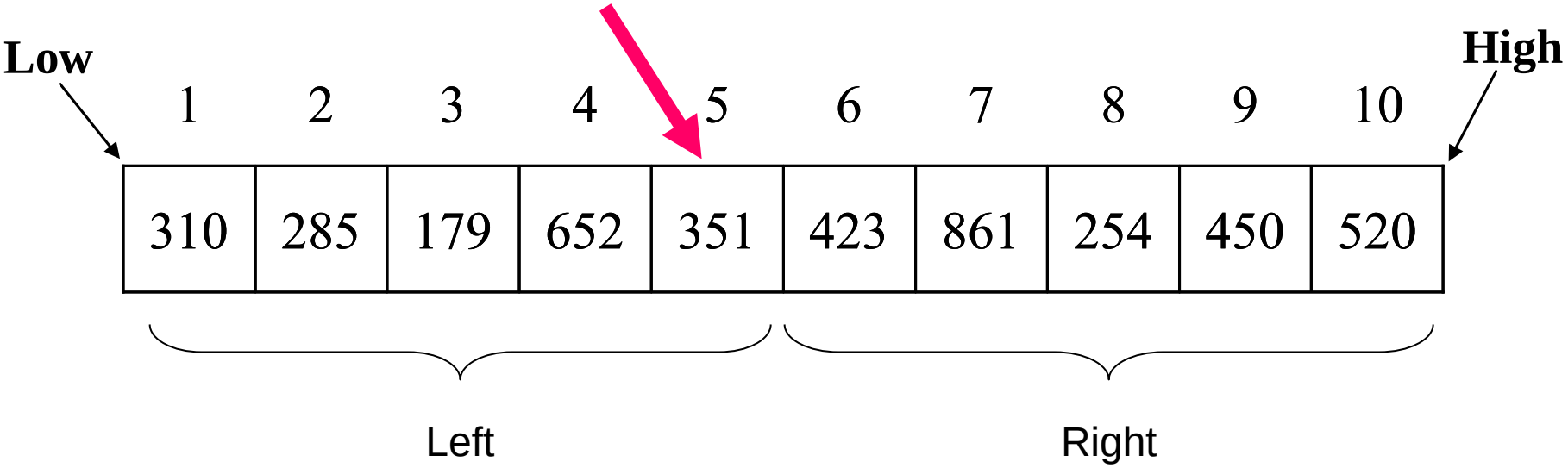

```
Alg mergesort (low , high)
{
  if (low < high)                          // !small(p)
  {
        mid := ⌊(low + high) / 2⌋ ;        // Divide
        mergsort (low , mid) ;             // recursive (left)
        mergesort (mid + 1  ,  high) ;     // recursive (right)
        merge (low , mid , high)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

Merge sort

لیست زیر را با استفاده از الگوریتمmerge sort مرتب کنید

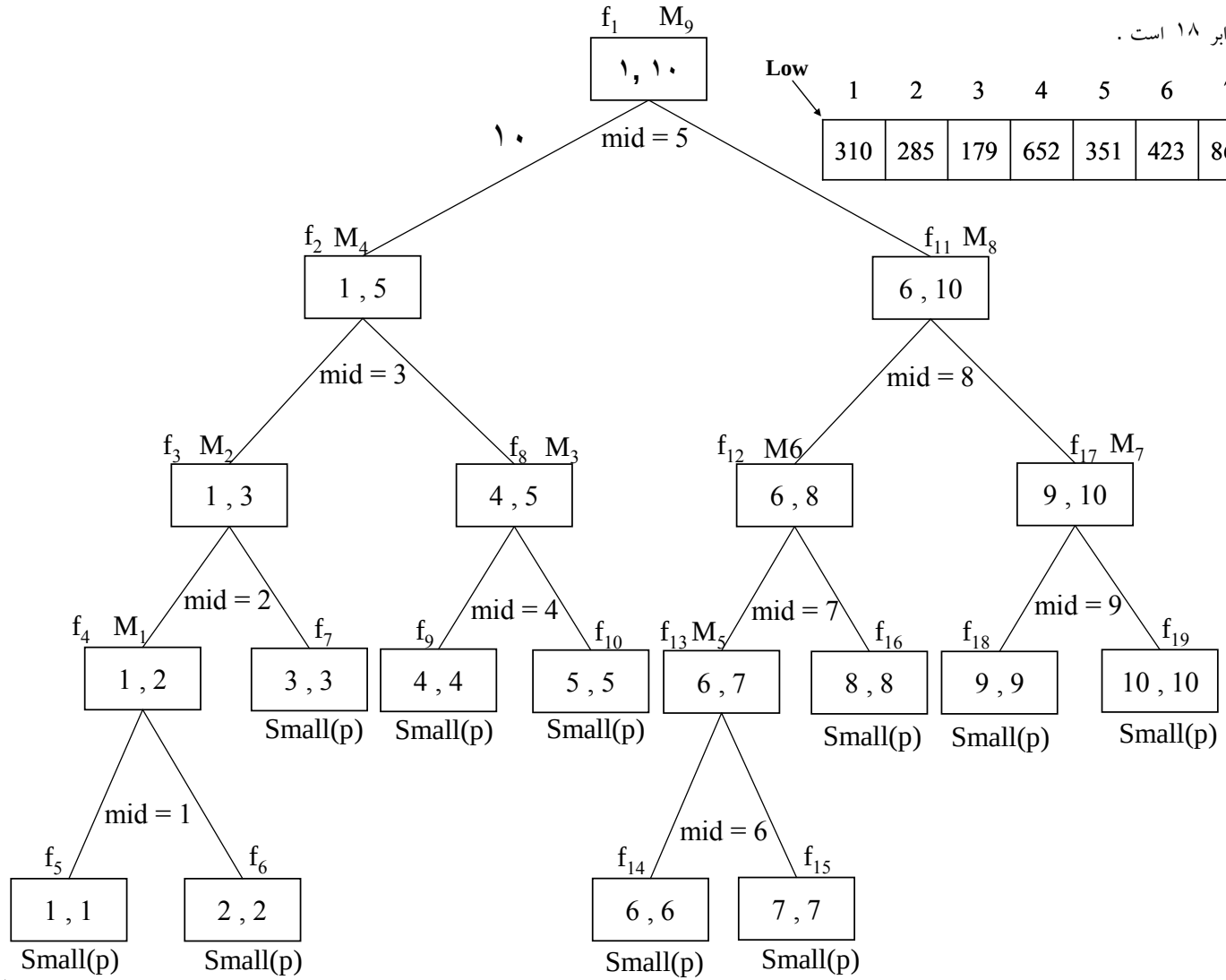
310 285 179 652 351 423 861 254 450 520

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



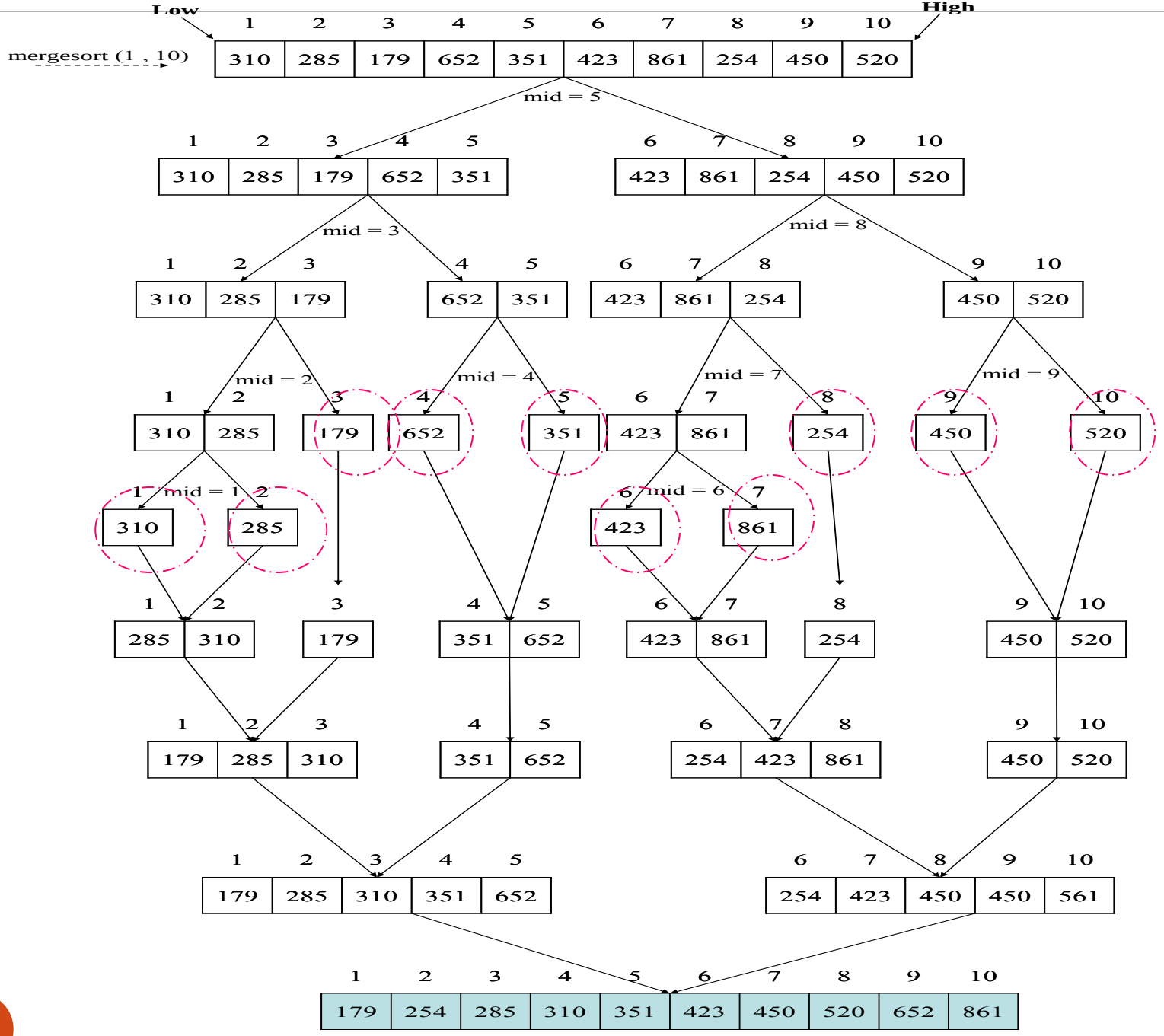
Merge sort

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



☜تعداد فراخوانی بازگشتی برابر ۱۸ است .

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

الگوریتم ادغام (Merge)

```
void  merge۲ (int low, int mid, int high)
 {
    index i, j , k;
    keytype U [ low..high];
    i = low; j = mid +۱ ; k = low;
    while ( i <= mid && j <=  high)  {
          if ( arr [i]  < arr [j] )  {
              U [k] = arr [i];
              i++ ;
           }
 else  {
          U [k] = arr [j];
          j ++;
       }
       k ++;
    }
    if ( i >  mid )
         move arr [j]  through arr [high] to U [k]  through U [high];
    else
       move arr [i]  through arr [mid] to U [k]  through U [high];
       move U [low] through U [high] to arr [low] through arr [high];
 }
```

حافظه مصرفی اضافی:O(n)

الگوریتم Merge به صورت درجا

```
void merge( int arr[ ], int low, int mid, int high )
{
  int i, j, k, t;
  j = low;
  for( i = mid + ۱ ; i <= high ; i++ )
  {
    while( arr[ j ] <= arr[ i ] && j < i )
    {
      j++;
    }
    if( j == i )  {  break; }
    t = arr[ i ];
    for( k = i ; k > j ; k -- )
    {
      arr[ k ] = arr[ k - ۱ ];
    }
    arr[ j ] = t;
  }
}
```

حافظه اضافی: O(۱)

پیچیدگی الگوریتم merge sort

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

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

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)

merge sort

- پیچیدگی محاسباتی الگوریتم ادغام در بدترین حالت و حالت میانگین برابر با $O(n \log n)$ است.
- این الگوریتم نیازمند فضای اضافی متناسب با تعداد رکوردهایی که باید مرتب شوند می باشد.
- چنانچه در مرحله ادغام از روشی استفاده شود که به میزان حافظه از $O(1)$ نیاز داشته باشد نیاز حافظه کل الگوریتم به $O(1)$ کاهش می یابد ولی در این صورت الگوریتم حاصل به صورت قابل توجهی کندتر از نسخه اصلی خواهد بود.
- مرتب سازی ادغام پایدار است.
- زمان مرتب سازی ادغام روی یک لیست مرتب $O(n \log n)$ است.

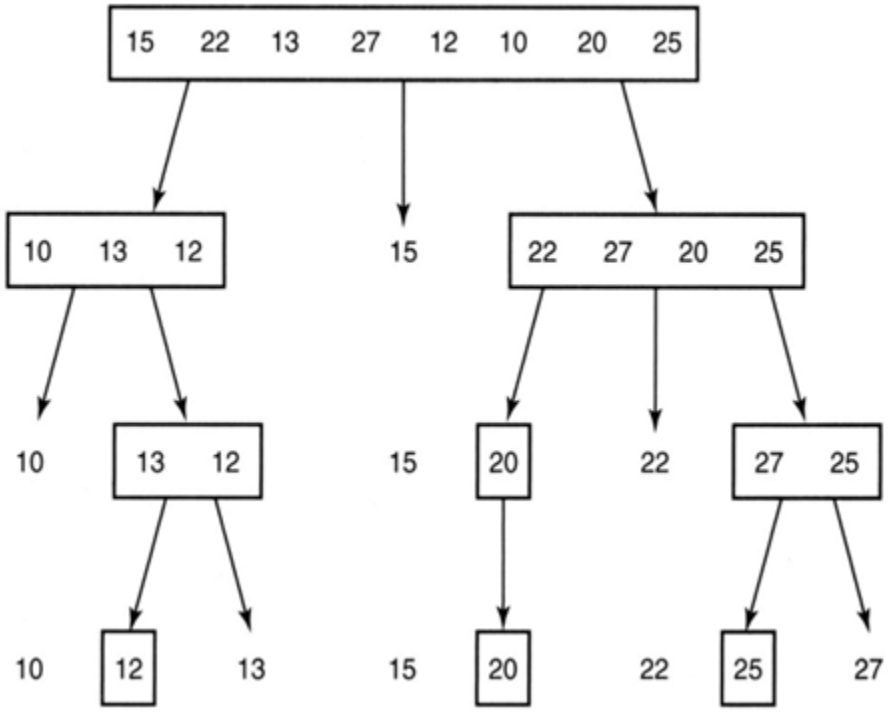
مرتب سازی سریع (quick sort) به روش تقسیم و حل

- مراحل الگوریتم:
  - انتخاب عنصر محوری (pivot) : می تواند هر عنصری باشد (مثلاً عنصر اول)
  - جابجایی عناصر آرایه و تقسیم آن به دو بخش طوری که عناصر کوچکتر از عنصر محوری در سمت چپ و عناصر بزرگتر از آن در سمت راست آن قرار بگیرند.

  - مرتب سازی هر بخش به صورت بازگشتی به روش مرتب سازی سریع
  - مرتب سازی سریع، به طور بازگشتی فراخوانی می شود تا هر یک از دوآرایه را مرتب کند، آن ها نیز افراز می شوند و این روال ادامه می یابد تا به آرایه ای با یک عنصربرسیم. چنین آرایه ای ذاتاً مرتب است.

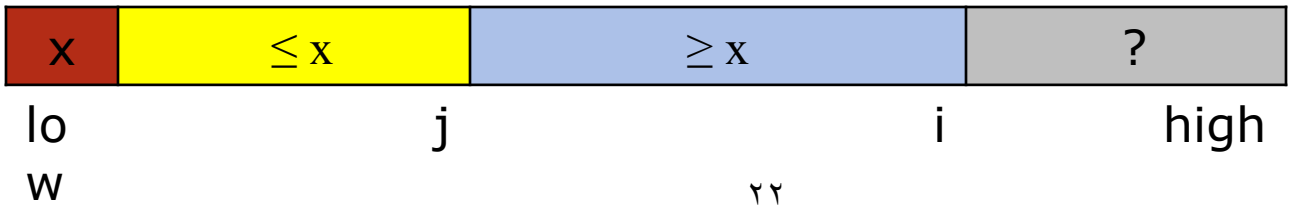
Quick sort

```
1.  void quickSort(index low, index high)
2.  {
3.      index pivotpoint;
4.      if (high > low)
5.      {
6.          partition(low, high, pivotpoint);
7.          quickSort(low, pivotpoint - 1);
8.          quickSort(pivotpoint + 1, high);
9.      }
10. }
```

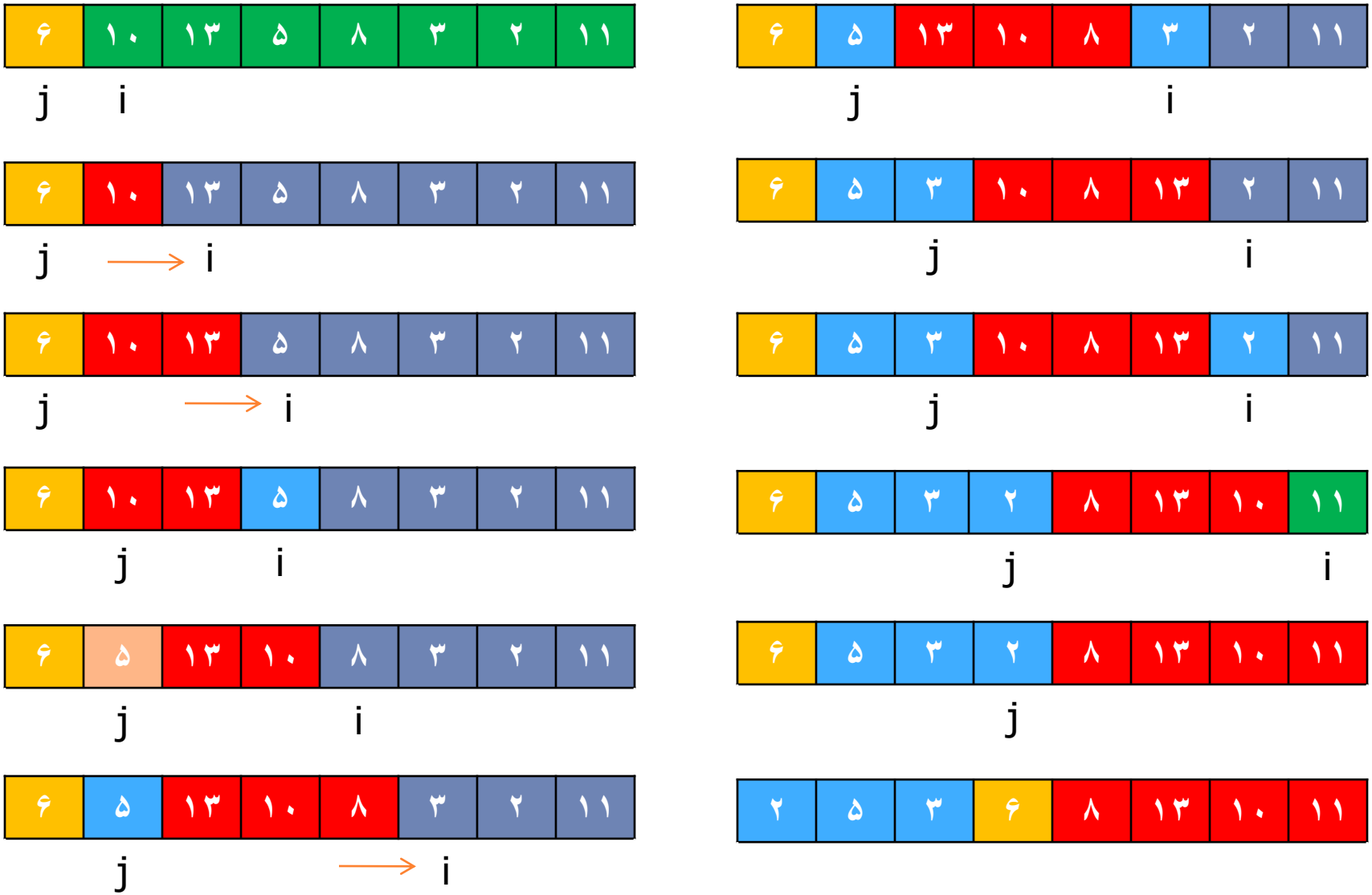


افراز (partitioning) در Quick sort

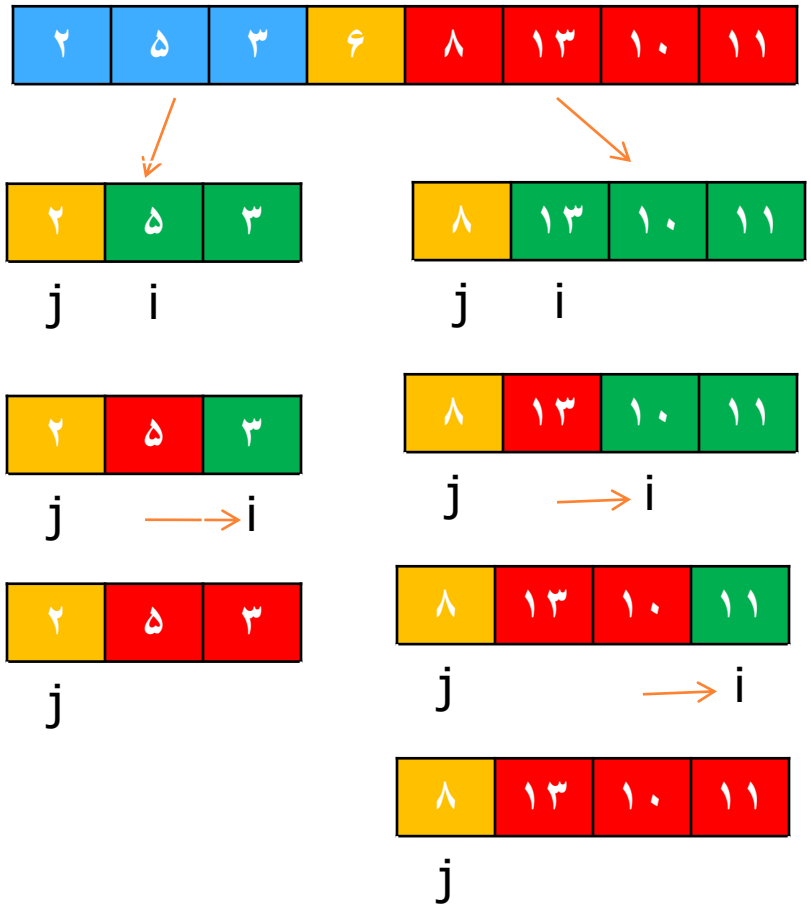
```
1.  void partition(index low, index high, index& pivotpoint)
2.  {
3.      index i, j;
4.      keytype pivotitem;
5.      pivotitem = S[low];  // Choose first item for pivotitem.
6.      j = low;
7.      for (i = low + 1; i <= high; i++)
8.          if (S[i] < pivotitem)
9.          {
10.             j++;
11.             exchange S[i] and S[j];
12.         }
13.     pivotpoint = j;
14.     exchange S[low] and S[pivotpoint]; // Put pivotitem at pivotpoint.
15. }
```



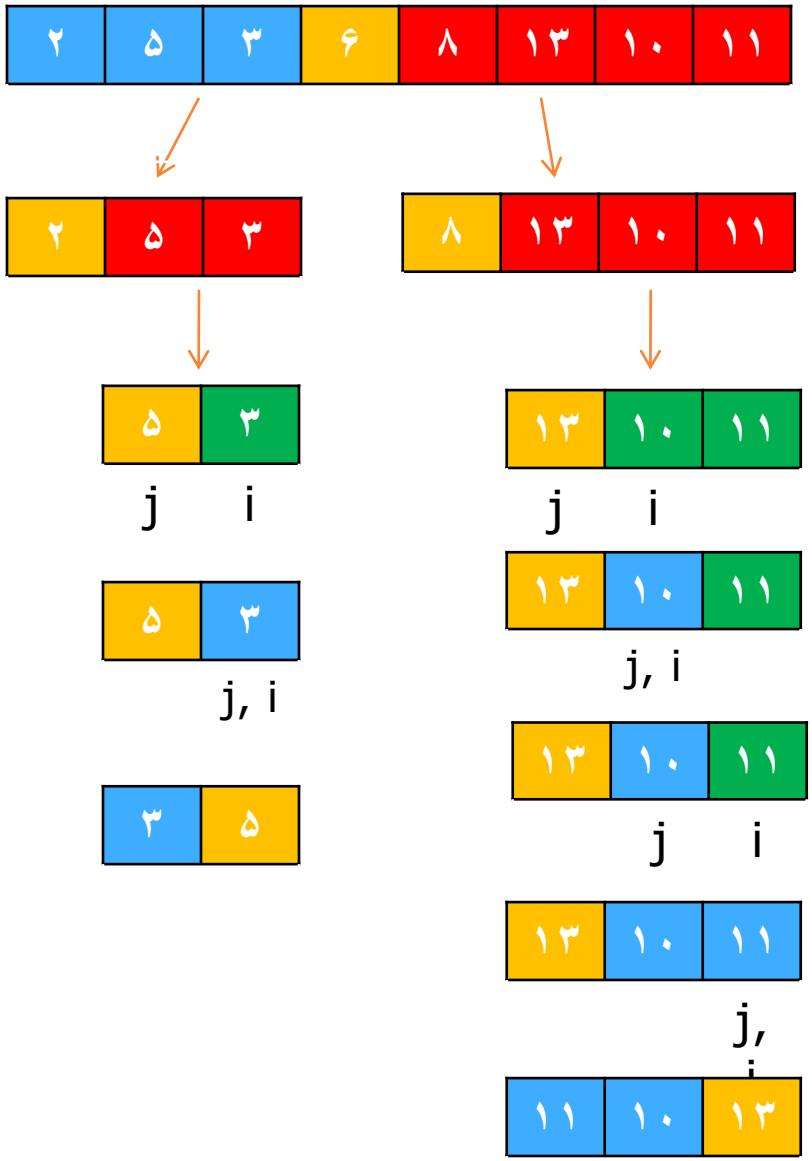
partitioning (سطح صفر)



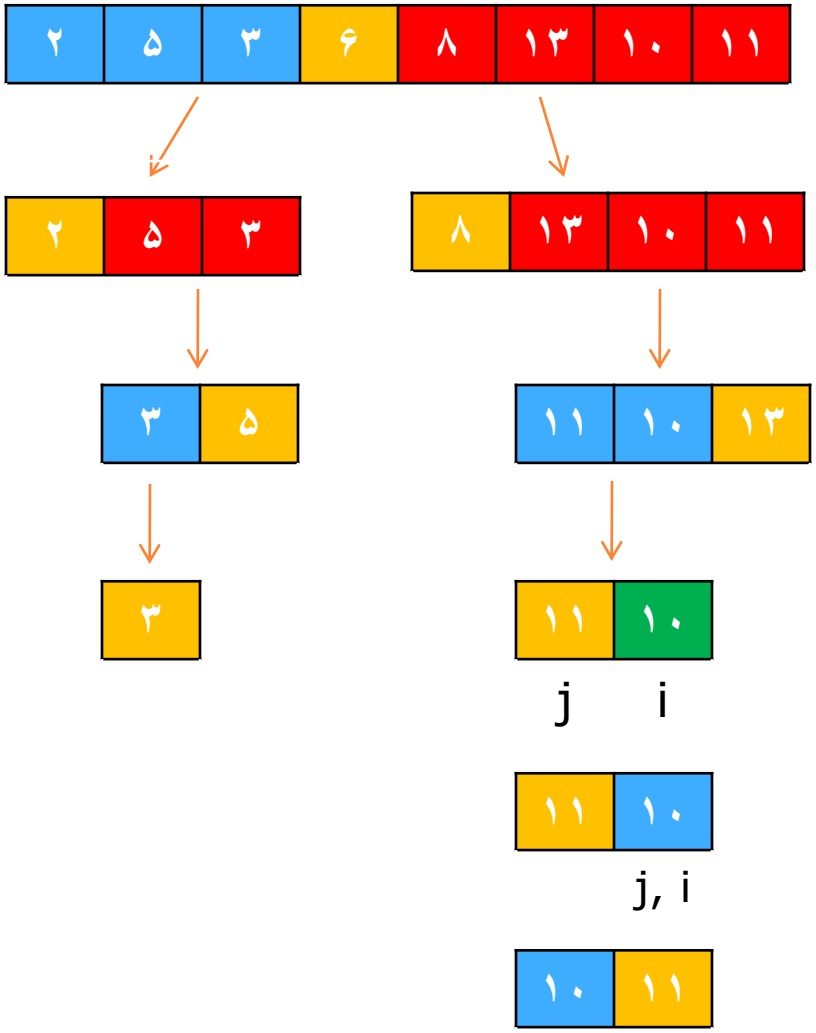
Partitioning (سطح یک)



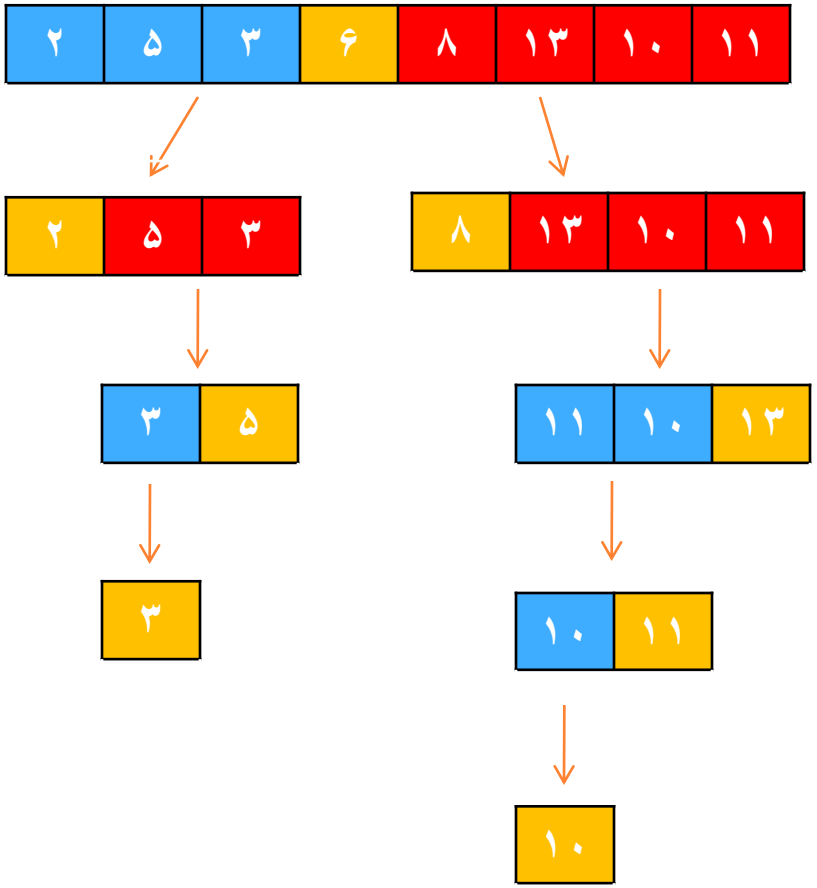
Partitioning (سطح دو)



Partitioning (سطح سه)



Partitioning (سطح چهار)



افراز (partition) در Quick sort
الگوریتم ۲

الگوریتم partition ()

```
Alg partition (p , q + 1)
{
  V = a[p] ;   i = p ;    j = q + 1 ;
  repeat
  {
    repeat
       i = i + 1 ;
    until (a[i] ≥ V );

    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 ;
}
```

V : عنصر محور

i : اندیس ابتدای لیست

j : اندیس انتهای لیست

حلقه با اندیس i عناصر سمت چپ وحلقه با 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

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
  }
}
```

پیچیدگی زمانی الگوریتم مرتب سازی سریع(بدترین حالت)

- زمان اجرا به متوازن بودن یا نبودن افراز بستگی دارد.
- بدترین حالت
  - زمانی است که آرایه از قبل مرتب باشد.
  - همواره آرایه اصلی به یک آرایه خالی و یک آرایه با n-۱ عنصر افراز می شود.
- عمل اصلی: مقایسه عناصر با عنصر محوری
- اندازه ورودی: تعداد عناصر آرایه

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

- زمان مرتب سازی آرایه سمت چپ
- زمان مرتب سازی آرایه سمت راست
- زمان انجام عمل افراز

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

- حل:

$$w(n) = \frac{n(n-1)}{2} \in \Theta(n^2)$$

پیچیدگی زمانی الگوریتم مرتب سازی سریع(بهترین حالت)

- بهترین حالت
  - زمانی است که عنصر محوری لیست را به دو زیرلیست مساوی افراز کند

اندازه ورودی: $n = high - how + ۱$ ، تعداد عناصرموجود در زیر آرایه.

در بهترین حالت داریم:

$$T(n) = n + ۲T(n/۲)$$

$$B(n) = n \log_۲ n$$

پیچیدگی زمانی الگوریتم مرتب سازی سریع (حالت میانگین)

- احتمال اینکه هر یک از عناصر به عنوان محور انتخاب شود را مساوی و $1/n$ در نظر می گیریم.

$$A(n) = \sum_{i=1}^{n} \frac{1}{n} \left[A(i-1) + A(n-i)\right] + n - 1$$

$$nT(n) = n(n-1) + 2(T(0) + \ldots + T(n-1))$$

$$T(n) \approx (n+1)2\,\mathrm{Ln}\ n = (n+1)2(\mathrm{Ln}\ 2)(\log n) \approx 1/38(n+1)\log n \rightarrow T(n) \in \theta(n \log n)$$

- حل:

$$A(n) \approx 1.38(n+1)\lg n \in \Theta(n \lg n)$$

پیچیدگی زمانی الگوریتم افراز(partition)

- در تمام حالات T(n)
  - عمل اصلی: تعداد مقایسه
  - اندازه ورودی: تعداد عناصر آرایه

$$n = high - low + ۱$$

- عنصر اول با بقیه عناصر مقایسه می شود.

$$T(n) = n - ۱$$

تمرین

- تمرین۲-۱: زمان اجرای الگوریتم مرتب سازی سریع زمانی که تمام عناصر برابر باشند را تحلیل کنید.
- تمرین۲-۲: با استفاده از مرتب سازی ادغام و مرتب سازی سریع لیست زیر را مرتب کنید. برای هر الگوریتم مراحل انجام و درخت فراخوانی های بازگشتی را ترسیم نمائید.

۱۵ ۲۰ ۱۰ ۲۵ ۸ ۳۰ ۲ ۲۸

- تمرین ۲-۳: با استفاده از تابع partition و بدون نیاز به مرتب سازی، الگوریتمی برای انتخاب k مین عنصر کوچک یک آرایه ارائه دهید. زمان اجرای آن را به دست آورید.
- تمرین ۲-۴: نتیجه گیری خود را از حل تمرین ۱-۷ بیان کنید.

الگوریتم های بازگشتی

- در بعضی از الگوریتم های بازگشتی، هیچ عملی پس از فراخوانی بازگشتی انجام نمی گردد.
  - مثال:
    - جست وجوی دودویی
    - مرتب سازی سریع

- در این دسته از الگوریتم ها ایجاد یک نسخه تکراری کار آسانی است.
  - مزایای تبدیل به الگوریتم تکراری:
    - حذف پشته در فراخوانی ها و صرفه جویی درحافظه
    - الگوریتم های تکراری سریع تر از بازگشتی ها هستند.

ضرب ماتریسها به روش استراسن

```
void matrixmult (int n const number A [ ] [ ],
                 const number B [ ] [ ],  number C [ ] [ ])
{
  index i , j, k;
  for ( i = ١; I <= n ; i++)
   for (i = ١; j <= n ; j++)}
      C [i] [j] = ٠;
      for (k = ١ ; k <= n ; k++)
        C [i][j] = C[i] [j] + A [i][k] * B [k][j]
  }
}
```

- ضرب ماتریس ها به روش استاندارد:
  - تعداد ضربها $T(n) = n^3$
  - تعداد جمع و تفریق ها

$$T(n) = n^3 - n^2$$

- ضرب ماتریس ها به روش استراسن
  - توسط Strassen در سال ۱۹۶۹ ارائه شد.
  - پیچیدگی آن بهتر از درجه سوم می باشد.
  - تعیین حاصلضرب دو ماتریس n× n که در آن n توانی از ۲ است.

ضرب ماتریسها به روش استراسن

- برای محاسبه $A_{2*2} * B_{2*2}$

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

- استراسن معین کرد که اگر فرض کنیم

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

در روش استراسن برای ضرب دو ماتریس ۲*۲ به ۷ عمل ضرب و ۱۸ عمل جمع و تفریق نیاز است.

در روش عادی به ۸ عمل ضرب و ۴ عمل جمع و تفریق نیاز است

- آنگاه حاصل ضرب C به صورت زیر بدست می آید

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

ضرب ماتریسها به روش استراسن

❖ مراحل الگوریتم

- تقسیم: ابعاد ماتریس ها توانی از ۲ باید باشد. هر یک از ماتریس های A و B را به چهار زیرماتریس با ابعاد n/۲ تقسیم می کنیم.
- حل: ضرب هر کدام از بخش های مورد نیاز با فراخوانی بازگشتی انجام می شود.
- ترکیب: پاسخ های به دست آمده را مطابق با روابط استراسن ترکیب نمایید.

$$A_{11} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1,n/2} \\ a_{21} & a_{22} & \cdots & a_{2,n/2} \\ & & \vdots & \\ a_{n/2,1} & a_{n/2,2} & & a_{n/2,n/2} \end{bmatrix}$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C_{21} = M_2 + M_4$$
$$M_2 = (A_{21} + A_{22})B_{11}$$

ضرب ماتریسها به روش استراسن

مثال:

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

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

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

✓ هنگامی که ماتریس ها به اندازه کافی کوچک باشند آنها را به روش استاندارد در هم ضرب می کنیم.

$$M_1=\begin{bmatrix}3&5\\11&13\end{bmatrix}\times\begin{bmatrix}17&10\\7&9\end{bmatrix}=\begin{bmatrix}3\times17+5\times7&3\times10+5\times9\\11\times17+13\times7&11\times10+13\times9\end{bmatrix}=\begin{bmatrix}86&75\\278&227\end{bmatrix}$$

✓ M_2 تا M_7 نیز به همین شیوه محاسبه می شوند و مقادیر c_{11} ، c_{12} ، c_{21} و c_{22} به دست می آیند. با ترکیب آنها ماتریس نهایی حاصل می شود.

الگوریتم ضرب ماتریسها به روش استراسن

```
1.  void strassen(int n,
                  n × n_matrix A,
                  n × n_matrix B,
                  n × n_matrix& C)
2.  {
3.      if (n<=threshold)
4.          compute C = A × B using the standard algorithm;
5.      else
6.      {
7.          partition A into four submatrices A11, A12, A21, A22;
8.          partition B into four submatrices B11, B12, B21, B22;
9.          compute C = A × B using Strassen's method;
10.           // example recursive call; strassen (n/2, A11 + A22, B11 + B22, M1)
11.     }
12. }
```

threshold (حد آستانه): نقطه‌ای است که در آن احساس می‌شود استفاده از الگوریتم ضرب استاندارد بهتر از فراخوانی بازگشتی روال strassen باشد.

الگوریتم ضرب ماتریسها به روش استراسن– کامل

```
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 الگوریتم استراسن

- اندازه ورودی: n، تعداد سطر و ستون ماتریس ها
- **عمل اصلی: یک ضرب ساده**

$$\begin{cases} T(n) = 7T(\frac{n}{2}) & for \quad n>1, \quad n \quad a \quad power \quad of \quad 2 \\ T(1) = 1 \end{cases}$$

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

جمع و تفریق عناصر دو ماتریس n/۲ * n/۲

- **عمل اصلی: جمع و تفریق**

$$\begin{cases} T(n) = 7T(\frac{n}{2}) + 18(\frac{n}{2})^2 & for \quad n>1, \quad n \quad a \quad power \quad of \quad 2 \\ T(1) = 0 \end{cases}$$

$$T(n) = \Theta(n^{2.81})$$

حل: با استفاده از قضیه اصلی

مقایسه روش استراسن با روش استاندارد

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	$n^{2.81}$	$6n^{2.81} - 6n^2$

ضرب ماتریسها به روش استراسن

- اگر اندازه ماتریسها توانی از ۲ نباشد
  - به تعداد کافی سطر و ستون صفر به آنها اضافه می کنیم.
- ضرب ماتریسها نیازمند الگوریتمی با پیچیدگی زمانی حداقل از درجه دوم دارد:
  - هنوز کسی نتوانسته است برای ضرب ماتریسها الگوریتمی درجه دو ارائه کند.
  - کسی هم نتوانسته است اثبات کند که ایجاد چنین الگوریتمی غیر ممکن است.

ضرب ماتریسها به روش استراسن

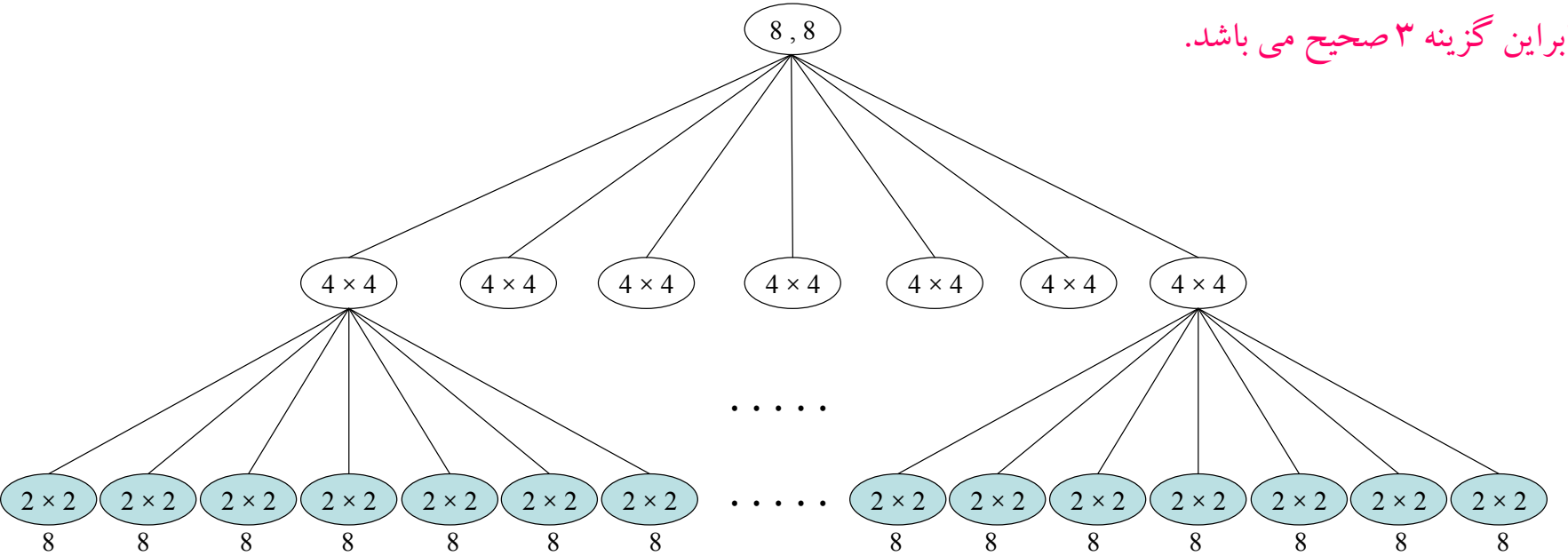
سوال کنکور(سراسری۸۵) در ضرب ماتریس‌ها به روش استراسن اگر مسأله کوچک، ضرب ماتریس ۲ × ۲ باشد، برای ضرب دو ماتریس ۸ × ۸ چند ضرب عددی صورت می‌پذیرد؟

۱) ۵۷ ۲) ۳۴۳ ۳) ۳۹۲ ٤) ۵۱۲

جواب : اگر فرض کنیم ضرب ماتریس ۲ × ۲ 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.



تعیین مقادیر آستانه

- منظور از مقدار آستانه ی n برای یک الگوریتم بازگشتی، اندازه نمونه ایست که به ازای تمام نمونه های کوچکتر از آن، بهتر است به جای فراخوانی بازگشتی، از یک الگوریتم دیگر استفاده شود.
  - از لحاظ زمان اجرا
- در تعیین مقدار آستانه، بدترین حالت الگوریتم را مورد بررسی قرار می دهیم (در واقع سعی داریم رفتار بدترین حالت را بهینه نماییم).

تعیین مقادیر آستانه (مثال۱)

- فرض کنید برای مرتب سازی ادغامی، t را به عنوان مقدار آستانه داشته باشیم. در اینصورت برای نمونه های کوچکتر از t، از مرتب سازی تعویضی استفاده می کنیم. فرض کنید زمان تقسیم و ادغام زیرآرایه به طول n، ۳۲n میکرو ثانیه باشد و لذا:

$$W(n)=\begin{cases}\dfrac{n(n-1)}{2}\mu s & if\ (n\le t)\\ W(\left\lfloor\dfrac{n}{2}\right\rfloor)+W(\left\lceil\dfrac{n}{2}\right\rceil)+32n\mu s & if\ (n>t)\end{cases}$$

- مقدار آستانه، مقداریست که در آن زمان اجرای هر دو الگوریتم یکسان باشد.

$$t=\begin{cases}128 & if\ (t\%2=0)\\ 128.008 & if\ (t\%2=1)\end{cases}$$

تعیین مقادیر آستانه (مثال۲)

- فرض کنید برای انجام کاری در کامپیوتر از دو الگوریتم بازگشتی و غیر بازگشتی به صورت ترکیبی با زمانهای اجرای زیر استفاده می شود. مقدار آستانه t را بدست آورید.

N	زمان غیربازگشتی	زمان بازگشتی
۶۲	۳۸۴۴	۳۸۷۵
۶۳	۳۹۶۹	۴۰۸۰
۶۴	۴۰۹۶	۴۰۹۶
۶۵	۴۲۲۵	۴۳۰۷
۶۸	۴۶۲۴	۴۵۵۶
۶۹	۴۷۶۱	۴۷۷۹
۷۰	۴۹۰۰	۴۷۹۵
۷۱	۵۰۴۱	۵۰۲۴

$$W(n)=\begin{cases} n^2\,\mu s & if\ (n\le t) \\ 3W(\left\lceil \frac{n}{2} \right\rceil)+16n\mu s & if\ (n>t) \end{cases}$$

حل:

$$t=\begin{cases} 64 & if\ (t\%2=0) \\ 70.04 & if\ (t\%2=1) \end{cases}$$

مسأله پیدا کردن ماکزیمم و مینیمم به روش غیربازگشتی (تکراری)

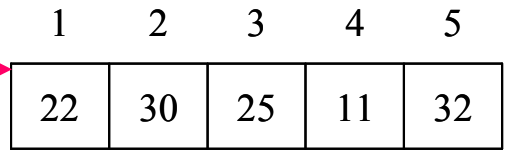
```
void maxmin (a , n , max , min)
{
max = min = a[۱] ;
for (i = ۲ ; i < = n ; i + + )
{
if (a[i] > max) then max = a[i] ;
if (a[i] < min) then min = a[i] ;
}
}
```

☜با گذاشتن یک else قبل از if دوم الگوریتم کمی بهتر می‌شود.

```
void maxmin (a , n , max , min)
{
max = min = a[۱] ;
for (i = ۲ ; i < = n ; i + + )
{
if (a[i] > max) then max = a[i] ;
else if (a[i] < min) then min = a[i] ;
}
}
```

max و min لیست زیر را با توجه به الگوریتم فوق بدست آورید .

۲۲ ۳۰ ۲۵ ۱۱ ۳۲

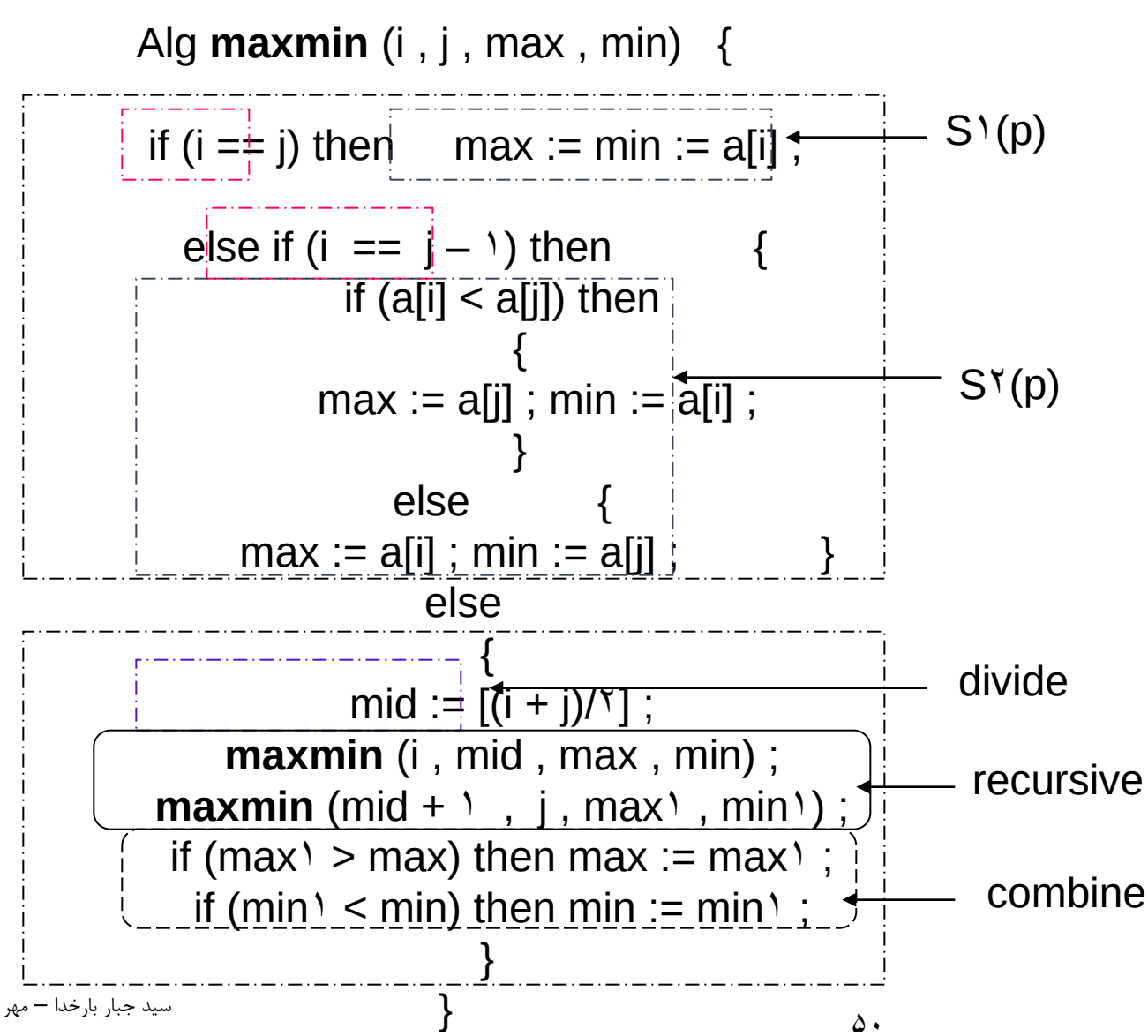


$$max = min = 22 = a[1]$$

تعداد مقایسه ها برابر با
۸ = (n – ۱) ۲ می باشد.

i	مقایسه انجام شده	نتیجه مقایسه	مقدار Max جدید	مقدار MIN جدید
1	-	-	**22**	**22**
2	30 > 22	T	**30**	22
	30 < 22	F	30	22
3	25 > 30	F	30	22
	25 < 22	F	30	22
4	11 > 30	F	30	22
	11 <22	T	30	**11**
5	32 > 30	T	**32**	11
	32 < 11	F	32	11

الگوریتم تقسیم و حل پیدا کردن ماکزیمم و مینیمم

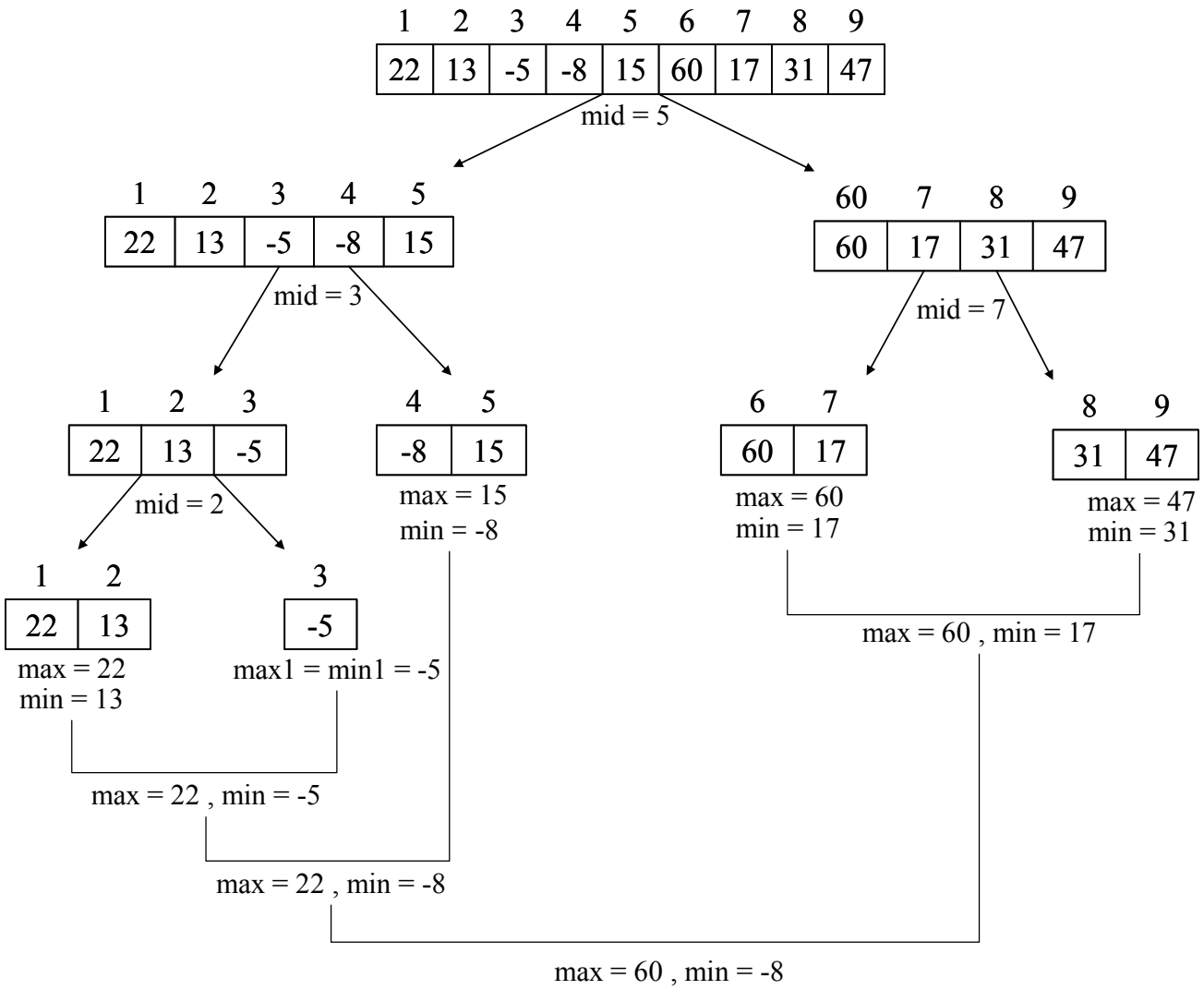


Small۱(p) → i=j

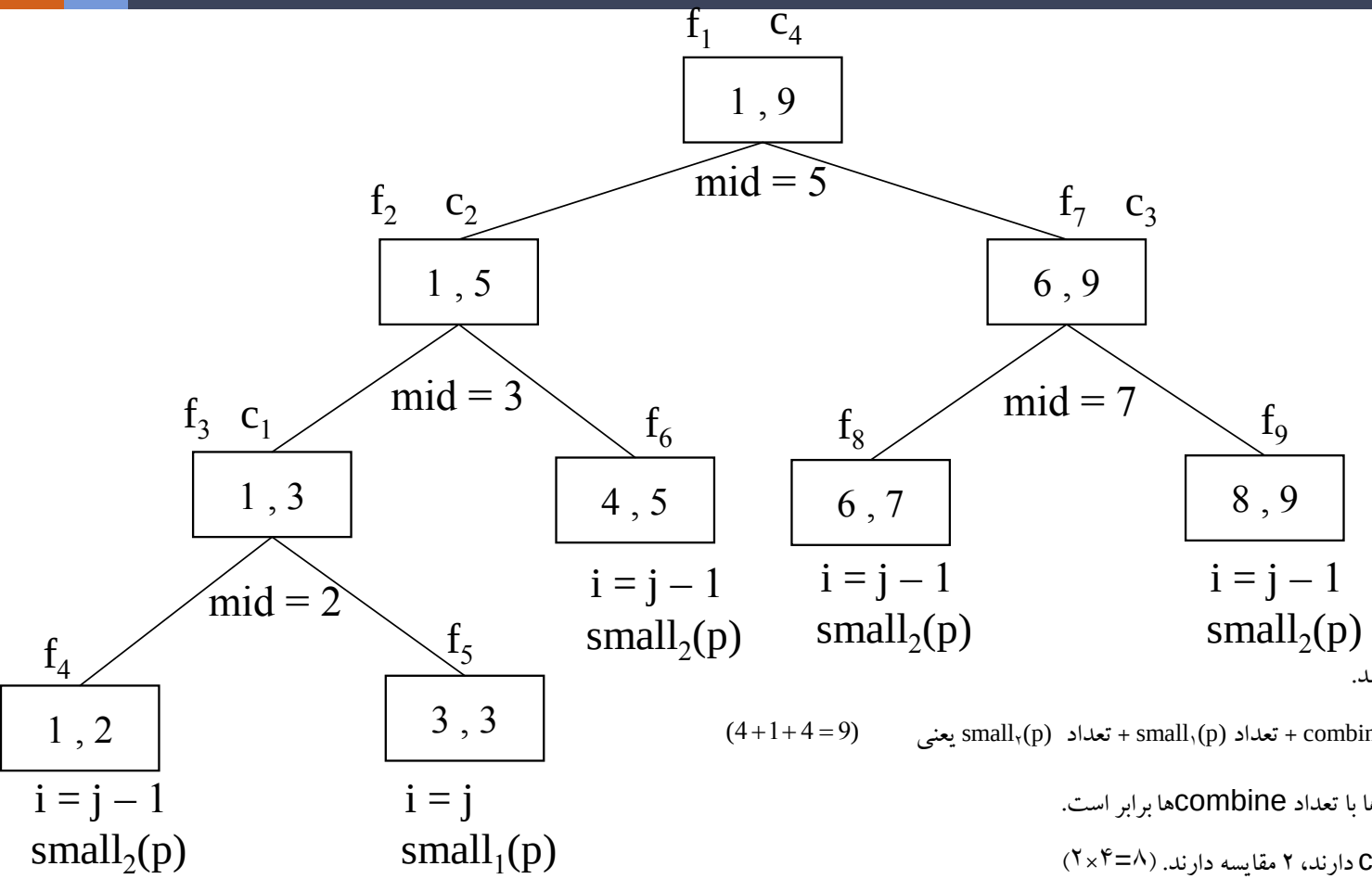
Small۲(p) → i=j-۱

min و maxلیست زیر را با توجه به الگوریتم بازگشتی maxmin به دست آورید. درخت بازگشتی و درخت فراخوانی بازگشتی آنرا رسم نمایید.

۲۲ ۱۳ -۵ -۸ ۱۵ ۶۰ ۱۷ ۳۱ ۴۷



بدین ترتیب که برای هر فراخوانی یک گره ایجاد می کنیم و درخت فراخوانی زیر به دست می آید



f_i: ترتیب فراخوانی‌ها

C_i : ترتیب combineها

☜تعداد کل فراخوانی‌ها برابر با ۹ می باشد.

☜تعداد کل فراخوانی برابر است با تعداد combine + تعداد $small_1(p)$ + تعداد $small_2(p)$ یعنی $(4+1+4=9)$

☜در الگوریتم فوق تعداد mid ها با تعداد combineها برابر است.

☜فراخوانی‌هایی که combine دارند، ۲ مقایسه دارند. (۸=۴×۲)

☜فراخوانی‌های $small_2(p)$ یک مقایسه دارند. (۴=۱×۴)

☜در فراخوانی‌های $small_1(p)$ مقایسه‌ای انجام نمی شود. (۰=۱×۰)

☜تعداد کل مقایسه‌ها در الگوریتم فوق: $2\times4+1\times4+0\times1=12$

پیچیدگی الگوریتم بازگشتی max-min

$$T(n) = \begin{cases} 0 & n = 1 \\ 1 & n = 2 \\ 2 + T(\frac{n}{2}) + T(\frac{n}{2}) & else \end{cases}$$

پیچیدگی الگوریتم‌های بازگشتی و غیر بازگشتی maxmin

نام الگوریتم	روش حل مساله	بهترین حالت	بدترین حالت	میانگین
max-min	غیر بازگشتی	$2(n-1)$	$2(n-1)$	$2(n-1)$
max-min	غیر بازگشتی بهینه	$(n-1)$	$2(n-1)$	کمتر از $2(n-1)$
max-min	بازگشتی (تقسیم و حل)	$\frac{3n}{2}-2$	$\frac{3n}{2}-2$	$\frac{3n}{2}-2$

در الگوریتم غیر بازگشتی بهترین حالت، زمانی است که لیست صعودی مرتب باشد
وبدترین حالت زمانی است که لیست نزولی مرتب باشد .

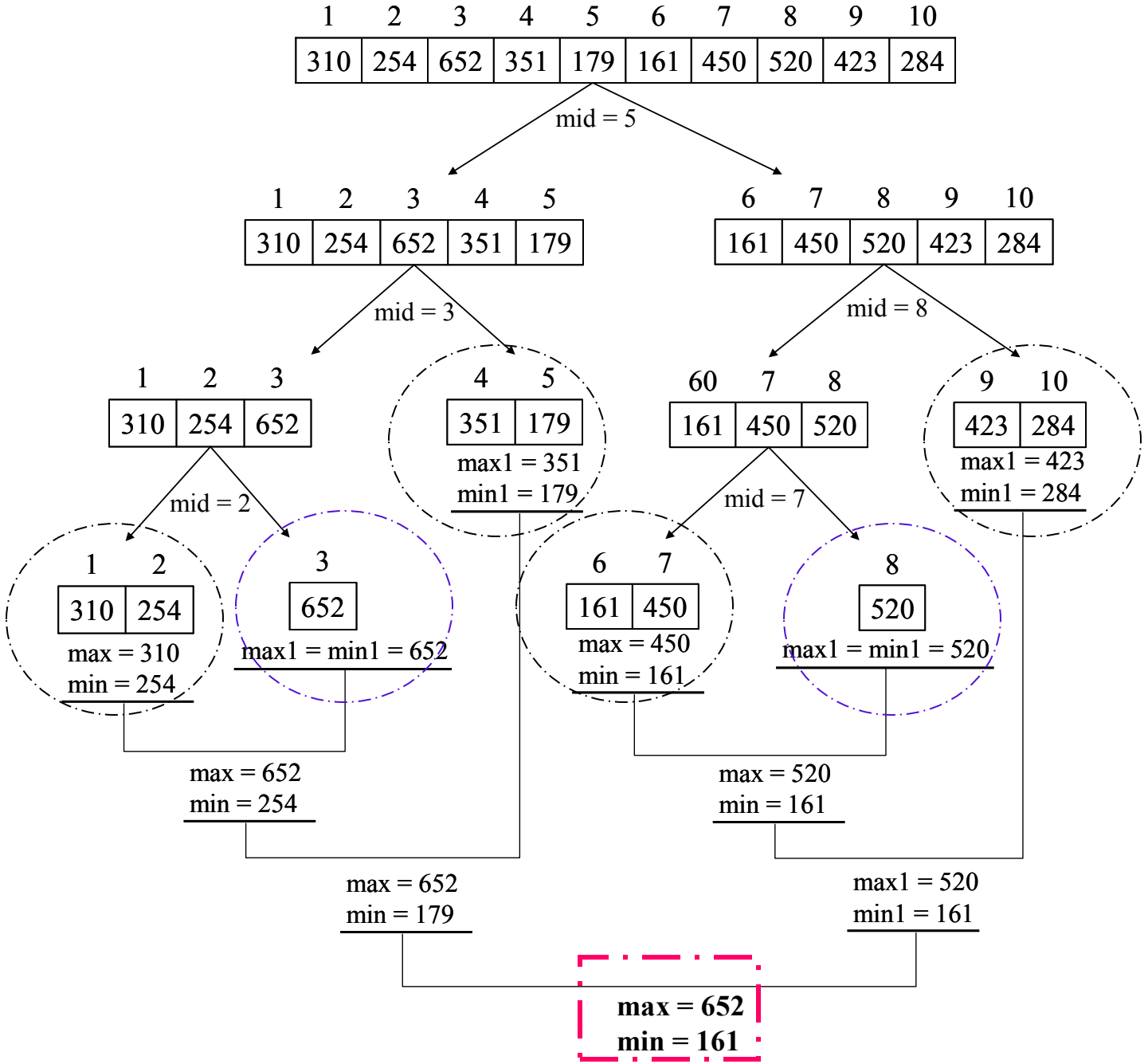
مثال

- الگوریتم بازگشتی max-min را در نظر بگیرید.

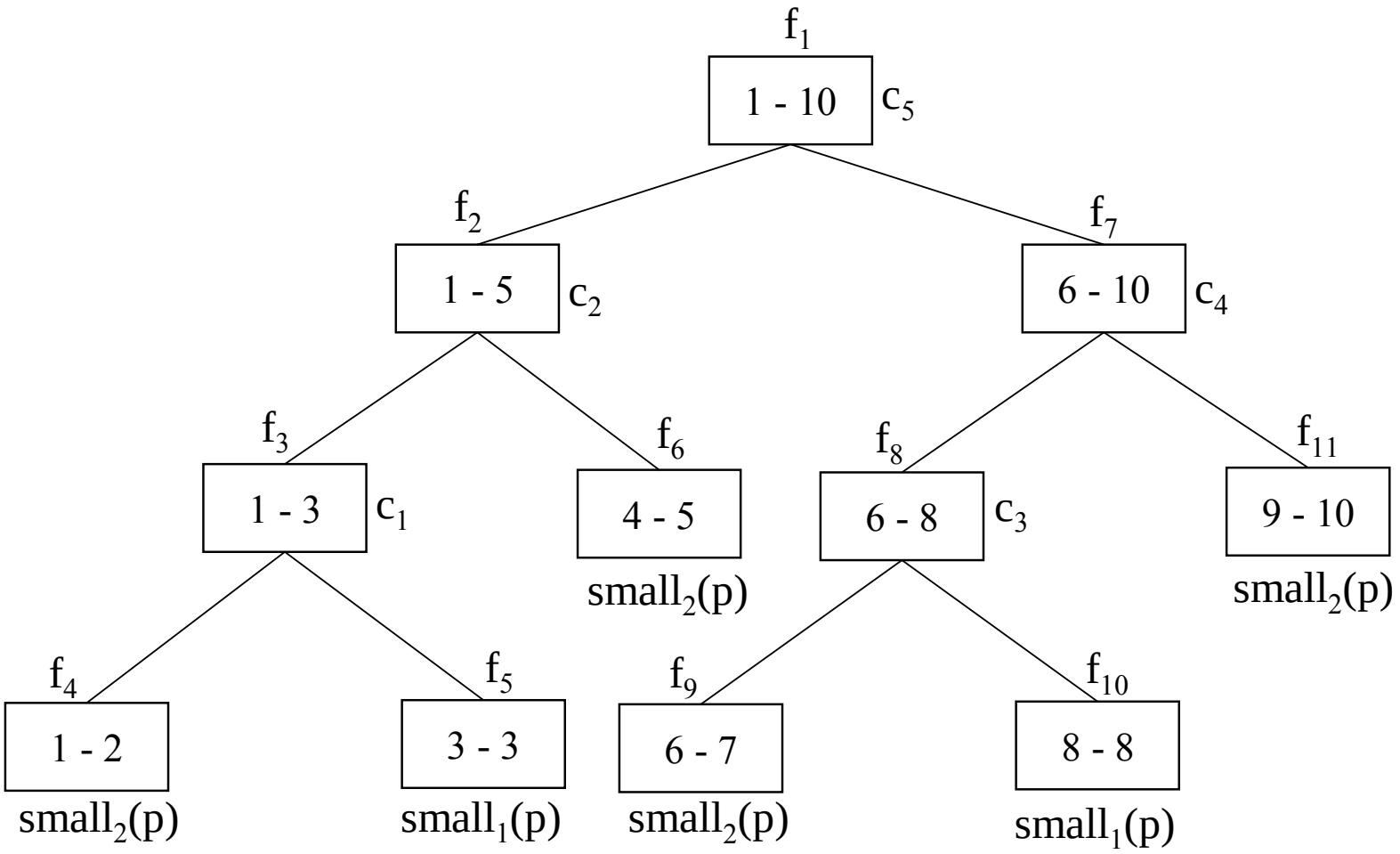
الف) برای لیست اعداد زیر چه تعداد مقایسه لازم است تا ماکزیمم و مینیمم لیست پیدا شود.

۲۸۴ ۴۲۳ ۵۲۰ ۴۵۰ ۱۶۱ ۱۷۹ ۳۵۱ ۶۵۲ ۲۵۴ ۳۱۰

ب) نمودار فراخوانی را رسم کرده و کلیه مقایسه‌ها را بنویسید.


1 2 3 4 5 6 7 8 9 10
310 254 652 351 179 161 450 520 423 284
mid = 5
1 2 3 4 5
310 254 652 351 179
6 7 8 9 10
161 450 520 423 284
mid = 3
mid = 8
1 2 3
310 254 652
4 5
351 179
max1 = 351
min1 = 179
60 7 8
161 450 520
9 10
423 284
max1 = 423
min1 = 284
mid = 2
mid = 7
1 2
310 254
max = 310
min = 254
3
652
max1 = min1 = 652
6 7
161 450
max = 450
min = 161
8
520
max1 = min1 = 520
max = 652
min = 254
max = 520
min = 161
max = 652
min = 179
max1 = 520
min1 = 161
max = 652
min = 161

درخت فراخوانی تابع max min



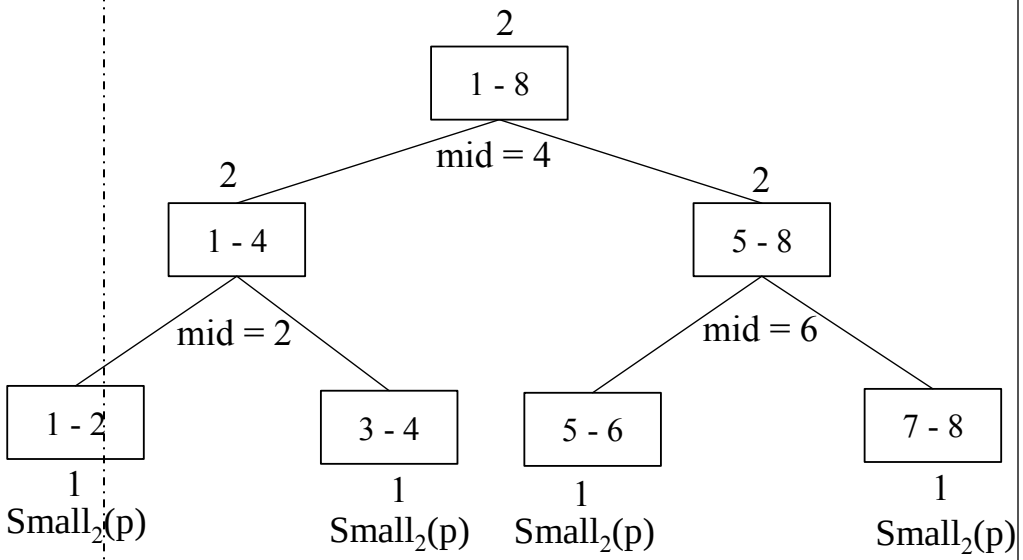
۵- الگوریتم زیر برای پیدا کردن عناصر ماکزیمم و مینیمم یک آرایه به n عنصر پیشنهاد شده است: (مهندسی کامپیوتر - دولتی ۷۵)

تعداد مقایسه‌های این الگوریتم (فقط مقایسه‌هایی که زیر آنها خط کشیده شده است.) برای $n = ۸$ چقدر است؟

۱) ۱۲ ۲) ۱۳ ۳) ۱۰ ۴) ۱۱

```
Procedure minmax (i , j : integer ; var min , max : real)
                        ;
              Var k : integer ;
        min۱ , min۲ , max۱ , max۲ : real ;
                    Begin
                if i + ۱ = j then
                  if A[i] > A[j] then
                          begin
              min := A[j] ;        max := A[i] ;
                           end
                          else
                           begin
              min := A[i] ;        max := A[j] ;
                           end
                        else
                          begin
                    K := (i + j) div ۲ ;
              minmax  (i , k , min۱ , max۱) ;
            minmax (k + ۱ , j , min۲ , max۲) ;
          if   (min۱ < min۲ )   then min := min۱ ;
                    else min := min۲ ;
        if  (max۱ < max۲)     then max := max۲ ;
                    else max := max۱ ;
                           end
                         End
```

جواب :



۶ + ۴ = ۱۰

بنابراین گزینه ۳ صحیح می‌باشد

ضرب اعداد صحیح بزرگ (دو عدد n رقمی)

- الگوریتم عادی ضرب دو عدد n رقمی
  - زمان n^2
- الگوریتم تقسیم و غلبه ضرب اعداد بزرگ

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

$$u = x \times 10^m + y$$
$$v = w \times 10^m + z$$

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

$$\underbrace{567{,}832}_{\text{۶ رقم}} = \underbrace{567}_{\text{۳ رقم}} \times 10^3 + \underbrace{832}_{\text{۳ رقم}}$$

$$\underbrace{9{,}432{,}732}_{\text{۷ رقم}} = \underbrace{9423}_{\text{۴ رقم}} \times 10^3 + \underbrace{723}_{\text{۳ رقم}}$$

ضرب اعداد صحیح بزرگ

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

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

این اعداد صحیح کوچکتر از اعداد اصلی را می‌توان به طور بازگشتی، با تقسیم آنها به اعداد کوچکتر، در هم ضرب کرد.

این فرآیند آنقدر ادامه می‌یابد تا به یک مقدار آستانه برسیم که در آن زمان، عمل ضرب به طریق استاندارد (معمولی) انجام می‌شود.

الگوریتم ضرب اعداد صحیح بزرگ

```
1.  large_integer prod(large_integer u, large_integer v)
2.  {
3.      large_integer x, y, w, z;
4.      int n, m;
5.      n = maximum (number of digits in u, number of digits in v)
6.      if (u = = 0 || v = = 0)
7.          return 0;
8.      else if (n <= threshold)
9.          return u × v obtained in the usual way;
10.     else
11.     {
12.         m = [n/2];
13.         x = u divide 10^m;
14.         y = u rem 10^m;
15.         w = v divide 10^m;
16.         z = v rem 10^m;
17.         return prod(x,w) × 10^2m + (prod(x,z) + prod(w,y)) × 10^m + prod(y,z);
18.     }
19. }
```

تحلیل الگوریتم ضرب اعداد صحیح بزرگ

- بدترین حالت زمانی است که هر دو عدد صحیح هیچ رقمی مساوی صفر نداشته باشند. زیرا در این حالت فراخوانی‌های بازگشتی زمانی به پایان می‌رسند که به حد آستانه‌ای برسیم.
- فرض می‌شود که n توانی از ۲ باشد. در این حالت x، y، w، z دقیقا $n/2$ رقم خواهند داشت.
- عملیات جمع، تفریق، **divide** 10^m، **rem** 10^m یا $10^m \times$ همگی دارای پیچیدگی زمانی خطی می‌باشند.

$$T(n) = \begin{cases} \Theta(1) = 0 & n \le s \\ 4T(\frac{n}{2}) + cn & n = x^2, n > s \end{cases}$$

حد آستانه: تعداد ارقامی که در آن زمان ضرب به صورت معمول اجرا می‌شود.

$$T(n) = \Theta(4^{\lg n}) = \Theta(n^{\lg 4}) = \Theta(n^2)$$

حل ۱: مشابه روشی که در مرتب سازی ادغامی جهت تحلیل استفاده شد.
حل ۲: با استفاده از قضیه اصلی

بهبود الگوریتم ضرب اعداد صحیح بزرگ– کاهش ضرب ها

- الگوریتمی که برای ضرب اعداد صحیح بزرگ ارائه شد هنوز از درجه دوم می‌باشد.
- در الگوریتم ارائه شده چهار فراخوانی برای ضرب‌های xw، xz، yw و yz انجام می‌شود.
- می‌خواهیم با تغییر کوچکی در روابط استفاده شده تعداد ضرب‌ها را کاهش دهیم.
- بازنویسی روابط مورد نیاز:

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

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

اکنون با توجه به رابطه فوق، جهت بدست آوردن xw، $xz + yw$ و yz که در الگوریتم مورد نیاز می‌باشد می‌توانیم سه مقدار زیر را محاسبه نماییم.

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

- برای محاسبه سه مقدار فوق به سه عمل ضرب و چند عمل جمع و تفریق اضافی (که زمان آنها خطی است) نیاز خواهد بود.

الگوریتم بهبودیافته ضرب اعداد صحیح بزرگ

```
1.  large_integer prod2(large_integer u, large_integer v)
2.  {
3.      large_integer x, y, w, z, r, p, q;
4.      int n, m;
5.      n = maximum (number of digits in u, number of digits in v);
6.      if (u == 0 || v == 0)
7.          return 0;
8.      else if (n <= threshold)
9.          return u × v obtained in the usual way;
10.     else
11.     {
12.         m = [n/2];
13.         x = u divide 10^m; y = u rem 10^m;
14.         w = v divide 10^m; z = v rem 10^m;
15.         r = prod2(x + y, w + z);
16.         p = prod2(x, w);
17.         q = prod2(y, z);
18.         return p × 10^2m + (r-p-q) × 10^m+q;
19.     }
20. }
```

تحلیل – الگوریتم بهبودیافته ضرب اعداد صحیح بزرگ

- فرض می‌شود که n توانی از ۲ باشد. در این حالت x، y، w، z دقیقا $n/2$ رقم خواهند داشت.

$$n/2 \le \text{تعداد ارقام در } x+y \le (n/2)+1$$

$$n/2 \le \text{تعداد ارقام در } w+z \le (n/2)+1$$

- بنابراین اندازه ورودی‌ها برای فراخوانی‌های تابع بدین صورت است:

فراخوانی	اندازه ورودی
prod2(x+y, w+z)	$n/2 \le$ اندازه ورودی $\le (n/2)+1$
prod2(x, w)	$n/2$
prod2(y, z)	$n/2$

- از اینرو T(n) در شرایط زیر صدق می‌کند:

$$T(n)=\begin{cases} T(s)=0 & n \le s \\ 3T(\frac{n}{2})+cn \le T(n) \le 3T(\frac{n}{2}+1)+cn & n=x^2, n>s \end{cases}$$

$$T(n)=\Omega(n^{\lg_2 3}) \approx \Omega(n^{1.58})$$

$$T(n)=O(n^{\lg_2 3}) \approx O(n^{1.58})$$

چه وقت نباید از D&C استفاده کرد؟

- هنگامی که نمونه به اندازه n به دو یا چند نمونه تقسیم شود که اندازه آنها نیز تقریبا برابر با n است. (مرتبه نمایی)

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

- هنگامی که نمونه با اندازه n تقریبا به n نمونه با اندازه n/c تقسیم شود که c یک مقدار ثابت است. (مرتبه $n^{\lg n}$)

تمرین

- ۲-۵ حاصلضرب دو عدد ۴۵۶۴۷۳۷۸۹۴ و ۱۲۵۴۶۳۷۴۹۸ را با استفاده از الگوریتم تقسیم و حل ضرب اعداد صحیح بزرگ، به دست آورید.
- ۲-۶ الگوریتم تقسیم و حل برای ضرب چند جمله ای ها را بیان کنید، نحوه اجرا و زمان اجرای آن را تحلیل نمایید.