Unit-6 Sorting

Sorting

Sorting is among the most basic problems in algorithm design. We are given a sequence of items, each associated with a given key value. The problem is to permute the items so that they are in increasing (or decreasing) order by key. Sorting is important because it is often the first step in more complex algorithms. Sorting algorithms are usually divided into two classes:

Internal sorting algorithms, which assume that data is stored in an array in main memory, and **external sorting algorithm**, which assume that data is stored on disk or some other device that is best accessed sequentially. We will only consider internal sorting. Sorting algorithms often have additional properties that are of interest, depending on the application. Here are two important properties.

In brief the sorting is a process of arranging the items in a list in some order that is either ascending or descending order.

Let a[n] be an array of n elements a0,a1,a2,a3.....,a n-1 in memory. The sorting of the array a[n] means arranging the content of a[n] in either increasing or decreasing order.

i.e. a0<=a1<=a2<=a3<.=.....<=a n-1

consider a list of values: 2 ,4 ,6 ,8 ,9 ,1 ,22 ,4 ,77 ,8 ,9

After sorting the values: 1, 2, 4, 4, 6, 8, 8,9 , 9 , 22, 77

Sorting Technique:

Insertion Sort

The insertion sort inserts each element in proper place if there is n element in array and we place each element of array at proper place in the previously sorted element list.

Algorithm:-

Consider N elements in the array are

Pass 1: arr[0] is already sorted because of only one element.

Pass 2: arr[0] is inserted before or after arr[1]. So arr[0] & arr[1] are sorted.

Pass 3: arr[2] is inserted before arr[0], in between arr[0] & arr[1] or after arr[0] : so arr[0] , arr[1] & arr [2] are sorted.

Pass 4: arr[3] is inserted in to it's proper place in array arr[0], arr[1], arr[2], arr[3] & are sorted.

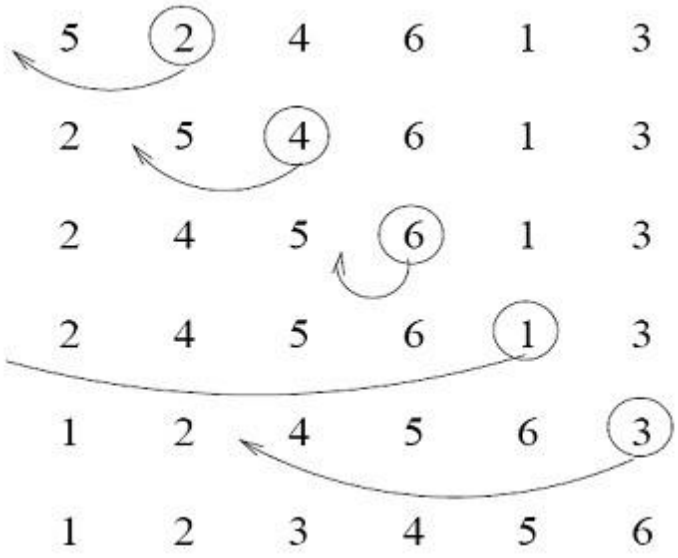
……

……

Pass N: arr[N-1] is inserted in to it's proper place in array arr[0], arr[1], ……. Arr[N-1]. So arr[0], arr[N-1] are sorted.

Example: Arrange the following data items using insertion sort:

5, 2, 4, 6, 1, 3



Thus sorted elements are: 1,2,3,4,5,6

Selection Sort

Selection sort is the selection of an element & keeping it in sorted order. Let us take an array arr[0]……..arr[N-1]. First find the position of smallest element from arr[0] to arr [n-1]. Then interchange the smallest element from arr[1] to arr[n-1], then interchanging the smallest element with arr[1]. Similarly, the process will be for arr[0] to arr[n-1] & so on.

Algorithm:-

Pass 1:- search the smallest element for arr[0] ……..arr[N-1].

- Interchange arr[0] with smallest element

Result : arr[0] is sorted.

Pass 2:- search the smallest element from arr[1],…….arr[N-1]

- Interchange arr[1] with smallest element

Result: arr[0], arr[1] is sorted.

……

……

Pass N-1:-

- search the smallest element from arr[N-2] & arr[N-1]

- Interchange arr[N-1] with smallest element

Result: arr[0]……. Arr[N-1] is sorted.

Q. Show all the passes using selection sort.

75 35 42 13 87 27 64 57

Pass 1 75 35 42 13 87 27 64 57

Pass 2 13 35 42 75 87 27 64 57

Pass 3 13 27 42 75 87 35 64 57

Pass 4 13 27 35 75 87 42 64 57

Pass 5 13 27 35 42 87 75 64 57

Pass 6 13 27 35 42 57 75 64 87

Pass 7 13 27 35 42 57 64 75 87

Thus sorted elements are: 13 27 35 42 57 64 75 87.

Bubble sort

In bubble sort each element is compared with its adjacent element. If the 1st element is large than the second one then the position of the elements are interchanged otherwise it is not changed.

Algorithim:

If N elements are given in memory then for sorting we do the following steps:-

Pass:-

1. first compare the 1st element and 2nd element of array. If 1st < 2nd then compare

the 2nd with 3rd.

2. if 2nd > 3rd.

Then interchange the value of 2nd & 3rd.

now compare the value of 3rd with 4th.

similarly compare until N-1th element is compared with nth element.

Now, highest value element is reached at the Nth place.

Elements will be compared until N-1 elements.

Q. Show all the passes using bubble sort.

13, 32, 20, 62, 68, 52, 38, 46

Pass 1:

Pass I. 13 32 20 62 68 52 38 46

Pass II 32 > 20 -interchange.

13 20 32 62 68 52 38 46

Pass III 32<62 - no change.

13 20 32 62 68 52 38 46

Pass IV 62<68 - no change.

13 20 32 62 68 52 38 46

Pass V 68> 52 - interchange.

13 20 32 62 52 68 38 46

Pass VI 68 >38 - interchange

13 20 32 62 52 38 68 46

Pass V II 68>46 -interchange

13 20 32 62 52 38 46 68

Pass 2:

Pass I 62>52 -interchange

13 20 32 52 62 38 46 68

Pass II 62>38 -interchange

13 20 32 52 38 62 46 68

Pass III 62>46 -interchange

13 20 32 52 38 46 62 68

Pass IV 62<68 - no change

Pass 3:

Pass I 52>38 -interchange

13 20 32 38 52 46 62 68

Pass II 52>46 -interchange

13 20 32 38 46 52 62 68

Pass III 62<68 - no change

Thus sorted elements are: 13 20 32 38 46 52 62 68.

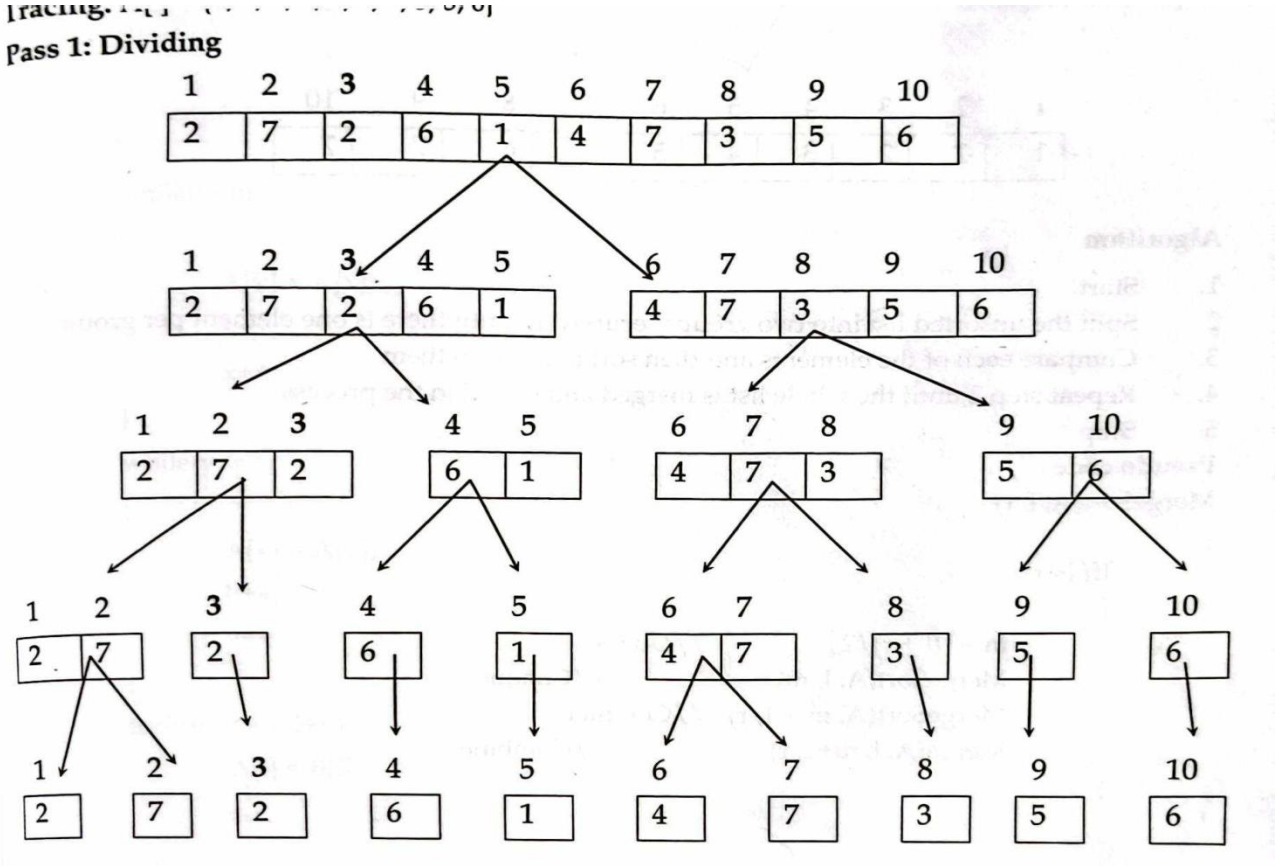
Merge Sort:-

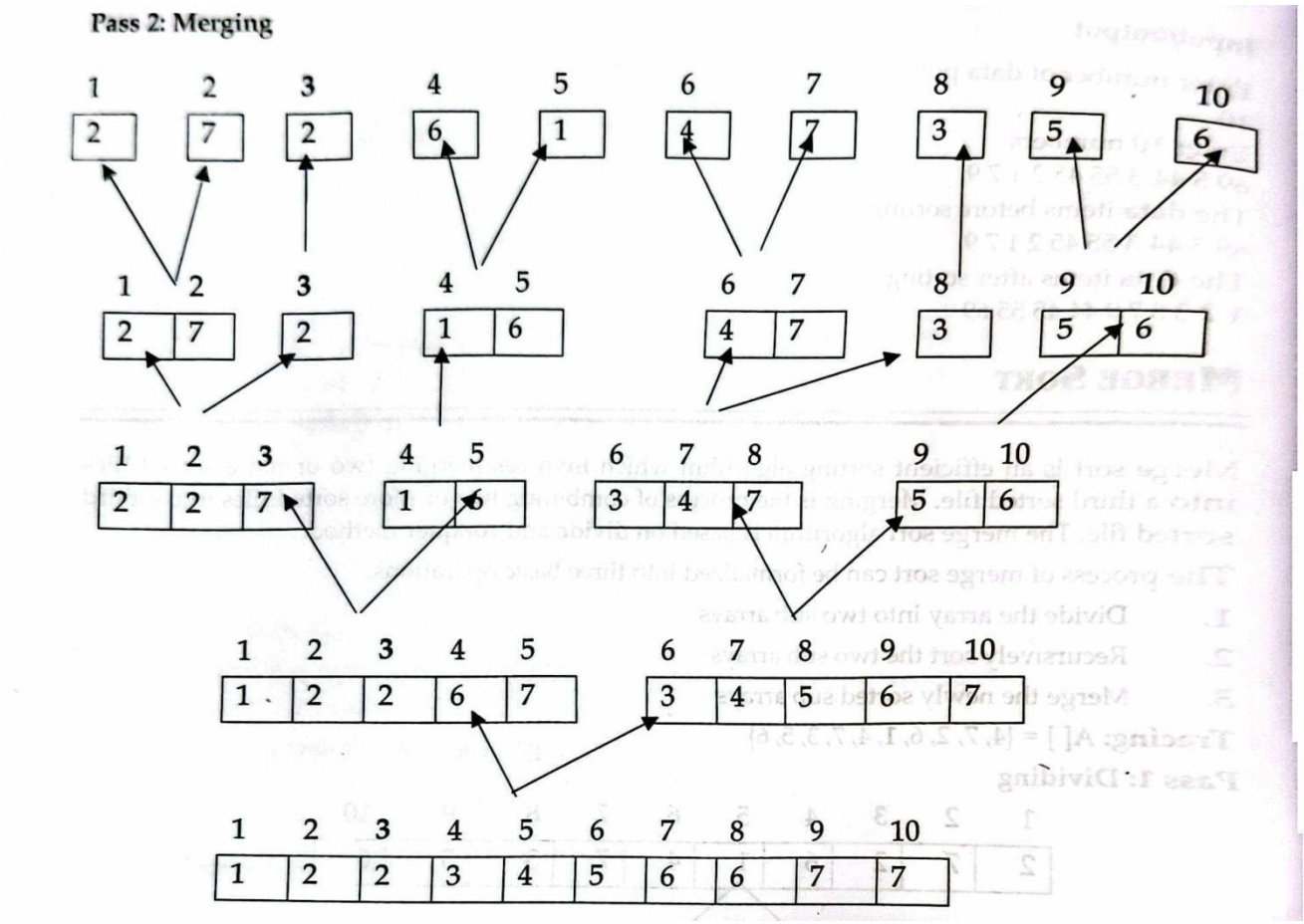
Merge sort is an efficient sorting algorithm which involves merging two or more sorted files into a third sorted file. Merging is the process of combining two or more sorted files into a third sorted file. The merge sort algorithm is based on divide and conquers method.

The process of merge sort can be formalized into three basic operations.

- Divide the array into two sub arrays
- Recursively sort the two sub arrays
- Merge the newly sorted sub arrays

Tracing: A[ ] = (2, 7, 2, 6, 1, 4, 7,3,5,6)





Quick Sort

Quick sort developed by C.A.R Hoare is an unstable sorting. In practice this is the fastest sorting method. It possesses very good average case complexity among all the sorting algorithms. This algorithm is based on the divide and conquer paradigm. The main idea behind this sorting is partitioning of the elements.

Steps for Quick Sort:

Divide: partition the array into two nonempty sub arrays.

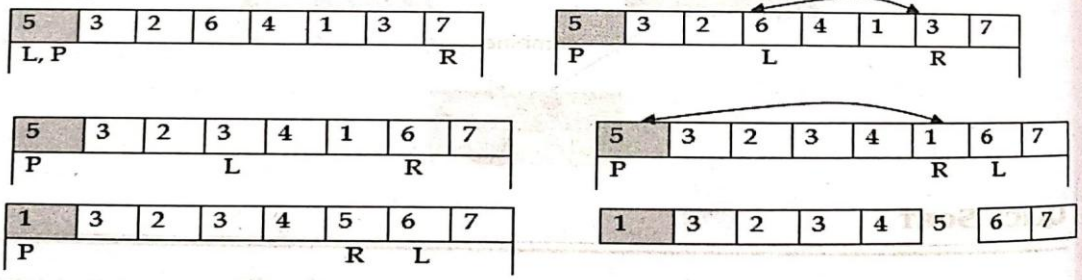
Conquer: two sub arrays are sorted recursively.

Combine: two sub arrays are already sorted in place so no need to combine.

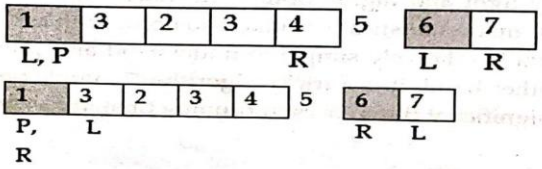
Tracing: Sort the following data items by using Quick sort

A[ ]={5, 3, 2, 6, 4, 1, 3, 7}

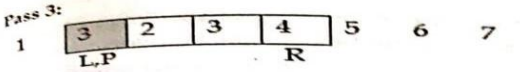
Pass 1:

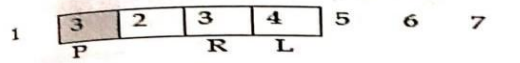


Pass 2:

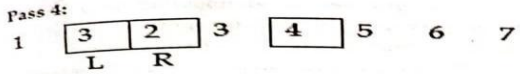


Pass 3:

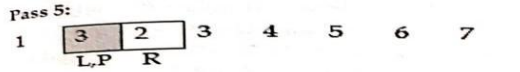


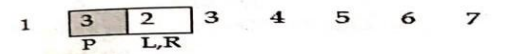


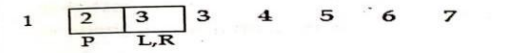
Pass 4:

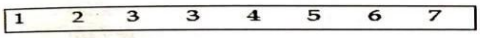


Pass 5:









Algorithm

1. Start
2. Choose a pivot
3. Set a left pointer and right pointer
4. Compare the left pointer element (lelement) with the pivot and the right pointer element (relement) with the pivot.
5. Check if lelement<pivot and relement>pivot:
   - If yes, increment the left pointer and decrement the right pointer
   - If not, swap the lelement and relement
6. When left >= right, swap the pivot with either left or right pointer.
7. Repeat steps 1 - 5 on the left half and the right half of the list till the entire list is sorted.
8. Stop

Shell Sort

Shell sort is the generalization of insertion sort which overcomes the drawbacks of insertion sort by comparing elements separated by a gap of several positions. In general, Shell sort performs the following steps.

- Step 1: Arrange the elements in the tabular form and sort the columns by using insertion sort.
- Step 2: Repeat Step 1; each time with smaller number of longer columns in such way that at the end, there is only one column of data to be sorted.

Though it is not the fastest algorithm known, **Shell sort** is a sub quadratic algorithm whose code is only slightly longer than the insertion sort, making it the simplest of the faster algorithms. Shells idea was to avoid the large amount of data movement, first by comparing elements that were far apart and then by comparing elements that were less far apart, and so on, gradually shrinking toward the basic insertion sort.

The shell sort is a diminishing increment sort. This sort divides the original file into separate sub-files. These sub-files contain every k^{th} element of the original file. The value of k is called increment.

For example, if there are n elements to be sorted and value of k is five then,

Sub-file 1→a[0], a[5], a[10], a[15].....

Sub-file 2→a[1], a[6], a[11], a[16].....

Sub-file 3→a[2], a[7], a[12], a[17].....

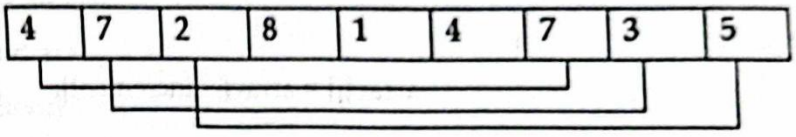
Sub-file 4→a[3], a[8], a[13], a[18].....

Sub-file 5→a[4], a[9], a[14], a[19].....

Tracing: A[ ] = {4, 7, 2, 8, 1, 4, 7, 3, 5}

Pass 1:

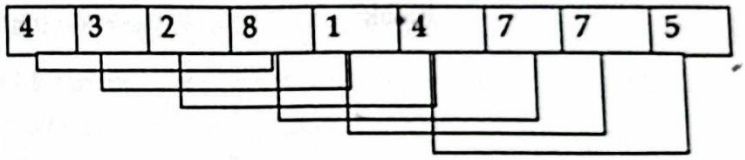
Increment=6(say):



4	3	2	8	1	4	7	7	5

Pass 2:

Increment=6/2=3:

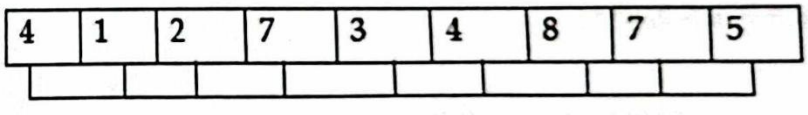


[redacted]

4	1	2	7	3	4	8	7	5

Pass 3:

Increment=3/2=1:



1	2	3	4	4	5	7	7	8