فصل هشتم

مرتب سازی

مدرس:
فرشید شیرافکن

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

تعریف

الگوریتم های مرتب سازی را از نظر نحوه مرتب سازی داده ها می توان به صورت زیر دسته بندی کرد:

۱- مرتب سازی مبتنی بر مقایسه (comparison based)
با مقایسه بین کلیدهای عناصر، ترتیب نسبی آنها پیدا می شود.

۲- مرتب سازی غیر مقایسه ای(خطی)
بدون مقایسه کلیدهای عناصر با هم، عمل مرتب سازی انجام می شود.

روش ها

روش های مقایسه ای

۱- حبابی (buble)
۲- انتخابی (selection)
۳- درجی (insertion)
۴- ادغامی (merge)
۵- سریع (quick)
۶- هیپ (heap)
۷- درختی (tree)
۸- شل (shell)

روش های غیر مقایسه ای

۱- شمارشی (count)
۲- مبنایی (radix sort)
۳- سطلی (bucket sort)
۴- لانه کبوتری

مرتب سازی حبابی

در این روش با چند بار حرکت در طول آرایه، یک عنصر با عنصر بعدی مقایسه شده و در صورت لزوم جا به جا می‌شوند.

First Pass:

(5 1 4 2 8)
(1 5 4 2 8)
(1 4 5 2 8)
(1 4 2 5 8)
(1 4 2 5 8)

Second Pass:

(1 4 2 5 8)
(1 4 2 5 8)
(1 2 4 5 8)
(1 2 4 5 8)

مثال(مرتب سازی حبابی)

First Pass:

8 4 3 2
4 8 3 2
4 3 8 2
4 3 2 8

Second Pass:

4 3 2 8
3 4 2 8
3 2 4 8

Thired Pass:

3 2 4 8
2 3 4 8

برنامه مرتب سازی حبابی

```
void  bubbleSort(int  arr[ ], int  n) {
    int i, j; bool  swapped;
    for (i = 0; i < n-1; i++) {
      swapped = false;
      for (j = 0; j < n-i-1; j++) {
         if (arr[j] > arr[j+1])
             { swap(&arr[j], &arr[j+1]);
               swapped = true; }
    }
    if (swapped == false)  break;
  }
}
```

مرتب‌سازی انتخابی (Selection Sort)

قراردادن کوچک‌ترین عنصر آرایه در مکان اول
قرار دادن کوچکترین عنصر دوم در مکان دوم
و ...

آرایه اولیه	15	21	1	12	17	30	10
مرحله اول	15	21	10	12	17	30	1
مرحله دوم	15	21	30	12	17	10	1
مرحله سوم	15	21	30	17	12	10	1
مرحله چهارم	17	21	30	15	12	10	1
مرحله پنجم	30	21	17	15	12	10	1
مرحله ششم	30	21	17	15	12	10	1

تابع مرتب سازی انتخابی

```
void  selectionSort(int   arr[] , int  n) {
   int i, j,
   m;
   for (i = 0; i < n-1; i++) {
       m = i;
       for (j = i+1; j < n; j++)
         if (arr[j] < arr[m])
             m = j;
    swap(&arr[m], &arr[i]);
   }
}
```

مرتب سازی درجی

اعداد را یکی یکی در محل درست درج می کنیم:

8, 5, 3, 21, 12, 11, 6, 2

5, 8, 3, 21, 12, 11, 6, 2

3, 5, 8, 21, 12, 11, 6, 2

3, 5, 8, 21, 12, 11, 6, 2

3, 5, 8, 12, 21, 11, 6, 2

3, 5, 8, 11, 12, 21, 6, 2

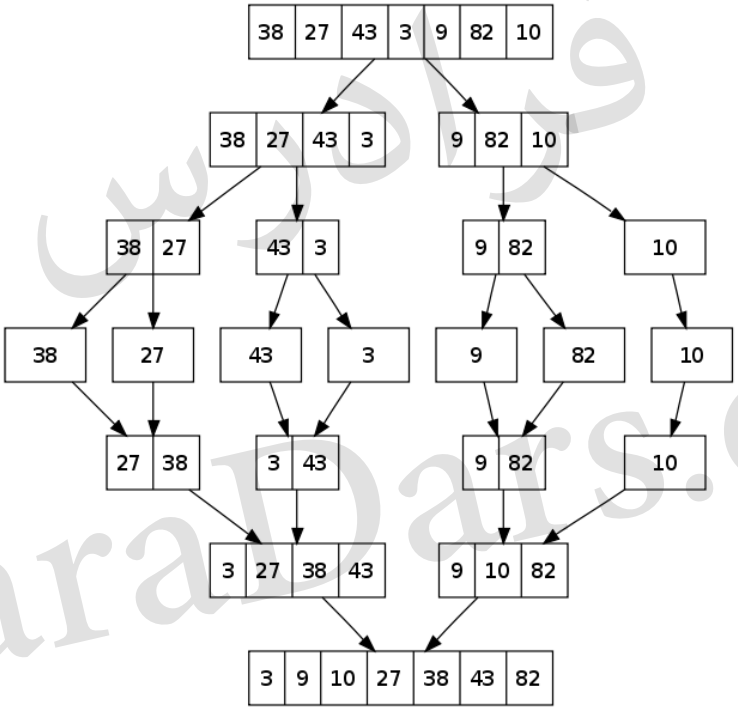
3, 5, 6, 8, 11, 12, 21, 2

2, 3, 5, 6, 8, 11, 12, 21

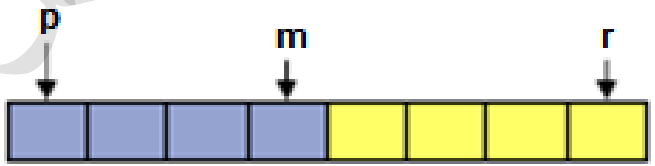
تابع

```
void   insertionSort(int   arr[], int  n) {
  int i, key, j;
  for (i = 1; i < n; i++){
     key = arr[i];
     j = i-1;
     while ( j >= 0  &&  arr[j] > key ) {
         arr[j+1] = arr[j];
         j = j-1;
       }
    arr[j+1] = key;
  }
}
```

مرتب سازی ادغامی (Merge Sort)



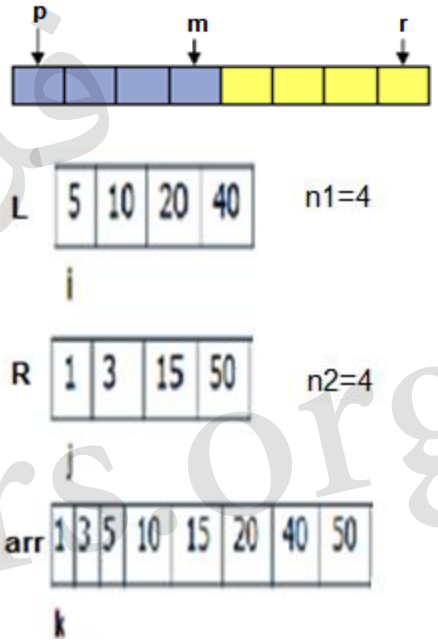
```
void mergeSort ( int  arr[ ] , int p , int r) {
    if (p < r)
    {
        int m = (p+r)/2;
        mergeSort (arr , p , m);
        mergeSort (arr , m+1 , r);
        merge(arr , p , m , r);
    }
}
```



$$T(n) = 2T(\frac{n}{2}) + n - 1$$
$$\Rightarrow T(n) = n\lg n - (n-1)$$
$$\in \theta(n\lg n)$$

ادغام دو آرایه مرتب

```
void merge(int  arr[ ] , int p , int m , int r) {
    int  i, j, k;
    int  n1 = m - p + 1;
    int  n2 =  r - m;
    int  L[n1] , R[n2];
    for (i = 0; i < n1; i++)    L[i] = arr[p + i];
    for (j = 0; j < n2; j++)   R[j] = arr[m + 1+ j];
    i = 0;  j = 0;  k = 0;
    while (i < n1  &&  j < n2)  {
       if (L[i] <= R[j])   { arr[k] = L[i];  i++; }
       else { arr[k] = R[j];  j++; }
       k++;
    }
     while (i < n1) {  arr[k] = L[i];  i++; k++; }
     while (j < n2) {  arr[k] = R[j]; j++;  k++;}
}
```



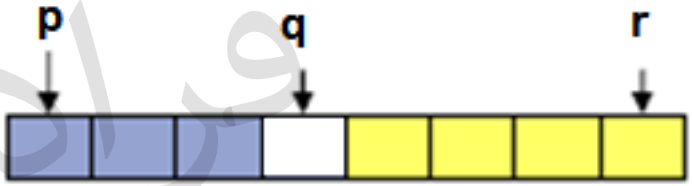
مقایسه

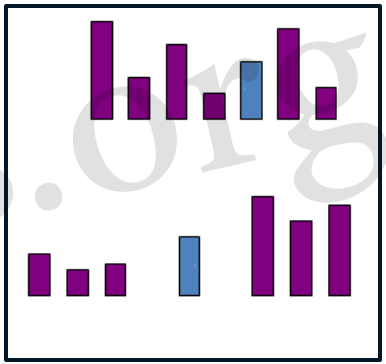
- مقایسه چند نوع مرتب سازی

	Best	Average	Worst	Additional Memory
Selection Sort	$O(N^2)$	$O(N^2)$	$O(N^2)$	O(1)
Insertion Sort	O(N)	$O(N^2)$	$O(N^2)$	O(1)
Merge Sort	O(NlogN)	O(NlogN)	O(NlogN)	O(N)

مرتب سازی سریع (Quick sort)

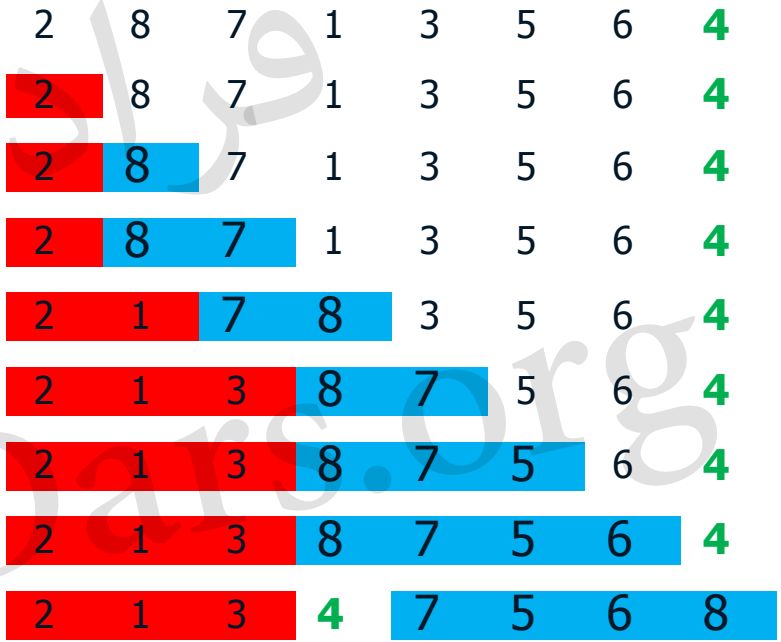
```
QuickSort (A , p , r)
{
  if ( p < r )
   {
     q = Partition (A , p , r) ;
     QuickSort (A , p , q-1) ;
     QuickSort (A , q+1, r ) ;
   }
}
```





الگوریتم پارتیشن بندی

```
Partition (A , p , r ){
   pivot= A[r];
   i = p-1;
   for (j = p ; j<= r-1 ; j++) {
      if  (A[j] <= pivot){
         i++; swap ( A[i] , A[j] ) ;
       }
    }
   swap ( A[i+1] , A[r] ) ;
   return  i+1 ;
}
```



مثال(مرتب سازی سریع)

عنصر اول به عنوان محور در نظر گرفته شده است.

8, 5, 3, 21, 12, 11, 6, 2

5, 3, 6, 2, 8 21, 12, 11

3, 2 5 6 8 12, 11 21

2 3 5 6 8 11 12 21

عملکرد مرتب سازی سریع

حالت خوب و میانگین :

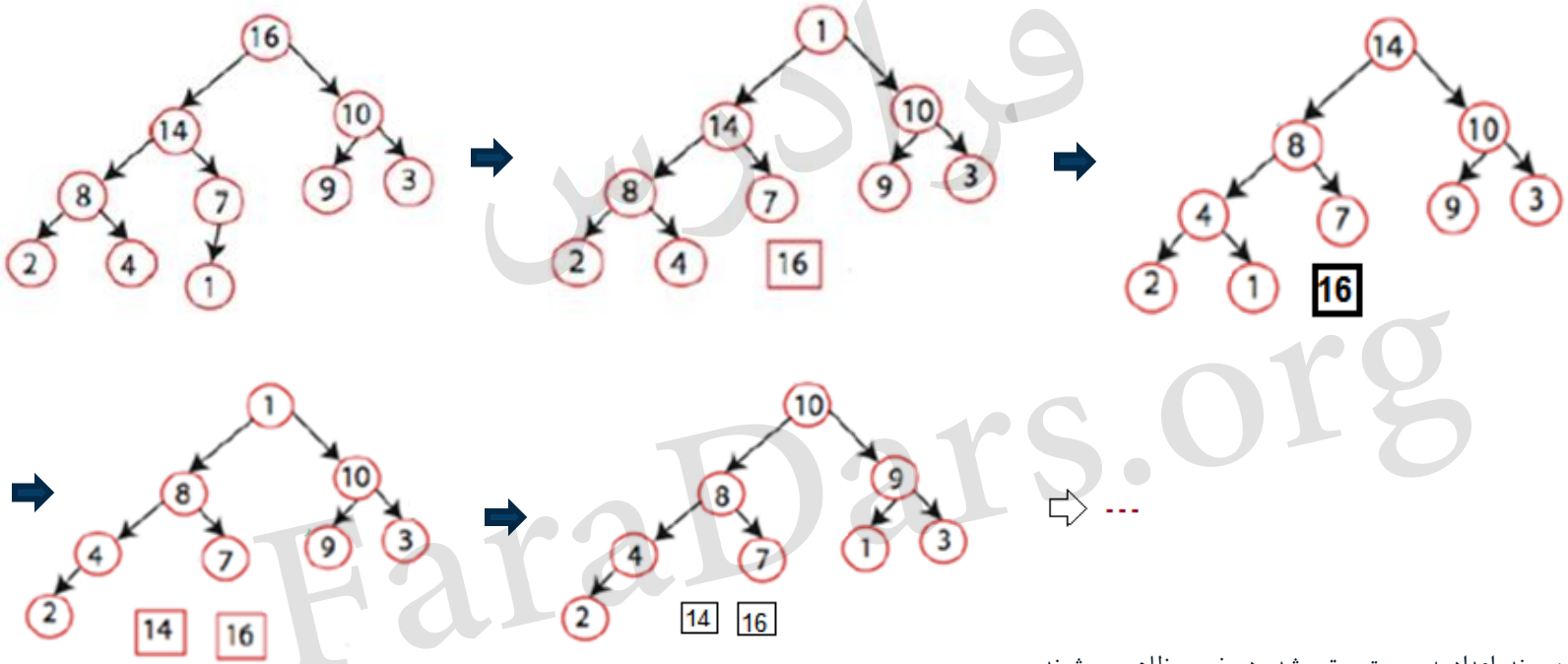
$$T(n) = T(n/2) + T(n/2)+O(n) \Rightarrow O(nlogn)$$

حالت بد:

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.

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

مرتب سازی هیپ



با ادامه این روند، اعداد به صورت مرتب شده در خروج ظاهر می شوند.

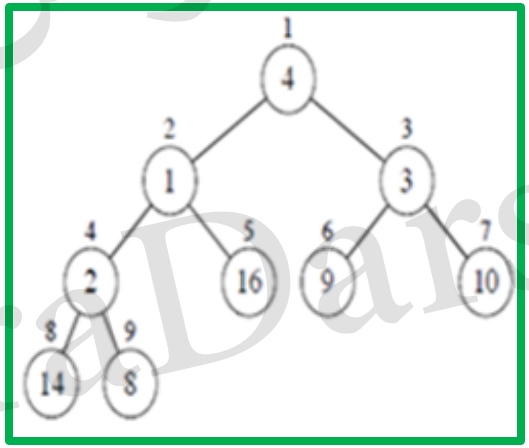
تابع مرتب سازی هیپ

```
Heapsort ( A , n )
{
   BuildMaxheap(A , n);
   for( i=n ; i>=2 ; i--)
   {
      exchange( A[1] , A[i]);
      MaxHeapify(A , 1 , i-1 );
   }
}
```

تابع ساختن درخت هیپ

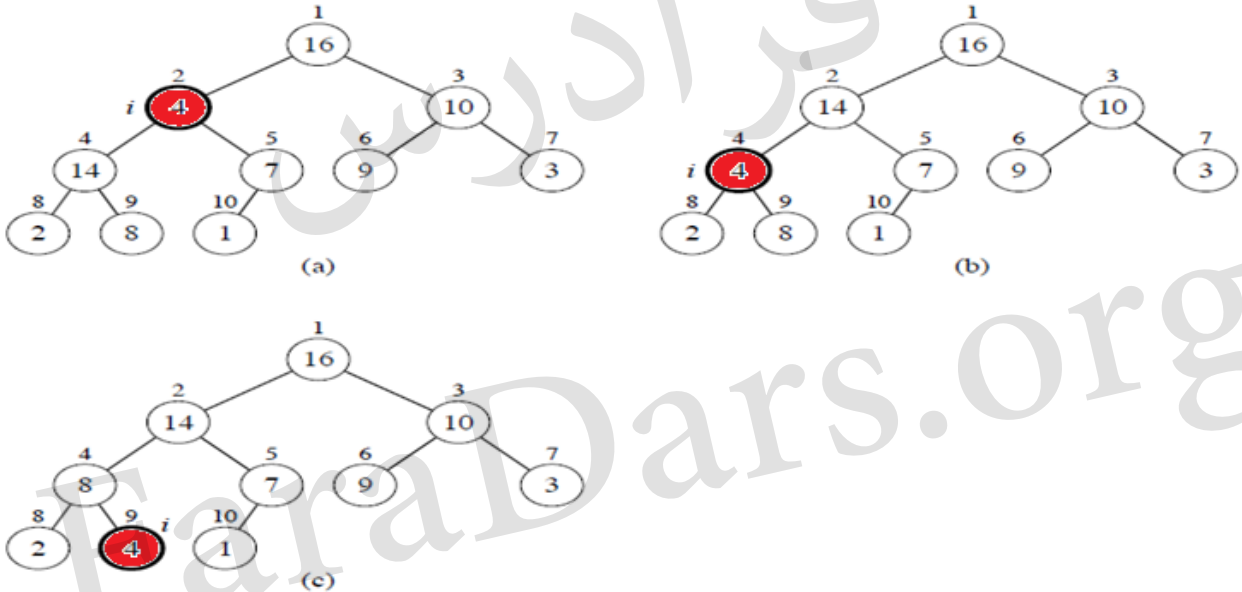
```
BuildMaxheap(A,n){
   for ( i=n/2 ;  i>=1 ; i--)
       Maxheapify(A , i , n);
}
```

	1	2	3	4	5	6	7	8	9
A	4	1	3	2	16	9	10	14	8



Maxheapify(A ,2, 10)

مثال

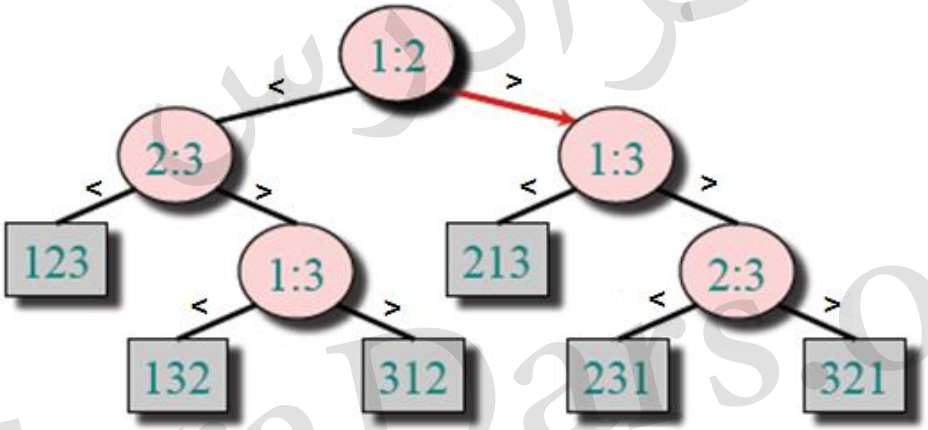


Maxheapify تابع

```
Maxheapify ( A , i , n ) {
   L = Left( i );    r = Right( i );
   if ( L <=n  &&  A[L] > A[i] )
            largest = L;   else   largest = i ;
   if ( r <=n  &&  A[r]  > A[largest] )
            largest = r ;
   if ( largest != i )
            exchange( A[i] , A[largest] );
   Maxheapify ( A , largest , n )
}
```

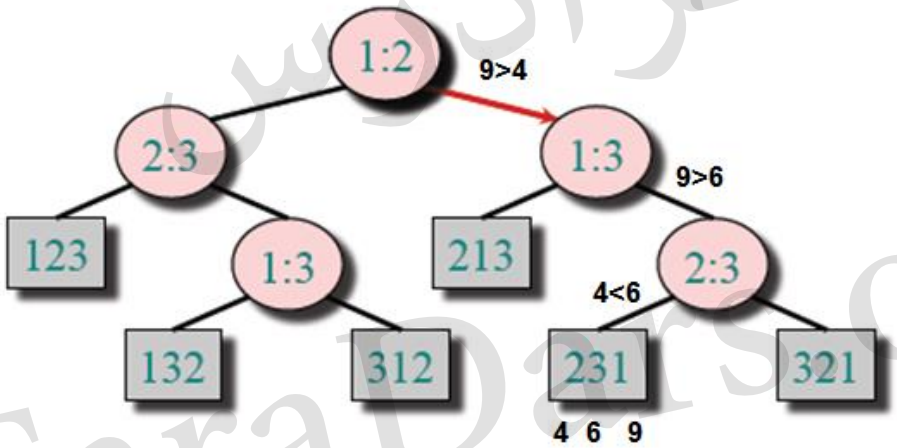
درخت تصمیم گیری

درخت تصمیم گیری برای مرتب سازی ۳ عنصر :



درخت تصمیم گیری

درخت تصمیم گیری برای مرتب سازی <9,4,6>



این اسلاید ها بر مبنای نکات مطرح شده در فرادرس
«مجموعه فرادرس های ساختمان داده ها»
تهیه شده است.

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

faradars.org/fvds9402