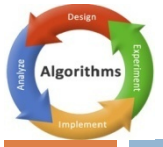
بسمه تعالی

بخش سوم

روش تقسیم وحل

(Divide-and-Conquer)

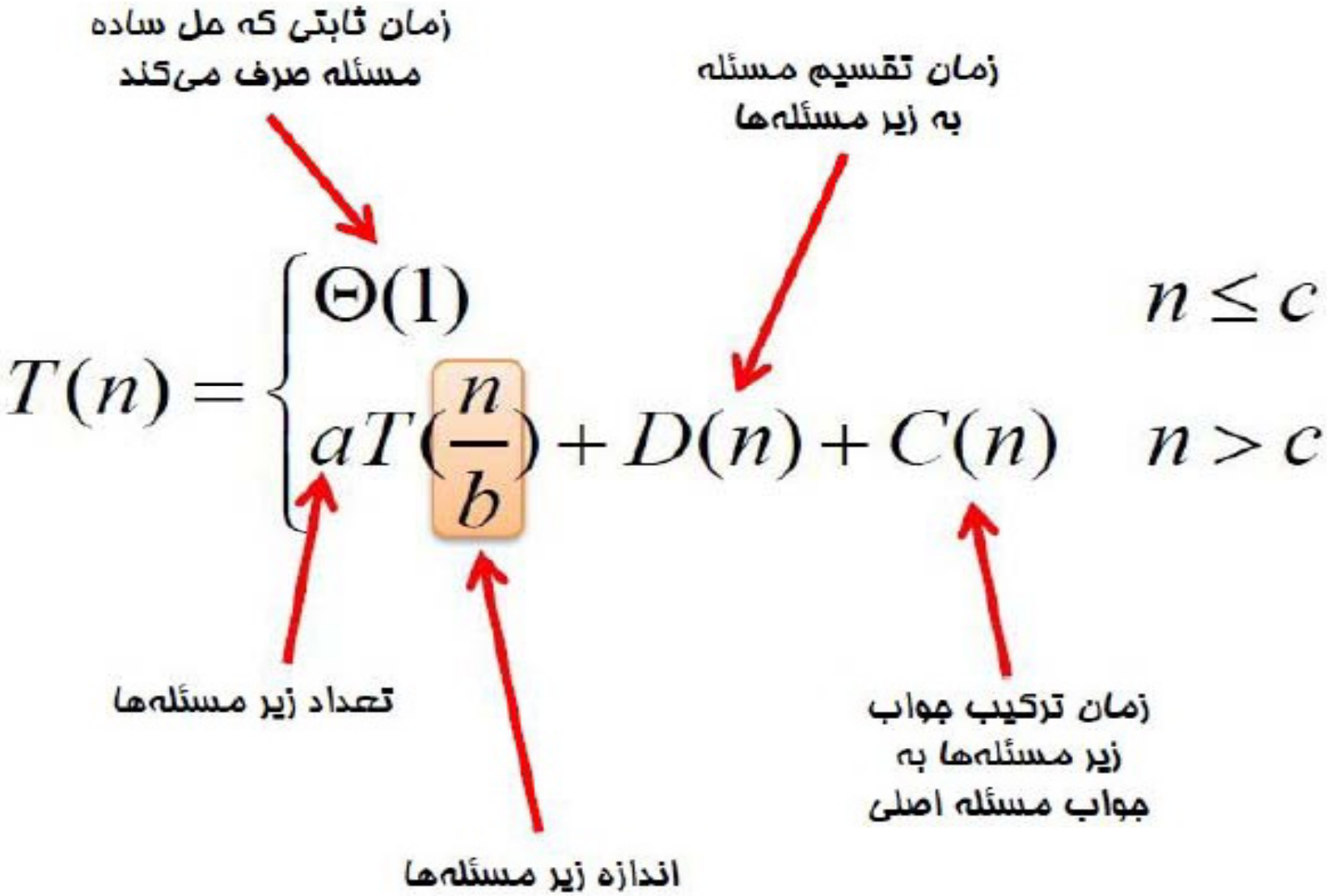
ارائه دهنده : محمد فرجی مهماندار

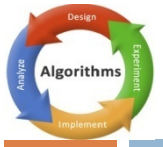


تقسیم و حل

- تکنیکی برای حل مسائل است که در آن مسئله اصلی با استفاده از ساختار بازگشتی به تعدادی زیر مسئله تقسیم می شود (تقسیم)
- اگر پس از تقسیم مسأله، زیرمسأله‌ها هنوز بزرگ و غیر قابل حل هستند، آنها را دوباره به چند زیرمسأله دیگر تقسیم می‌کنیم.
- زیرمسائل که به حد کافی بطور تکراری کوچک شده‌اند، حل می‌شوند.
- از ترکیب راه حل های زیر مسئله‌های کوچکتر، راه حل زیر مسئله‌های بزرگتر و در نهایت راه حل مسئله اصلی بدست می آید. (حل)
- روش تقسیم وحل یک رهیافت بالا به پایین (Top – Down) است که توسط روتین های بازگشتی به کار می رود.
- نمونه ای از یک مساله را به صورت بازگشتی به تعدادی نمونه کوچکتر تقسیم کن تا زمانی که راه حل نمونه های کوچکتر به سادگی قابل تعیین باشند.

تحلیل الگوریتم های تقسیم و حل





جستجوی دودویی

هدف این الگوریتم جستجو و یافتن یک عنصر در یک آرایه ی مرتب شده است.

اگر x با عنصر وسط برابر است خارج شو، در غیر این صورت:

۱ – تقسیم آرایه به دو زیر آرایه با اندازه ای تقریبا برابر نصف اندازه آرایه اولیه. اگر x کوچکتر از عنصر وسط می باشد، آرایه سمت چپ را انتخاب کن. اگر x بزرگتر از عنصر وسط می باشد، آرایه سمت راست را انتخاب کن.

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

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

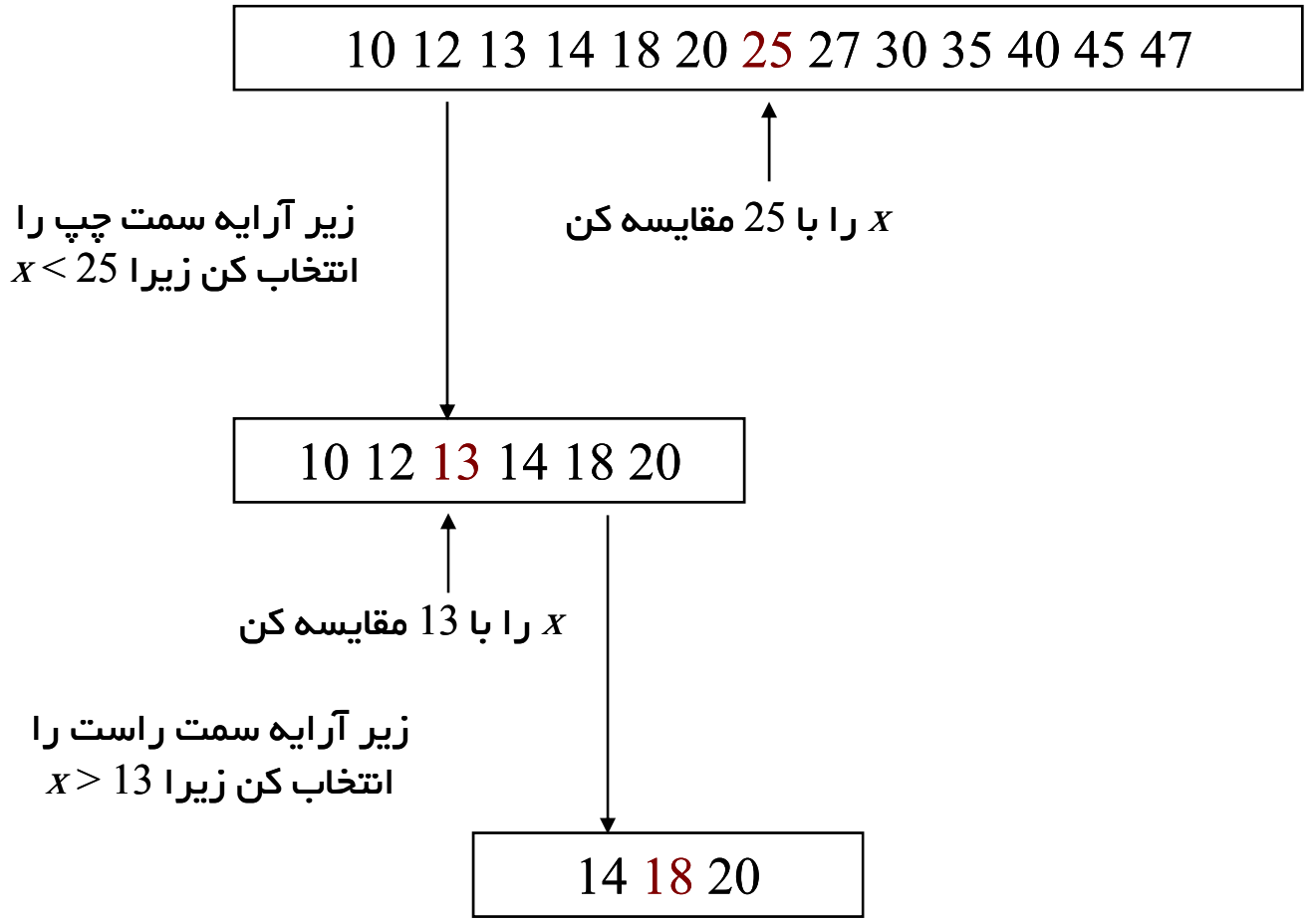
- مثال : فرض کنید $x = 18$ و آرایه به صورت زیر باشد:

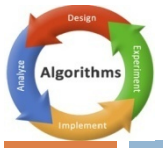
10 12 13 14 18 20 25 27 30 35 40 45 47

↑

عنصر وسط

کل فرآیند جستجو

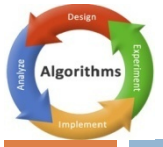




جستجوی دودویی

الگوریتم ۲-۱ جستجوی دودویی (بازگشتی)

- مساله: تعیین کنید که آیا x در آرایه مرتب شده S به اندازه n وجود دارد یا خیر.
- ورودی ها: عدد صحیح و مثبت n، آرایه مرتب S که از ۱ تا n اندیس گذاری شده است، کلید x.
- خروجی ها: $location$، موقعیت x در S (اگر x در S نباشد برابر صفر می باشد)

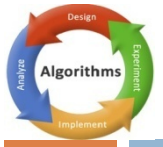


الگوریتم جستجوی دودویی

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

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

$T(n) \in O(\lg n)$



پیچیدگی زمانی: بدترین حالت

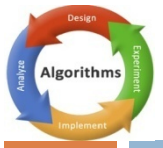
- عمل اصلی: مقایسه x با $S[mid]$
- اندازه ورودی: تعداد عناصر آرایه، n
- پیچیدگی زمانی:

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

- حل: $W(n) = \lg n + 1$

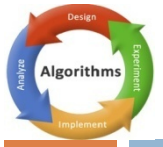
و اگر n به توانی از دو محدود نباشد:

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



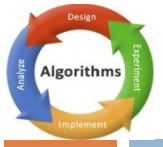
تمرین

- برای یافتن هر عنصر در آرایه های زیر چه تعداد مقایسه لازم است؟ میانگین این مقایسات چقدر است؟
- $A = \{10,12,13,14,18,20,25,27\}$
- $B = \{10,12,13,14,19,20,25,27,30,35,40,45,47\}$
- اگر در الگوریتم جستجوی دودویی بجای مقایسه عنصر x با عنصر وسط آرایه، مقایسه را همزمان با عناصر $n/4$، $2n/4$ و $3n/4$ آرایه انجام دهیم و بر اساس آنها تصمیم گیری کنیم، رابطه بازگشتی به چه صورت خواهد شد؟ آیا بهبودی در زمان اجرای این الگوریتم حاصل می شود یا خیر ؟



یافتن کوچکترین و بزرگترین عنصر آرایه

- **تقسیم** آرایه به دو زیر آرایه به اندازه $n / 2$ عنصر.
- **حل** هر یک از زیر آرایه ها با یافتن کوچکترین و بزرگترین عنصر در آن. اگر زیر آرایه به اندازه کافی کوچک نبود$(n > 2)$، برای حل آن به روش بازگشتی عمل می کنیم.
- **ترکیب** راه حل های زیر آرایه ها با یافتن کوچکترین عنصر در بین کوچکترین عنصر دو زیر آرایه و یافتن بزرگترین عنصر در بین بزرگترین عنصر دو زیر آرایه.



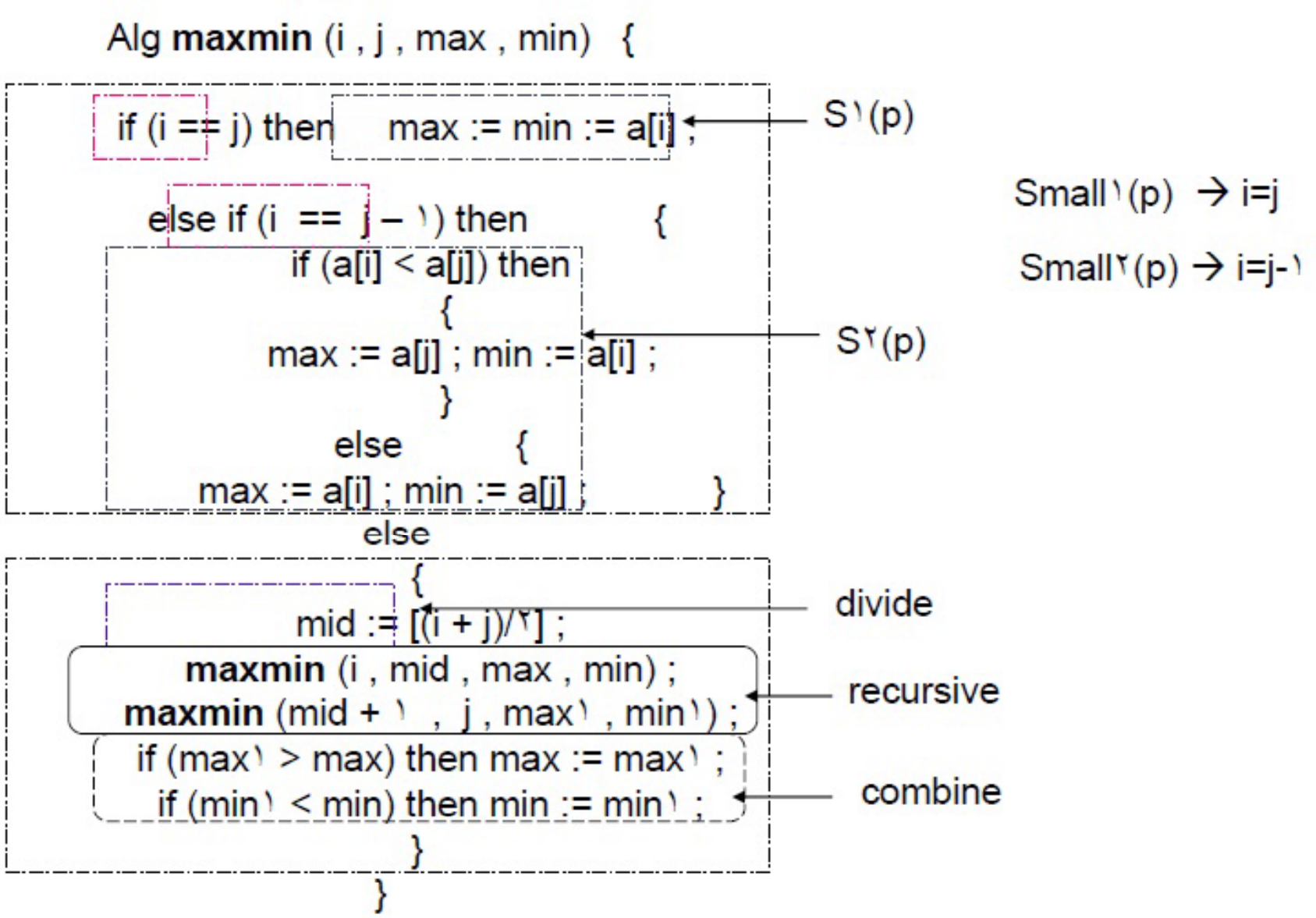
یافتن کوچکترین و بزرگترین عنصر آرایه بروش غیر بازگشتی

```
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] ;
                }
              }
```

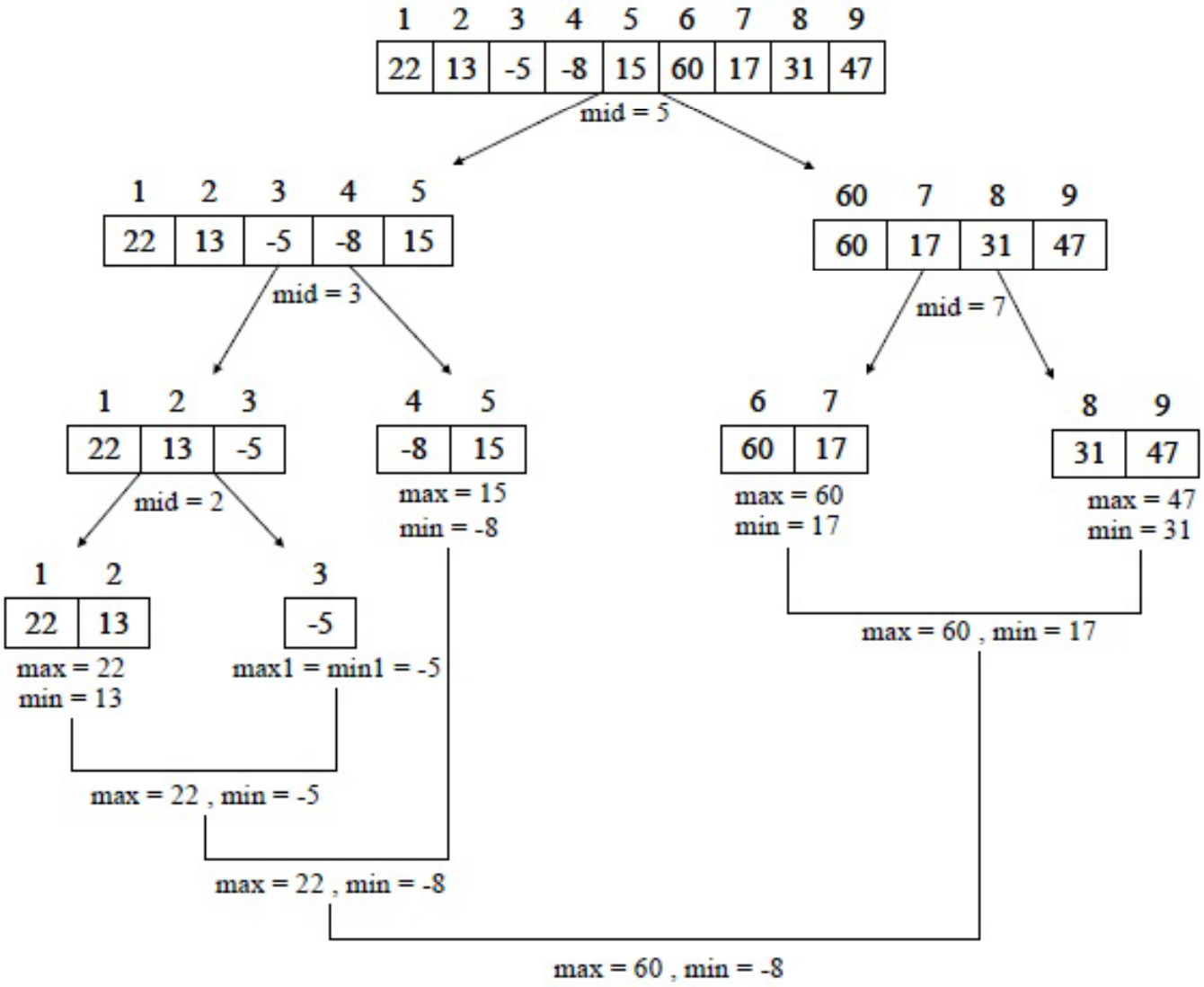
با گذاشتن یک else قبل از if دوم الگوریتم کمی بهتر می شود

```
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] ;
                }
              }
```

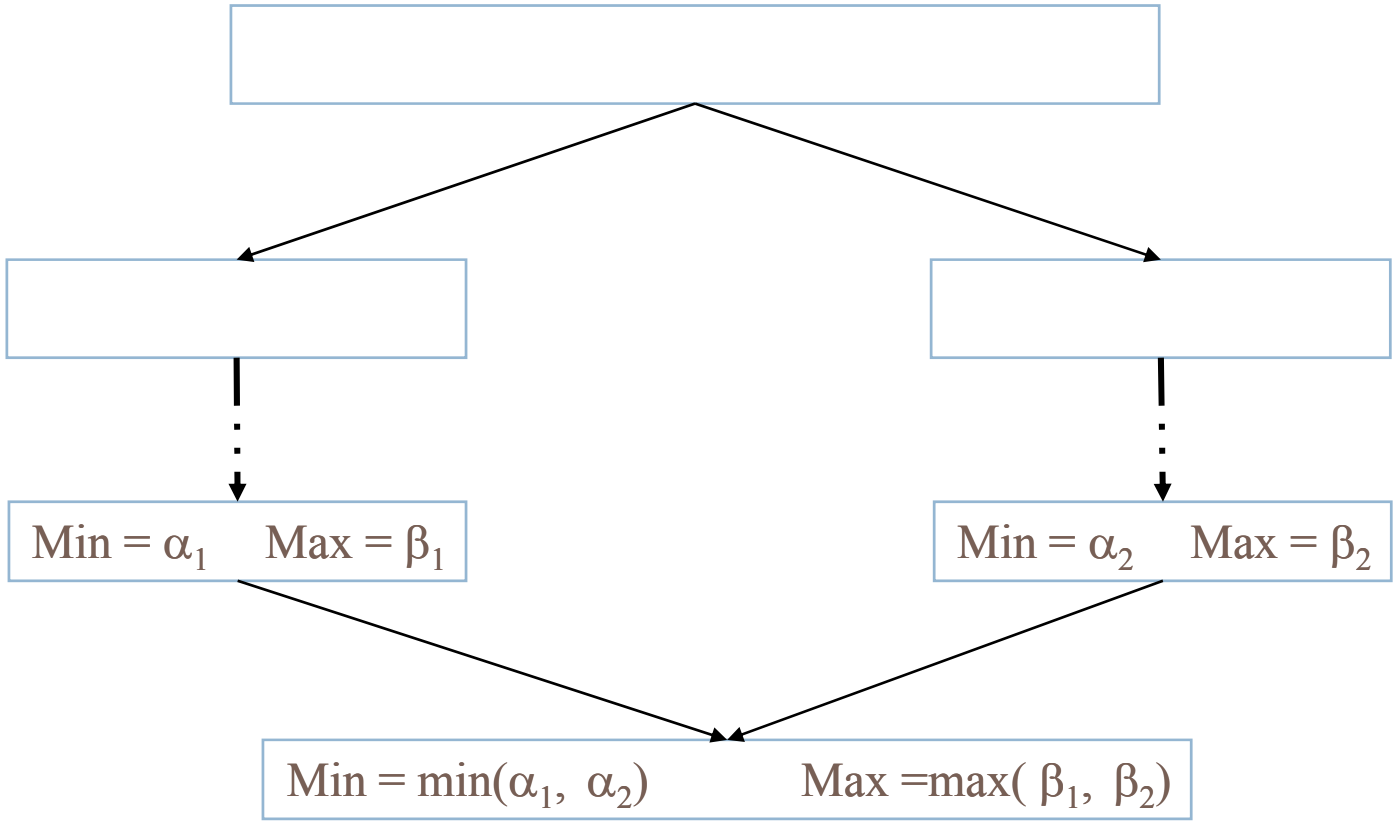
یافتن کوچکترین و بزرگترین عنصر آرایه بروش بازگشتی



یافتن کوچکترین و بزرگترین عنصر آرایه بروش بازگشتی



یافتن کوچکترین و بزرگترین عنصر آرایه بروش بازگشتی



پیچیدگی زمانی: همه حالات

- عمل اصلی: مقایسه
- اندازه ورودی: تعداد عناصر آرایه، n
- پیچیدگی زمانی:

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

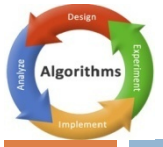
$$\begin{aligned}T(n)&=2T(n/2)+2=2[2T(n/2^2)+2]+2\\&=2^2T(n/2^2)+2+2^2=2[2T(n/2^3)+2]+2+2^2\\&=2^3T(n/2^3)+2+2^2+2^3\\&=\cdots\\&=2^iT(n/2^i)+\sum_{k=1}^{i}2^k\end{aligned}$$

$$n/2^i=2\Rightarrow n=2^{i+1}$$

$$T(n)=(n/2)T(2)+2^{i+1}-2=n/2+n-2=3n/2-2$$

و اگر n به توانی از دو محدود نباشد:

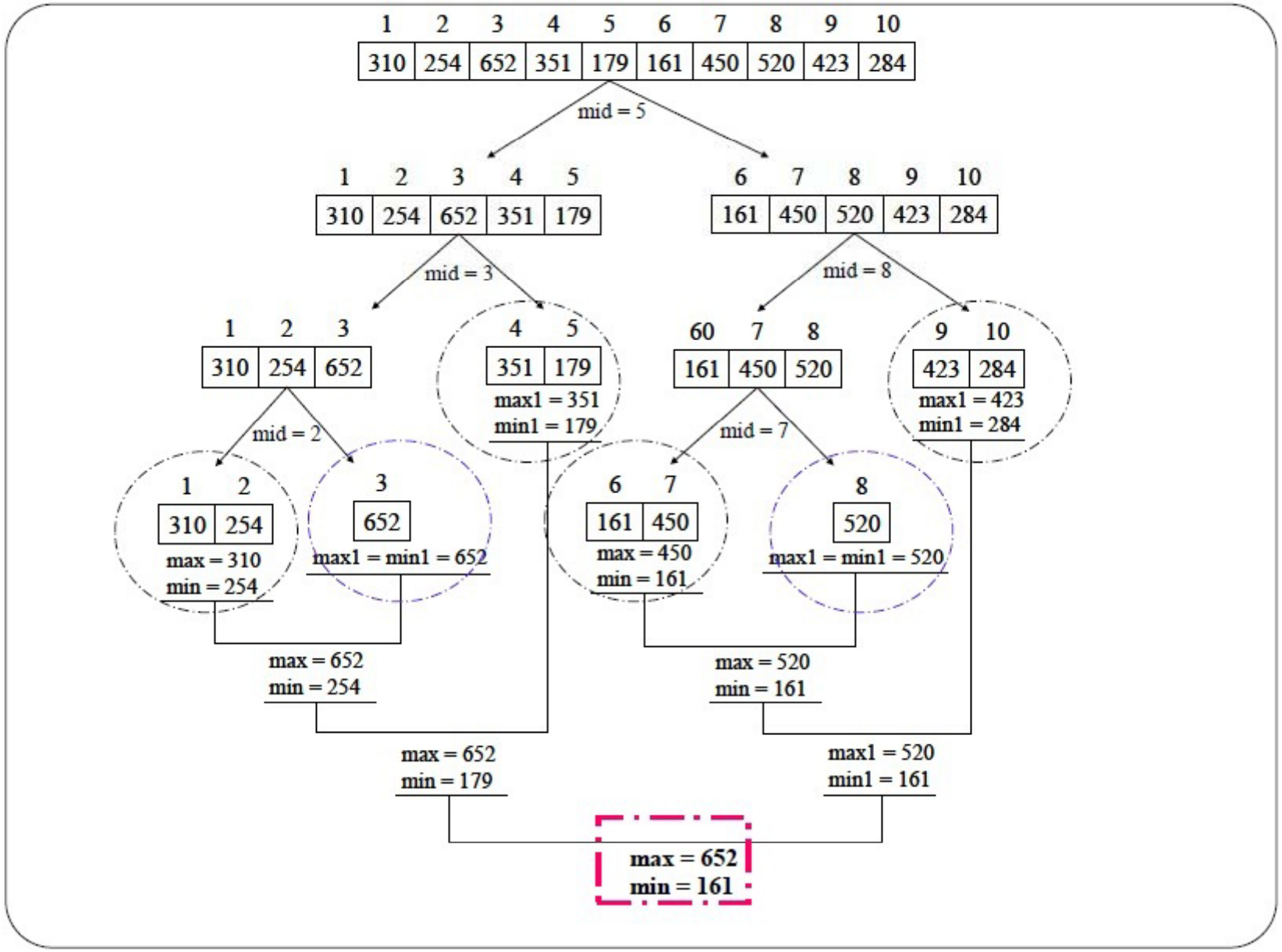
$$T(n)=\lceil 3n/2\rceil-2\in\Theta(n)$$



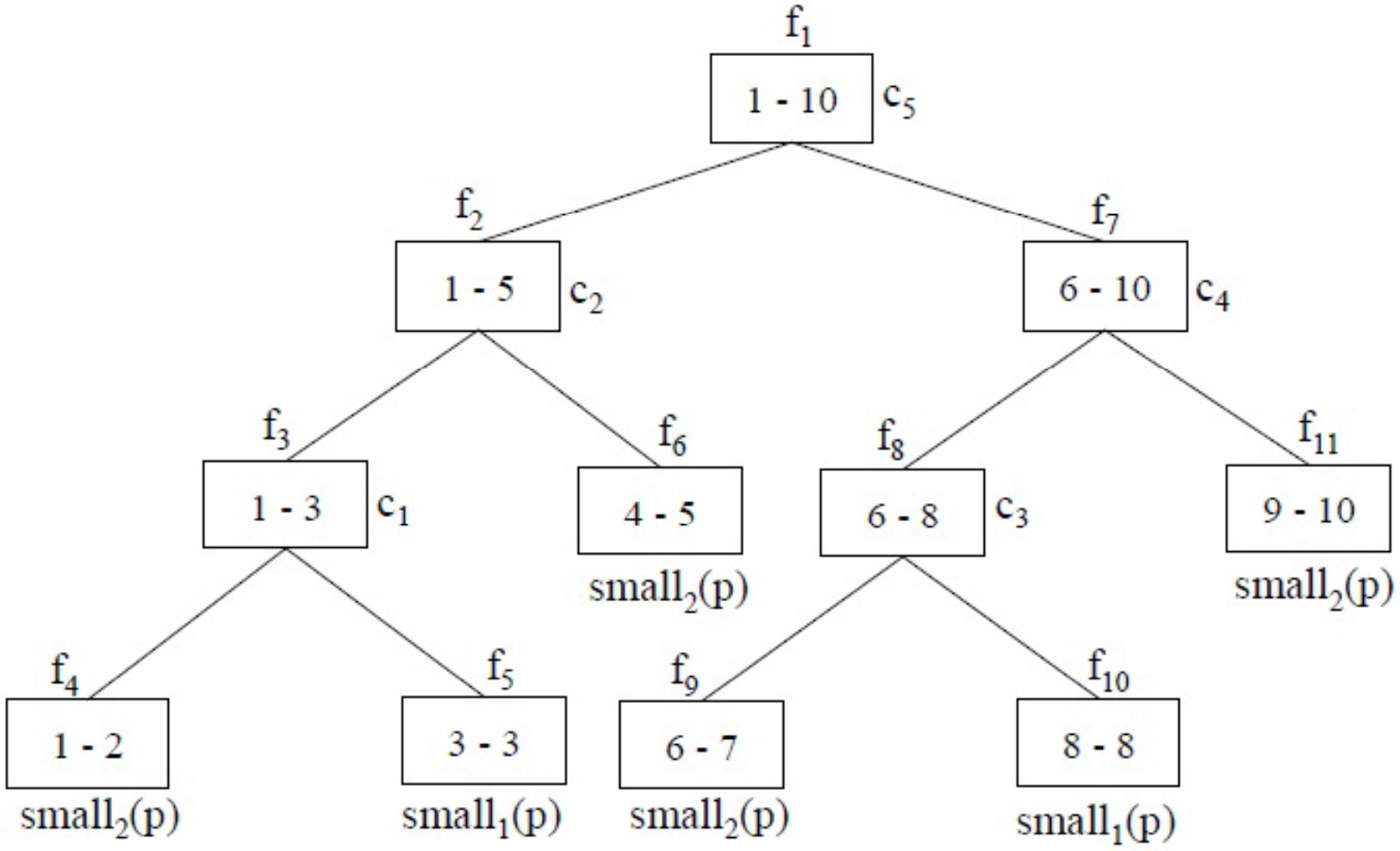
یافتن کوچکترین و بزرگترین عنصر آرایه بروش بازگشتی

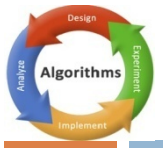
نام الگوریتم	روش حل مساله	بهترین حالت	بدترین حالت	میانگین
max-min	غیر بازگشتی	$2(n-1)$	$2(n-1)$	$2(n-1)$
max-min	غیر بازگشتی بهینه	$(n-1)$	$2(n-1)$	کمتر از $2(n-1)$
max-min	بازگشتی (تقسیم و حل)	$\frac{3n}{2}-2$	$\frac{3n}{2}-2$	$\frac{3n}{2}-2$

مثال یافتن کوچکترین و بزرگترین عنصر آرایه



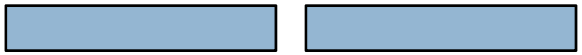
درخت فراخوانی تابع maxmin





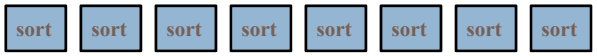
مرتب سازی ادغامی(Merge Sort)

- هدف این الگوریتم مرتب سازی یک آرایه است.
- الگوریتم مرتب سازی ادغامی به زبان ساده:
- ابتدا آرایه را به دو قسمت مساوی تقسیم می کنیم.



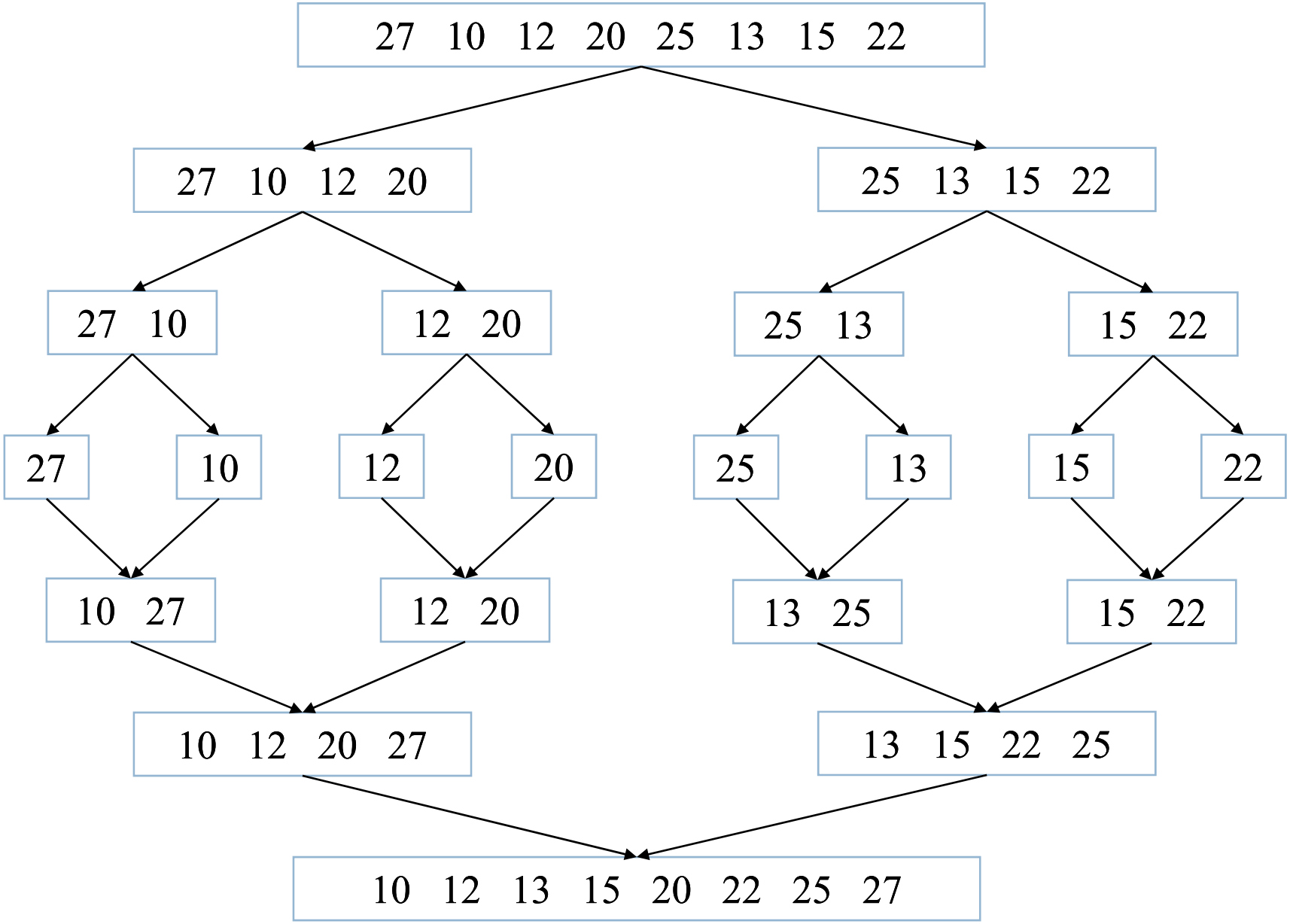
- اگر آرایه های تقسیم شده به اندازه کافی کوچک باشند که در زمان معقول مرتب شوند که هیچ و الا آنها را مجددا به دو قسمت تقسیم می کنیم. این مرحله را تا کوچک شدن معقول آرایه ها تکرار می کنیم.

- آرایه های به حد کافی کوچک را مرتب می نماییم.



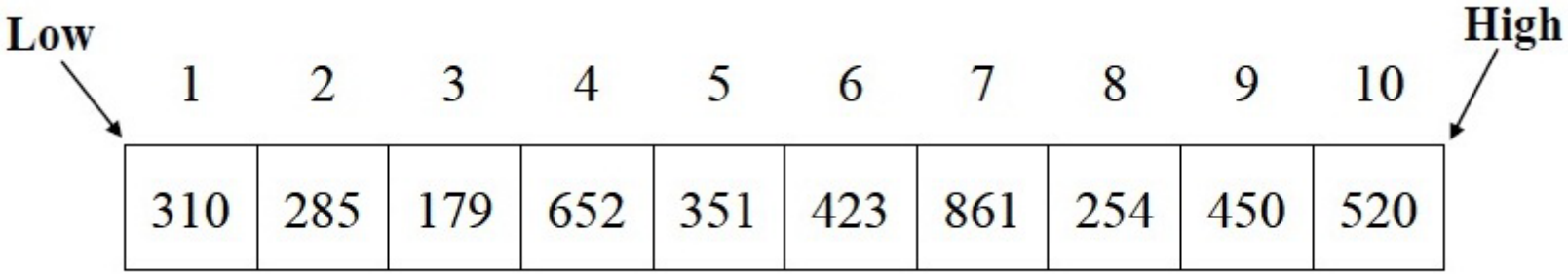
- آرایه های مرتب شده را ادغام می کنیم و این کار را تا جایی ادامه می دهیم تا ارایه اصلی مرتب شود.

مثال از مرتب سازی ادغامی



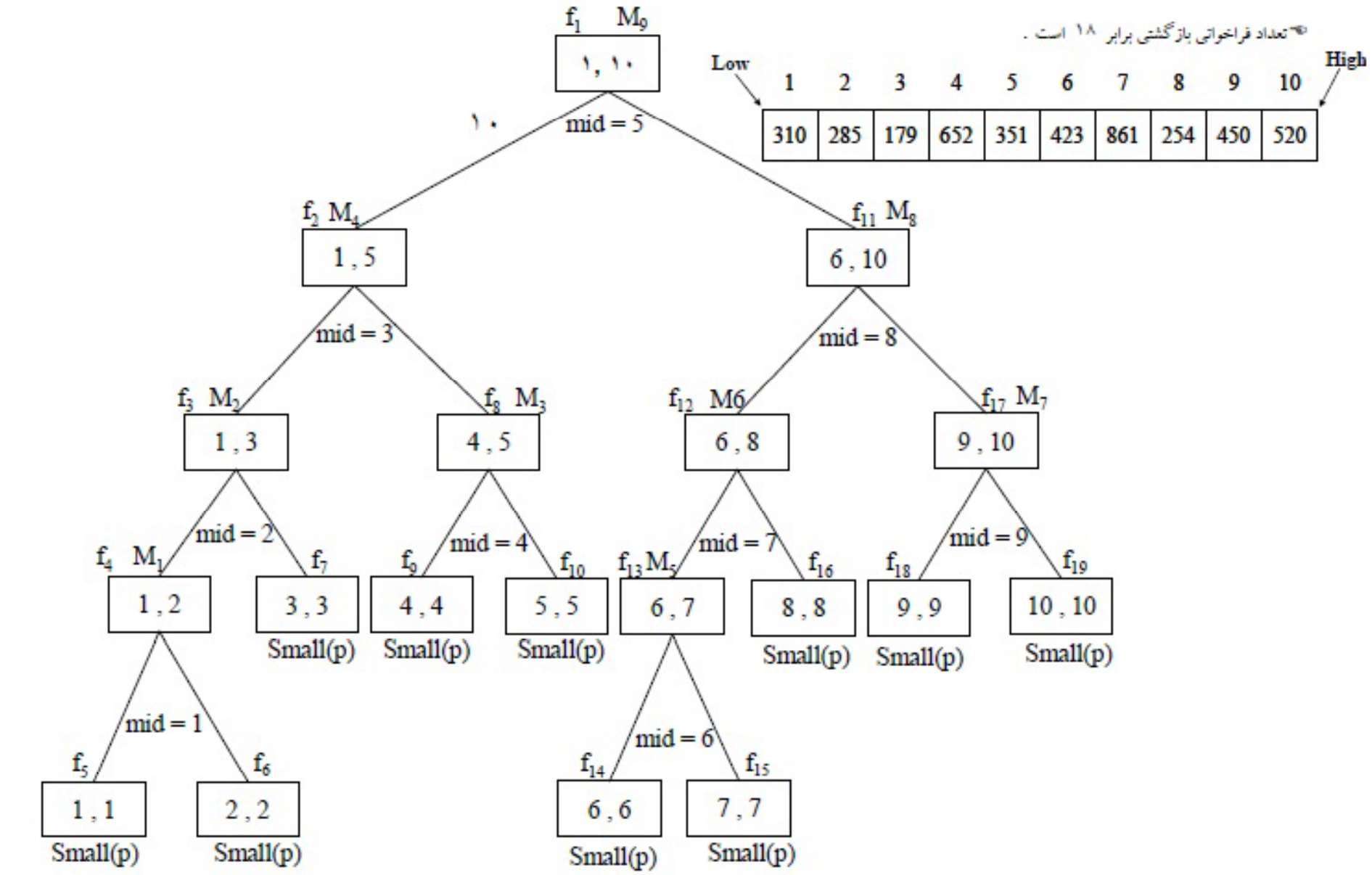
الگوریتم مرتب سازی ادغامی

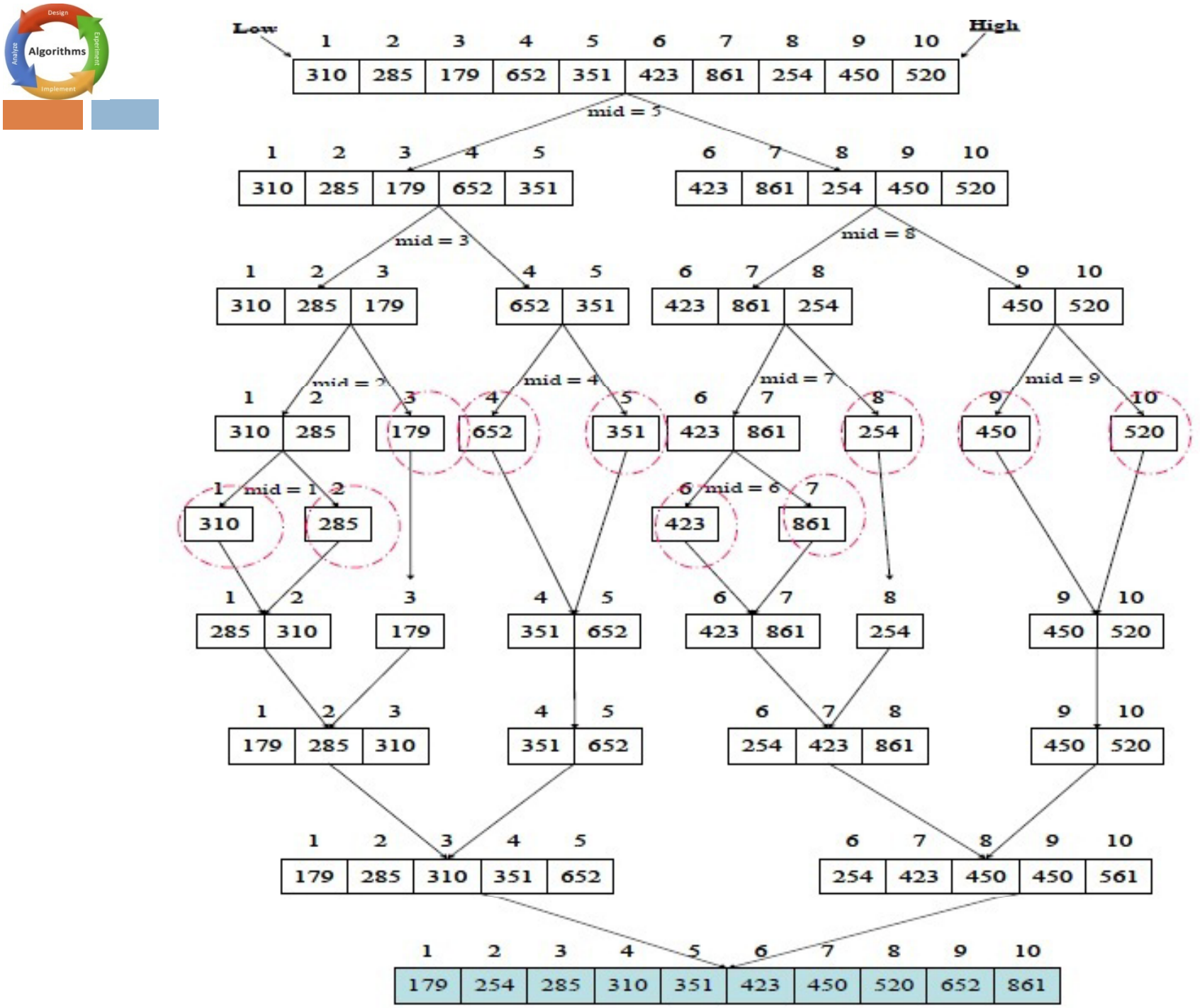
لیست نامرتبی با n عنصر داریم که می خواهیم این لیست را به ترتیب صعودی مرتب کنیم.

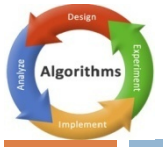



```
Alg mergesort (low , high)
{
  if (low < high)                          // !small(p)
  {
        mid := ⌊(low + high) / 2⌋ ;        // Divide
       mergsort (low , mid) ;              // recursive (left)
       mergesort (mid + 1  ,  high) ;      // recursive (right)
       merge (low , mid , high)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

الگوریتم مرتب سازی ادغامی

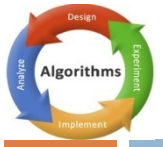



Low
High
1 2 3 4 5 6 7 8 9 10
310 285 179 652 351 423 861 254 450 520
mid = 5
1 2 3 4 5
310 285 179 652 351
6 7 8 9 10
423 861 254 450 520
mid = 3
mid = 8
1 2 3
310 285 179
4 5
652 351
6 7 8
423 861 254
9 10
450 520
mid = 4
mid = 7
mid = 9
1 2
310 285
3
179
4
652
5
351
6 7
423 861
8
254
9
450
10
520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



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 + ١ ; 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 - ١ ];
  }
  arr[ j ] = t;
 }
}
```



پیچیدگی الگوریتم Merge Sort

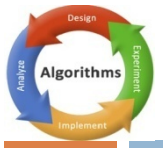
$T(n)$: تعداد مقایسه ها برای مرتب سازی یک آرایه با اندازه n با استفاده از مرتب سازی ادغامی

$T(n/2)$: تعداد مقایسه ها برای مرتب سازی یک آرایه با اندازه n/2 با استفاده از مرتب سازی ادغامی

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

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

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)

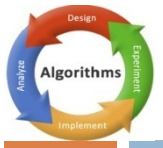


تمرین

- آرایه زیر را به شیوه مرتب سازی ادغامی، مرحله به مرحله مرتب نمایید.

$$A = \{20,12, 3,14,18,11,15, 7\}$$

- اگر در مرتب سازی ادغامی بجای تقسیم آرایه به دو زیر آرایه، آرایه را به سه زیر آرایه تقسیم نماییم، رابطه بازگشتی به چه صورت خواهد شد؟ آیا بهبودی در زمان اجرای این الگوریتم حاصل می شود یا خیر ؟
- نظرتان در مورد تقسیم به ۴ زیر آرایه چیست؟



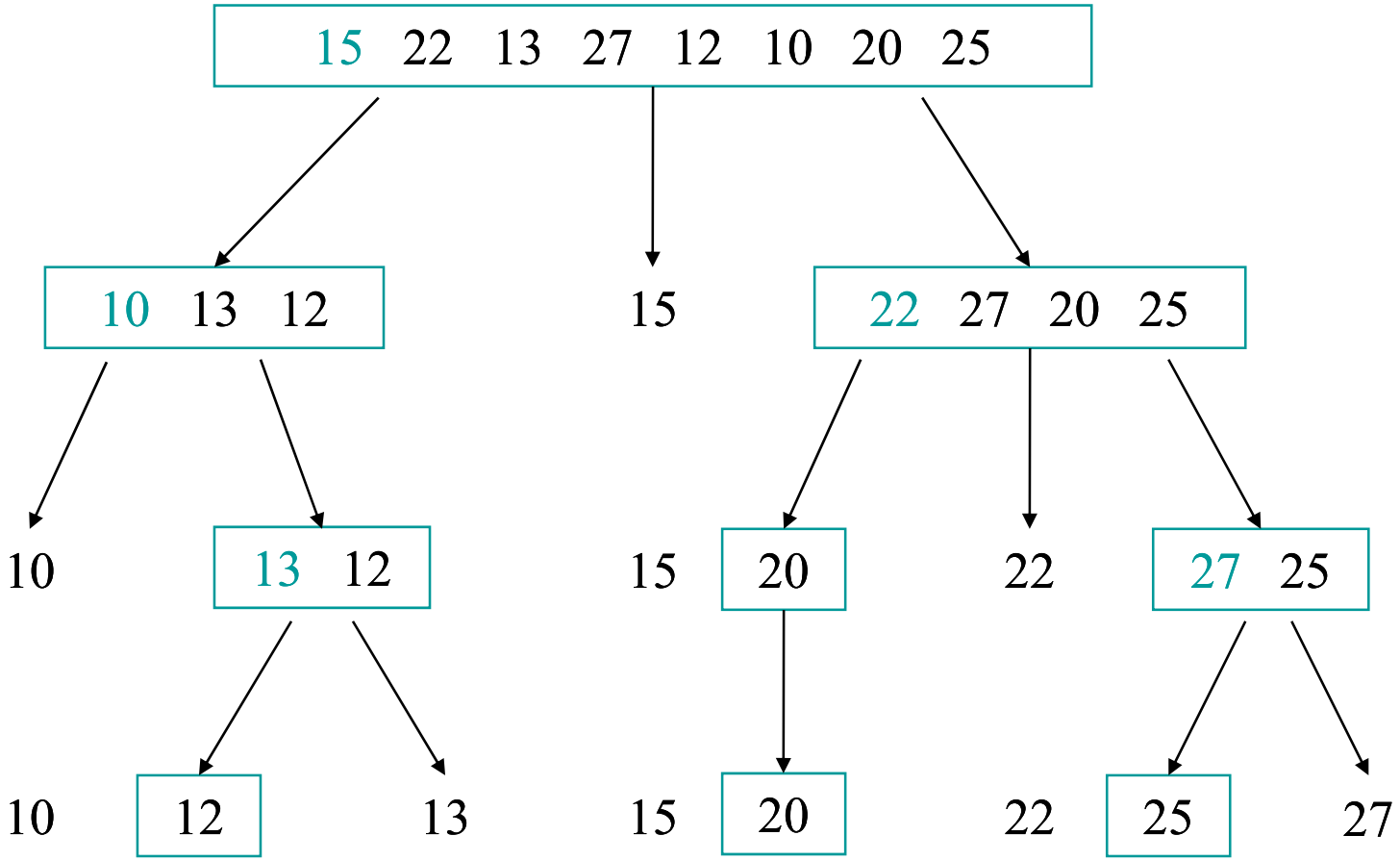
مرتب سازی سریع (Quick Sort)

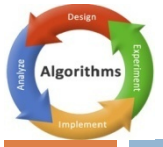
- توسعه یافته توسط Hoare (1962)
- مراحل الگوریتم :
  - انتخاب عنصر محوری (می تواند هر عنصری باشد معمولا عنصر اول)
  - تقسیم آرایه به دو بخش به طوری که عناصر کوچکتر از عنصر محوری در سمت چپ و عناصر بزرگتر از آن در سمت راست آن قرار بگیرند.

عناصر بزرگتر از عنصر محوری	عنصر محوری	عناصر کوچکتر ازعنصر محوری

  - مرتب سازی هر بخش به صورت بازگشتی
  - مرتب سازی سریع، به طور بازگشتی فراخوانی می شود تاهر یک از دوآرایه را مرتب کند، آن ها نیز افراز می شوند واین روال ادامه می یابد تا به آرایه ای با یک عنصر برسیم.

مثال: مرتب سازی سریع





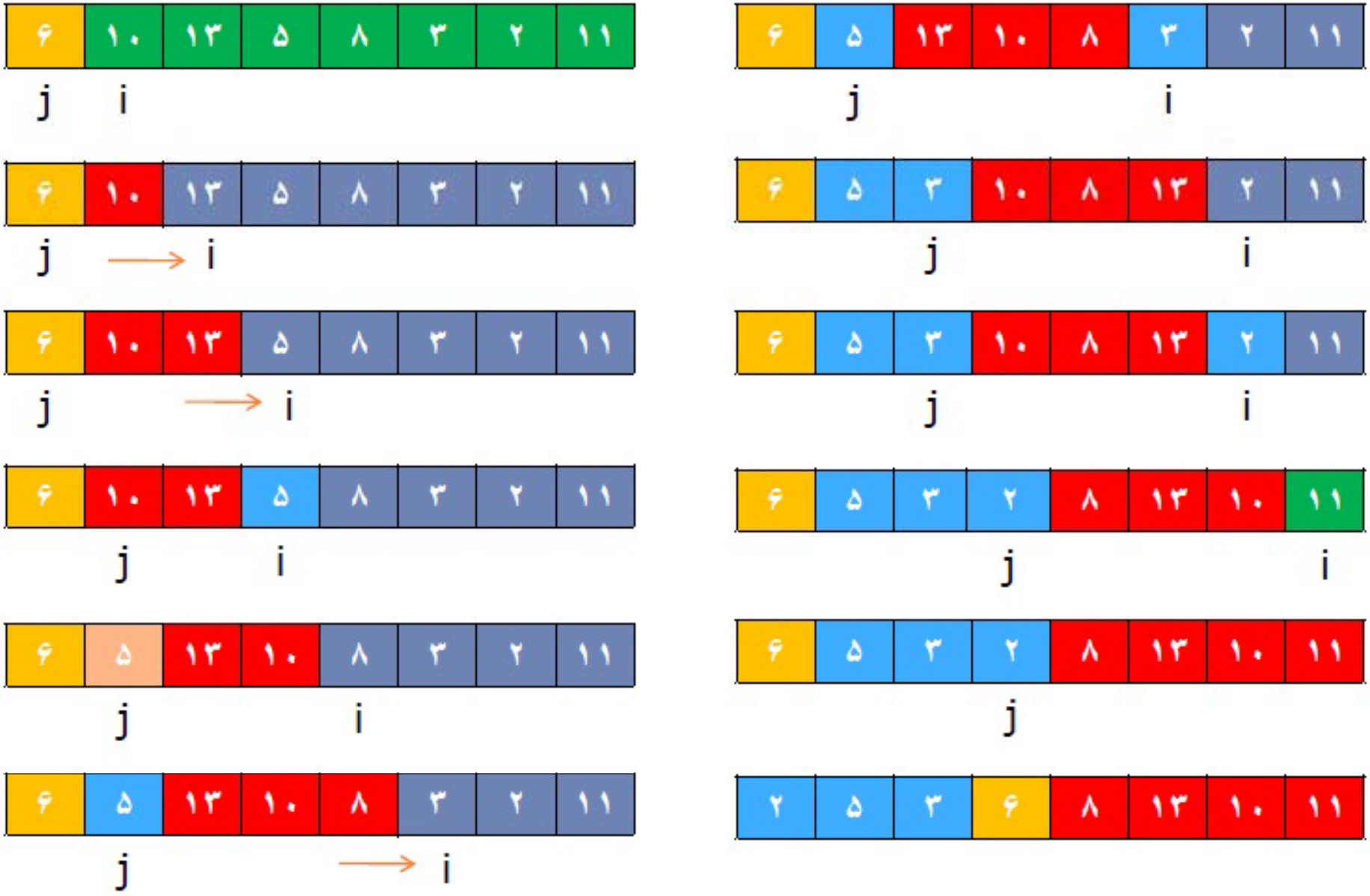
الگوریتم مرتب سازی سریع

```
Void quicksort(int low, int high)
{
  int pivot;
  if (high > low){
        partition(low, high, pivot);
        quicksort (low, pivot-1);
        quicksort (pivot+1,high);
}
```

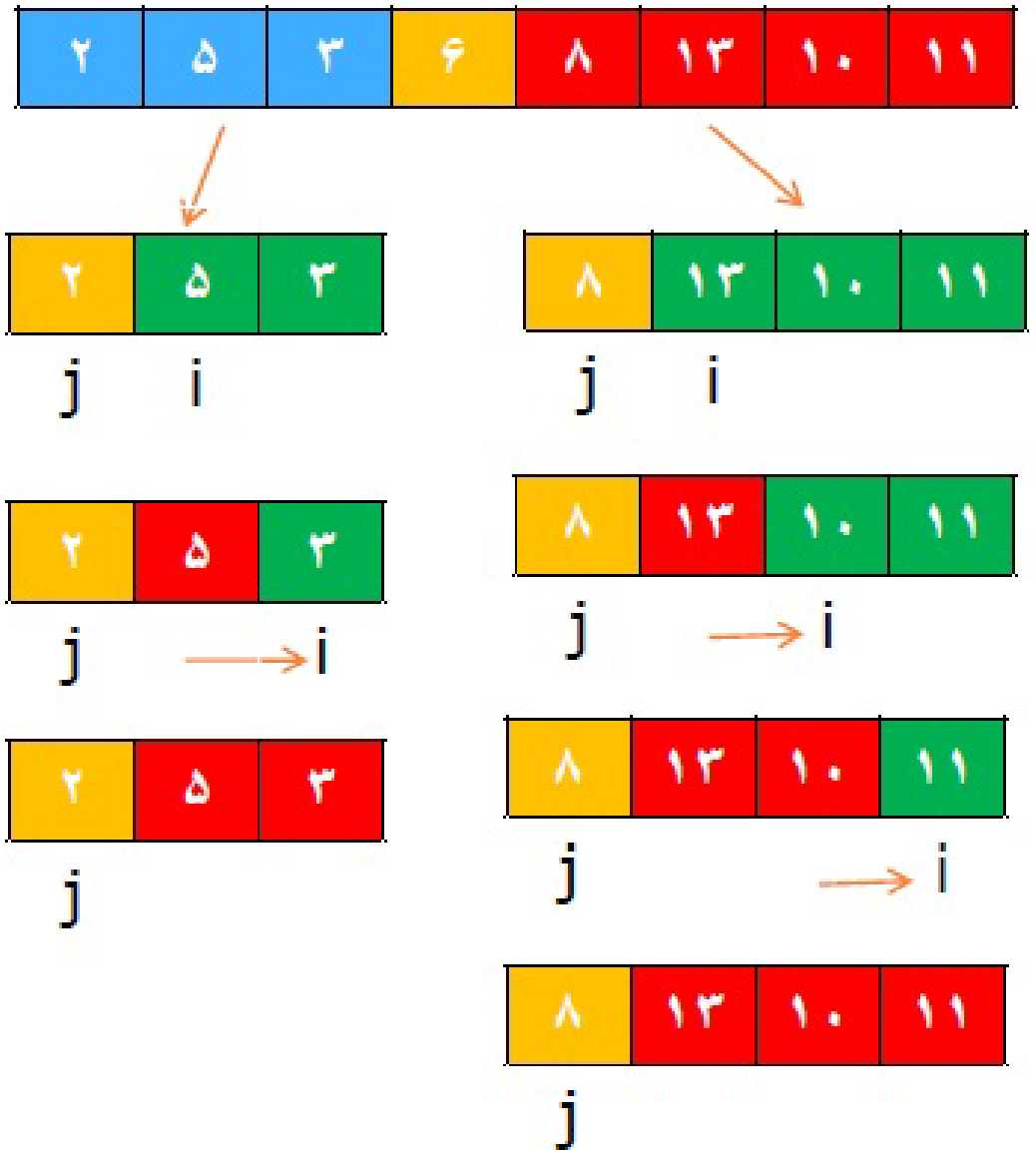
```
void partition (int low, int high, int&pivot)
{
  int i,j, pivotitem;
  pivotitem = S[low];
  j=low;
  for (i=low+1; i<=high; i++)
       if (S[i] < pivotitem){
          j++;
          swap S[i] and S[j];
       }
  pivot= j;
  swap S[low] and S[pivot];
}
```

$T(n) \in \Theta(n \lg n)$

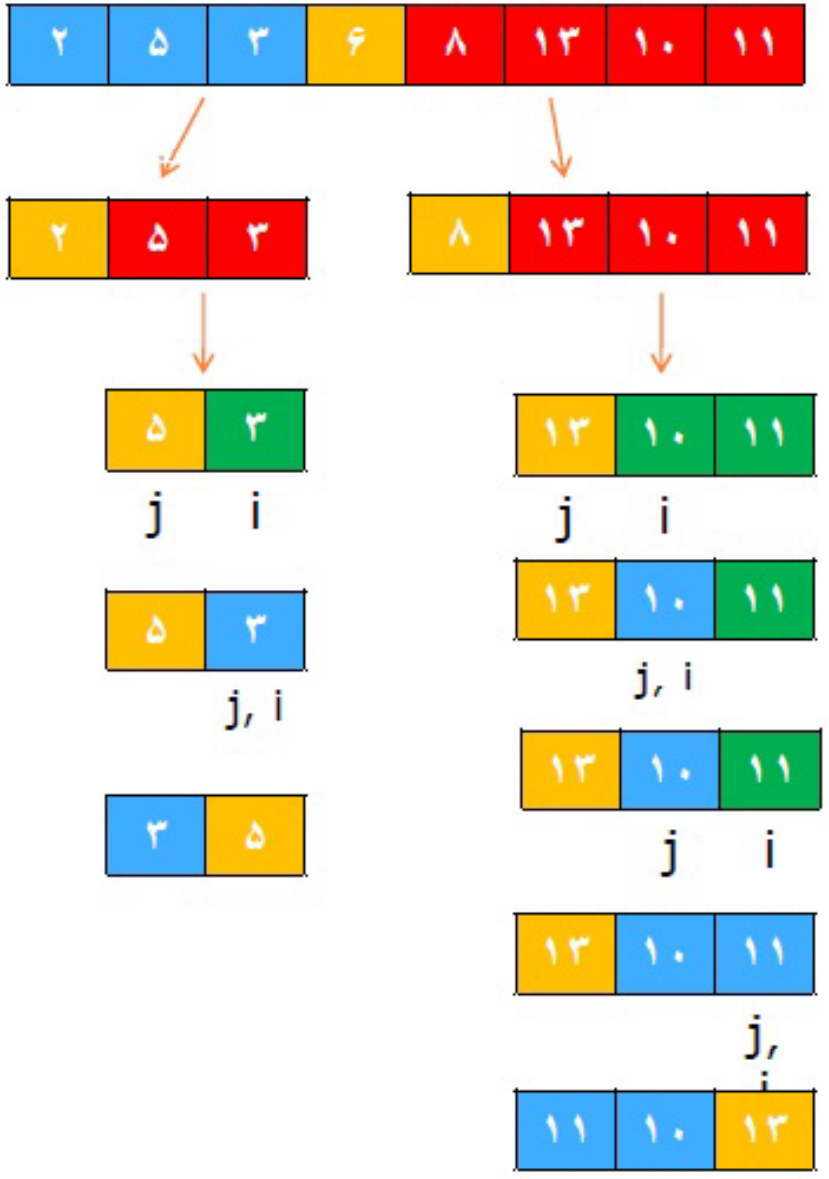
افراز (Partitioning) سطح صفر



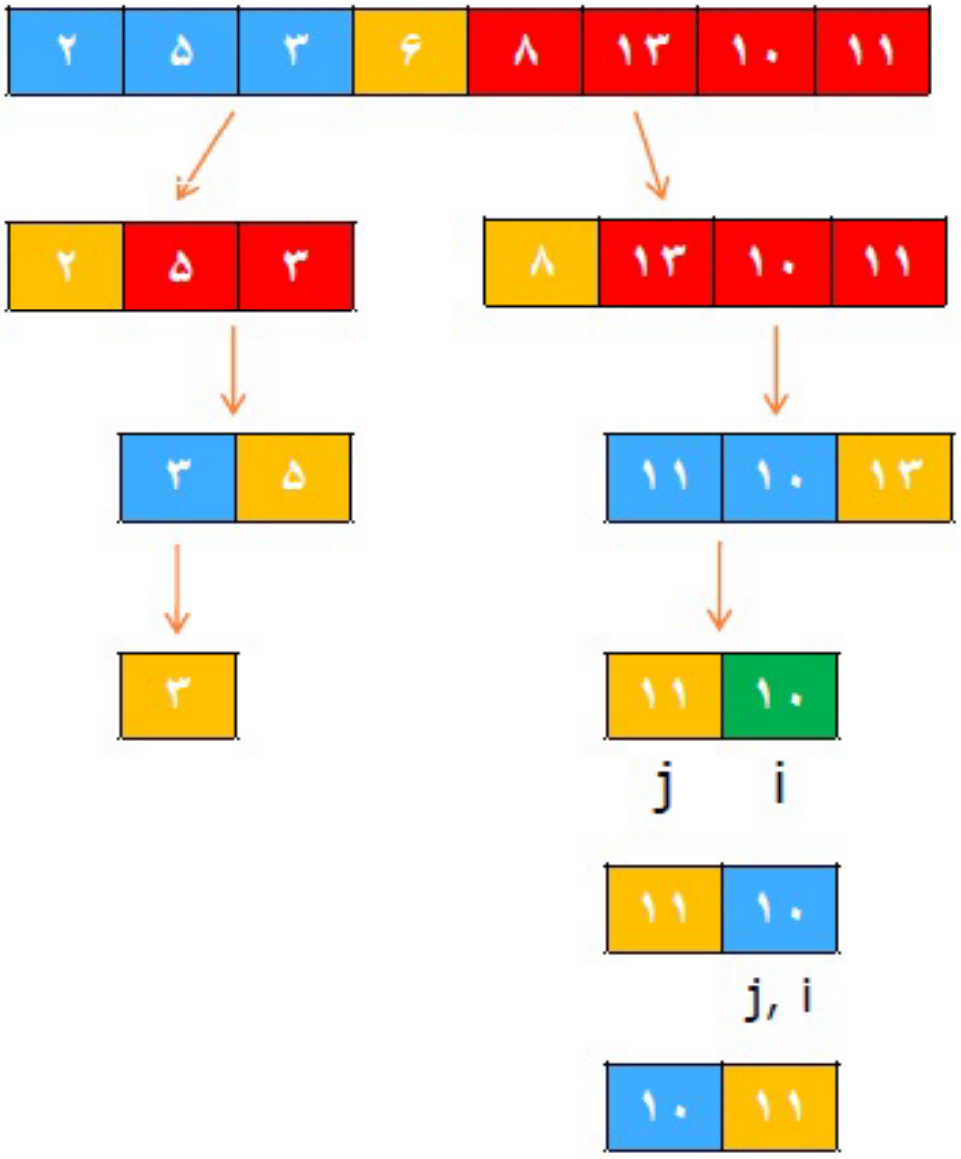
افراز (Partitioning) سطح یک



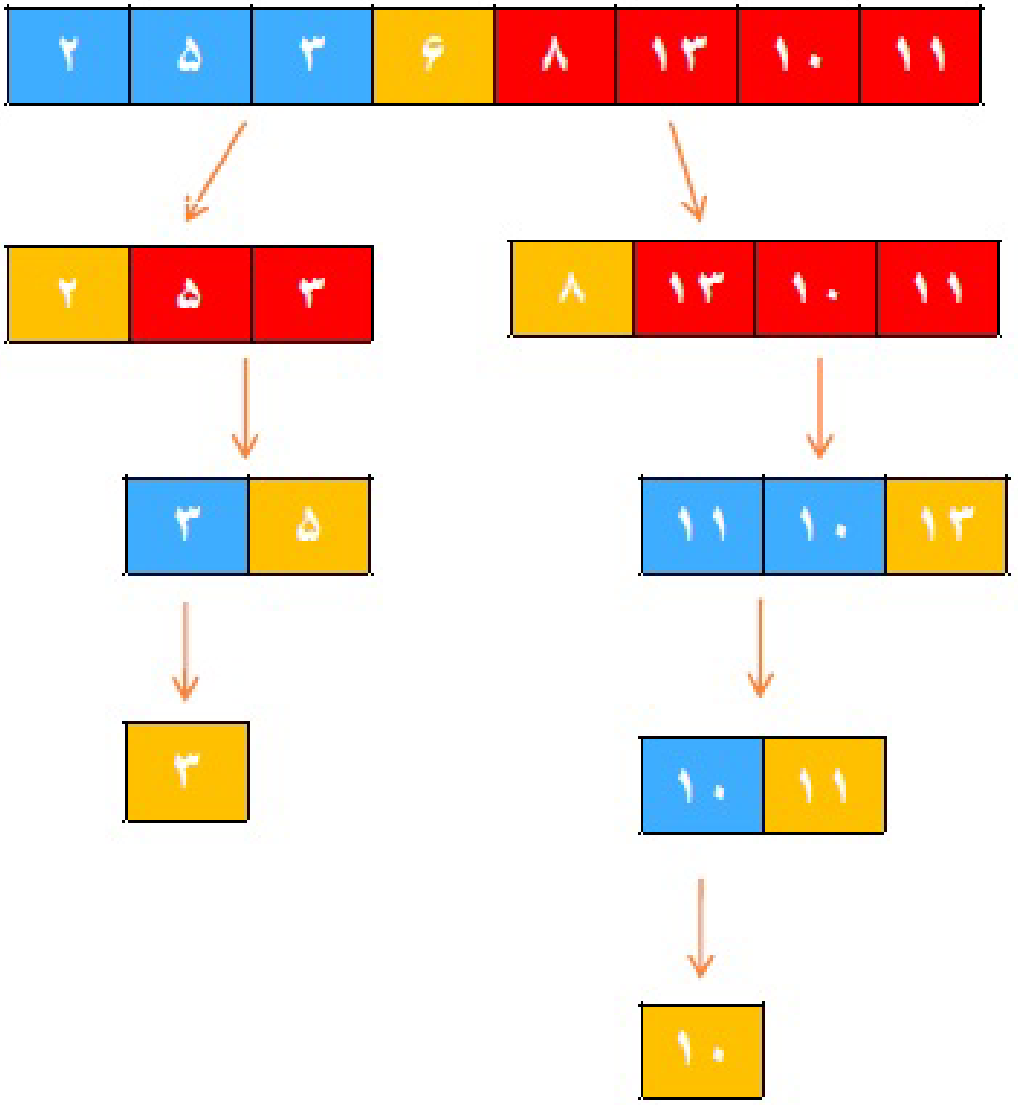
افراز (Partitioning) سطح دو



افراز (Partitioning) سطح سه



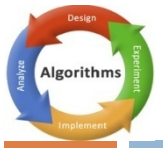
افراز (Partitioning) سطح چهار



یک مثال دیگر از رویه افراز

15	22	13	27	12	10	20	25

i	j	$S[1]$	$S[2]$	$S[3]$	$S[4]$	$S[5]$	$S[6]$	$S[7]$	$S[8]$	
—	—	15	22	13	27	12	10	20	25	← Initial values
2	1	**15**	**22**	13	27	12	10	20	25	
3	2	**15**	22	**13**	27	12	10	20	25	
4	2	**15**	13	22	**27**	12	10	20	25	
5	3	**15**	13	22	27	**12**	10	20	25	
6	4	**15**	13	12	27	22	**10**	20	25	
7	4	**15**	13	12	10	22	27	**20**	25	
8	4	**15**	13	12	10	22	27	20	**25**	
—	4	10	13	12	15	22	27	20	25	← Final values



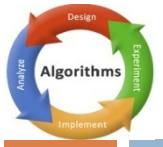
پیچیدگی زمانی الگوریتم افراز(Partitioning)درهمه حالات

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

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

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

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



پیچیدگی زمانی مرتب سازی سریع: بدترین حالت

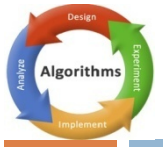
- زمان اجرا به متوازن بودن یا نبودن افراز بستگی دارد.
- بدترین حالت زمانی است که آرایه از قبل مرتب باشد.
- عمل اصلی: مقایسه $S[i]$ با $pivotitem$ در رویه $partition$
- اندازه ورودی: n، اندازه آرایه S
- پیچیدگی زمانی:

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

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

- حل:

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



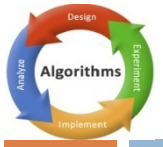
پیچیدگی زمانی مرتب سازی سریع: حالت متوسط

- عمل اصلی: مقایسه $S[1]$ با $pivotitem$ در رویه $partition$
- اندازه ورودی: n، اندازه آرایه S
- پیچیدگی زمانی:

$$A(n) = \sum_{p=1}^{n} \overbrace{\frac{1}{n}}^{\Pr obability\ pivotpoint\ is\ p} \underbrace{[A(p-1) + A(n-p)]}_{\substack{Average\ sort\ pivotpoint \\ time\ subarrays\ is \\ to\ when\ p}} + \underbrace{n-1}_{\substack{Time\ to \\ partition}}$$

- حل:

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



پیچیدگی زمانی مرتب سازی سریع: بهترین حالت

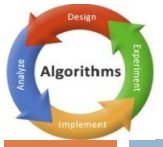
- بهترین حالت زمانی است که عنصر محوری لیست را به دو زیرلیست مساوی افراز کند. در این حالت داریم:

$$T(n) = 2T(n/2) + n \qquad B(n) = \Theta(n\lg n)$$

- مثال : آرایه زیر را به شیوه مرتب سازی سریع، مرحله به مرحله مرتب نمایید.

- $A = \{20,12, 3,14,18,11,15, 7\}$

- اگر در مرتب سازی سریع بجای تقسیم آرایه به دو زیر آرایه، با یک عنصر لولا، آرایه را به سه زیر آرایه با دو عنصر لولا، تقسیم نماییم، رابطه بازگشتی به چه صورت خواهد شد؟ آیا بهبودی در زمان اجرای ایم الگوریتم حاصل می شود یا خیر ؟



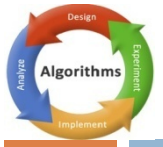
ضرب ماتریس ها

- مسئله ضرب دو ماتریس را در نظر بگیرید به عنوان مثال حاصل ضرب دو ماتریس 2×2 یک ماتریس 2×2 است که به صورت زیر نشان داده می‌شود:

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} * \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \qquad c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \qquad i = 1,2 \quad j = 1,2 \quad , \quad n = 2$$

- ضرب دو ماتریس $n \times n$ احتیاج به n^3 عمل ضرب و $n^2(n-1)$ عمل جمع خواهد داشت. زیرا ماتریس حاصلضرب، n^2 عنصر خواهد داشت که هر عنصر آن با n عمل ضرب و $n-1$ عمل جمع بدست می آید. پس زمان اجرای این الگوریتم برابر تعداد ضرب‌ها و از مرتبه $\Theta(n^3)$ خواهد بود.

$$\begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} * \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix} = \begin{bmatrix} a_{00} * b_{00} + a_{01} * b_{10} & a_{00} * b_{01} + a_{01} * b_{11} \\ a_{10} * b_{00} + a_{11} * b_{10} & a_{10} * b_{01} + a_{11} * b_{11} \end{bmatrix}$$

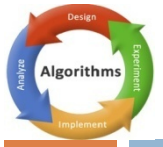


الگوریتم ضرب ماتریس ها به روش استراسن

```
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 طبق تعریف:
  - تعداد ضرب ها: $T(n) = n^3$
  - تعداد جمع ها: $T(n) = n^3 - n^2$

- الگوریتم استراسن برای ضرب ماتریس ها (1969)
  - پیچیدگی بهتر از درجه سوم (جمع و ضرب)



روش استراسن

- برای محاسبه:

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

- تعریف می کنیم:

$$m_1 = (a_{11} + a_{22})(b_{11} + b_{22})$$

$$m_2 = (a_{21} + a_{22})b_{11}$$

$$m_3 = a_{11}(b_{12} - b_{22})$$

$$m_4 = a_{22}(b_{21} - b_{11})$$

$$m_5 = (a_{11} + a_{12})b_{22}$$

$$m_6 = (a_{21} - a_{11})(b_{11} + b_{22})$$

$$m_7 = (a_{12} - a_{22})(b_{21} + b_{22})$$

روش	تعداد ضرب های ضرب دو ماتریس ۲ در ۲	تعداد جمع های ضرب دو ماتریس ۲ در ۲
سنتی	۸	۴
استراسن	۷	۱۸

- آنگاه :

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

برای ماتریس های بزرگ ...

- تقسیم ماتریس ها:

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

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

$n/2$ (top) and $n/2$ (left)

$$C_{21} = M_2 + M_4$$

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

یک مثال

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

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

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

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

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;
}
```

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

تحلیل ضرب ماتریس ها به روش استراسن

❑ پیچیدگی زمانی تعداد ضرب‌ها در الگوریتم استراسن

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

❑ پیچیدگی زمانی تعداد ضرب‌ها و جمع و تفریق ها در الگوریتم استراسن

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

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

روش	تعداد ضرب های ضرب دو ماتریس n در n	تعداد جمع و تفریق های ضرب دو ماتریس n در n
سنتی	n^3	$n^3 - n^2$
استراسن	$n^{2.81}$	$6n^{2.81} - 6n^2$

ضرب اعداد صحیح بزرگ

- مسئله ضرب دو عدد بزرگ n رقمی صحیح را در نظر بگیرید:

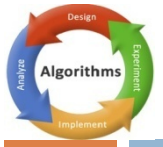
آرایه هایی شامل ارقام مانند:

A = 12345678901357986429 B = 87654321284820912836

- الگوریتمی که ابتدا به ذهن می رسد همان ضرب دبیرستان است:

$$\begin{array}{r} a_1\ a_2 \dots\ a_n \\ \times\quad b_1\ b_2 \dots\ b_n \\ \hline (d_{10})\ d_{11}\ d_{12} \dots\ d_{1n} \\ (d_{20})\ d_{21}\ d_{22} \dots\ d_{2n} \\ + \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots \\ \hline (d_{n0})\ d_{n1}\ d_{n2} \dots\ d_{nn} \end{array}$$

- بنابراین زمان این الگوریتم زمان لازم برای انجام تعداد ضرب ها خواهد بود که برای ضرب دو عدد بزرگ n رقمی صحیح n^2 ضرب خواهیم داشت پس زمان اجرای این الگوریتم از مرتبه $\Theta(n^2)$ خواهد بود.



اولین راهکار D & C برای ضرب اعداد صحیح بزرگ

- در حالت کلی اگر A و B دو عدد n رقمی باشند اگر $A = A_1A_2$ و $B = B_1B_2$ بطوریکه $A_1, A_2,$ B_1, B_2 همگی اعداد $n/2$ رقمی باشند، داریم:

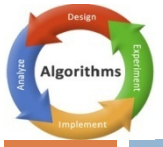
$$A = A_1 * 10^{n/2} + A_2 \quad \text{و} \quad B = B_1 * 10^{n/2} + B_2$$

$$A * B = A_1 * B_1 \cdot 10^{n} + (A_1 * B_2 + A_2 * B_1) \cdot 10^{n/2} + A_2 * B_2$$

$A = 2135$ و $B = 4014$

$$A = (21 \cdot 10^2 + 35), \quad B = (40 \cdot 10^2 + 14)$$

$$A*B = (21 \cdot 10^2 + 35)*(40 \cdot 10^2 + 14) = 21*40 \cdot 10^4 + (21*14 + 35*40) \cdot 10^2 + 35*14$$



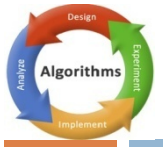
ضرب اعداد صحیح بزرگ

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

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

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 D & C برای ضرب اعداد صحیح بزرگ

```
large_integer prod(large_integer u, large_integer v)
{
    large_integer 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 = 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);
    }
}
```

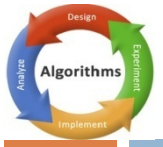
پیچیدگی زمانی الگوریتم D & C برای ضرب اعداد صحیح بزرگ

- **عمل اصلی:** دستکاری یک رقم دهدهی در یک عدد صحیح بزرگ هنگام جمع کردن، تفریق کردن، یا انجام اعمال تقسیم بر 10^m ، ضرب در 10^m و محاسبه باقیمانده بر 10^m
- همه عملیات بالا دارای پیچیدگی زمانی خطی می باشند.
- **اندازه ورودی:** n، تعداد ارقام هر یک از دو عدد
- **پیچیدگی زمانی:**

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

- **حل:** با استفاده از قضیه اصلی

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



بهبود راهکار D & C برای ضرب اعداد صحیح بزرگ

- با فرضیات قبلی همانطور که ملاحظه شد:

$$A = A_1 * 10^{n/2} + A_2 \qquad \text{و} \qquad B = B_1 * 10^{n/2} + B_2$$

$$A * B = A_1 * B_1 \cdot 10^{n} + (A_1 * B_2 + A_2 * B_1) \cdot 10^{n/2} + A_2 * B_2$$

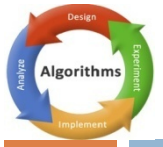
$$(A_1 * B_2 + A_2 * B_1) = (A_1 + A_2) * (B_1 + B_2) - A_1 * B_1 - A_2 * B_2$$

پس:

$$A*B = A_1 * B_1 \cdot 10^{n} + ((A_1 + A_2) * (B_1 + B_2) - A_1 * B_1 - A_2 * B_2) \cdot 10^{n/2} + A_2 * B_2$$

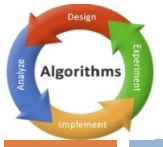
- بنابراین تنها باید سه حاصل ضرب زیر را محاسبه کنیم:

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



الگوریتم بهبود یافته D & C برای ضرب اعداد صحیح بزرگ

```
large_integer prod2(large_integer u, large_integer v)
{
    large_integer x, y, w, z, r, p, q;
    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 = u rem 10^m;
        w = v divide 10^m; z = v rem 10^m;
        r = prod2(x + y, w + z);
        p = prod2(x, w);
        q = prod2(y, z);
        return p × 10^2m + (r-p-q) × 10^m+q;
    }
}
```



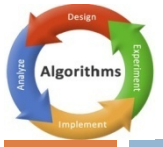
پیچیدگی زمانی الگوریتم برای ضرب اعداد صحیح بزرگ

- عمل اصلی: دستکاری یک رقم دهدهی در یک عدد صحیح بزرگ هنگام جمع کردن، تفریق کردن، یا انجام اعمال تقسیم بر 10^m ، ضرب در 10^m و محاسبه باقیمانده بر 10^m
- اندازه ورودی: n، تعداد ارقام هر یک از دو عدد
- پیچیدگی زمانی:

$$\begin{cases} 3W(\frac{n}{2}) + cn \le W(n) \le 3W(\frac{n}{2}+1) + cn & for \quad n > s, \quad n \quad a \quad power \quad of \quad 2 \\ W(s) = 0 \end{cases}$$

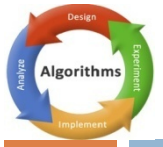
- حل:

$$W(n) \in \Theta(n^{\lg 3}) = \Theta(n^{1.58})$$



تعیین مقادیر آستانه

- فرآیند بازگشتی از لحاظ زمانی به قدری سر بار نیاز دارد
- مثال: ممکن است برای مرتب سازی ۸ کلید، الگوریتم مرتب سازی تعویضی ($\Theta(n^2)$)سریعتر از الگوریتم مرتب سازی سریع ($\Theta(n\lg n)$) باشد.
- منظور از مقدار آستانه t برای یک الگوریتم بازگشتی ، اندازه نمونه ای است که به ازای تمام نمونه های کوچکتر از آن، بهتر است به جای فراخوانی بازگشتی ، از یک الگوریتم دیگر استفاده شود.
- در تعیین مقدار آستانه ، بدترین حالت الگوریتم را مورد بررسی قرار می دهیم (در واقع سعی داریم رفتار بدترین حالت را بهینه نماییم.)



مثال: مرتب سازی ادغامی

- پیچیدگی زمانی در بدترین حالت: (بهینه سازی در بدترین حالت)

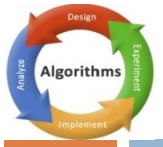
$$W(n) = W(\left\lfloor \frac{n}{2} \right\rfloor) + W(\left\lceil \frac{n}{2} \right\rceil) + n - 1$$

- اگر زمان لازم برای تقسیم و ترکیب یک نمونه به اندازه n در یک کامپیوتر مفروض برابر با 32n میکروثانیه باشد، آنگاه:

$$W(n) = W(\left\lfloor \frac{n}{2} \right\rfloor) + W(\left\lceil \frac{n}{2} \right\rceil) + 32n\ \mu s$$

- بنابراین:

$$W(n) = 2W(n/2) + 32n\ \mu s$$
$$W(1) = 0$$



تعیین مقدار آستانه برای مرتب سازی ادغامی

- تعیین t

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

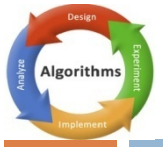
- حل معادله روبرو

$$W\left(\left\lfloor\frac{t}{2}\right\rfloor\right)+W\left(\left\lceil\frac{t}{2}\right\rceil\right)+32t=\frac{t(t-1)}{2}$$

$$W\left(\left\lfloor\frac{t}{2}\right\rfloor\right)=\frac{\lfloor t/2\rfloor(\lfloor t/2\rfloor-1)}{2}$$

$$W\left(\left\lceil\frac{t}{2}\right\rceil\right)=\frac{\lceil t/2\rceil(\lceil t/2\rceil-1)}{2}$$

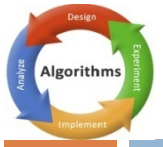
$$\frac{\lfloor t/2\rfloor(\lfloor t/2\rfloor-1)}{2}+\frac{\lceil t/2\rceil(\lceil t/2\rceil-1)}{2}+32t=\frac{t(t-1)}{2}$$



تعیین مقدار آستانه برای مرتب سازی ادغامی

$$\frac{\lfloor t/2 \rfloor(\lfloor t/2 \rfloor - 1)}{2} + \frac{\lceil t/2 \rceil(\lceil t/2 \rceil - 1)}{2} + 32t = \frac{t(t-1)}{2}$$

- اگر t زوج باشد، آنگاه $t = 128$
- اگر t فرد باشد؛ آنگاه $t = 128.008$
- بنابراین، مقدار آستانه بهینه برابر $t = 128$ می باشد



مثال تعیین مقدار آستانه

- پیچیدگی زمانی یک الگوریتم تقسیم و حل بر روی یک کامپیوتر خاص:

$$T(n) = 3T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 16n \quad \mu s$$

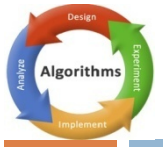
- پیچیدگی زمانی یک الگوریتم تکراری برای حل نمونه ای به اندازه n:

$$T(n) = n^2 \quad \mu s$$

- تعیین مقدار آستانه بهینه t:

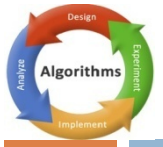
$$3T\left(\left\lceil \frac{t}{2} \right\rceil\right) + 16t = t^2 \quad \& \quad T\left(\left\lceil \frac{T}{2} \right\rceil\right) = \left\lceil \frac{t}{2} \right\rceil^2 \Rightarrow$$

$$3\left\lceil \frac{t}{2} \right\rceil^2 + 16t = t^2$$



مثال تعیین مقدار آستانه

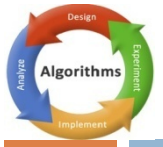
- برای تعیین t باید معادله زیر را حل نمود:
- الف) اگر t زوج باشد، آنگاه $t=64$
- ب) اگر t فرد باشد، آنگاه $t=70.04$
- چون دو مقدار برابر نیستند، آستانه بهینه وجود ندارد؛ یعنی:
  - اگر n یک عدد صحیح زوج بین ۶۴ و ۷۰ باشد، بهتر است یک بار دیگر تقسیم شود
  - اگر n یک عدد صحیح فرد بین ۶۴ و ۷۰ باشد، فراخوانی الگوریتم جانشین کارآیی بیشتری دارد
  - اگر n کوچکتر از ۶۴ باشد، همواره فراخوانی الگوریتم جانشین کارآیی بیشتری دارد
  - اگر n بزرگتر از ۷۰ باشد، همواره تقسیم دوباره نمونه کارآتر خواهد بود.



مثال تعیین مقدار آستانه

- مقایسه کارآیی الگوریتم بازگشتی و جانشین به ازاء مقادیر مختلف n:

n	n^2	
62	3844	3875
63	3969	4080
64	4069	4096
65	4225	4307
68	4624	4556
69	4761	4779
70	4900	4795
71	5041	5024



مواقعی که نباید از تقسیم و حل استفاده کنیم

- یک نمونه به اندازه n به دو یا چند نمونه تقسیم شود به طوری که اندازه هر یک از این نمونه ها تقریبا برابر اندازه نمونه اصلی باشد. ← نمایی
  - مثال: دنباله فیبونانچی
  - استثناء : مساله برج های هانوی (تمرین ۱۷)
- یک نمونه به اندازه n تقریبا به n نمونه با اندازه های n/c تقسیم شود به طوری که c یک عدد ثابت باشد. ← $\Theta(n^{\lg n})$