UNIT-5

SORTING & SEARCHING

1Q. Define Searching. Explain linear search. ---- 10 Marks

(Note: For Linear search & Binary Search if algorithm asked also you can write Program . so one is enough i.e., either algorithm or program)

Ans: . Searching: Searching is a process of finding an element in the given list. If the element is found, the searching is successful. Otherwise the searching is unsuccessful. Element which we search is known as "key element".

There are two types of searching techniques. They are:

1. Linear search
2. Binary search

Linear search: Linear search is also known as "sequential search process", where it compares Key elements in the list one by one. ***Time complexity of linear search is O(n).***

Example for Linear Search:

Consider Elements: 10 8 15 20 25
Key Element: 20

Step1: 10 8 15 20 25
key, flag =0

Step 2: 10 8 15 20 25
key, flag =0

Step 3: 10 8 15 20 25
key, flag =0

Step 4: 10 8 15 20 25
key, flag =1

Output: Element found at position 4

Program on Linear search:

```
import java.util.*;
class linear1
{
public static void main(String args[])
{
int[] a= { 10,8,15,20,25};
int n= a.length;

int key=20;
int flag =0,i;
for(i=0;i<n;i++)

{
   if(a[i]==key)
    {

       flag=1; break;
    }
}
 if(flag==1)
    System.out.println("Element Found at position"+(i+1));
 else
  System.out.println("Element Not Found");
}
}
```

Output: Element Found at position4

2Q. Define Searching. Explain binary search. ---- 10 Marks

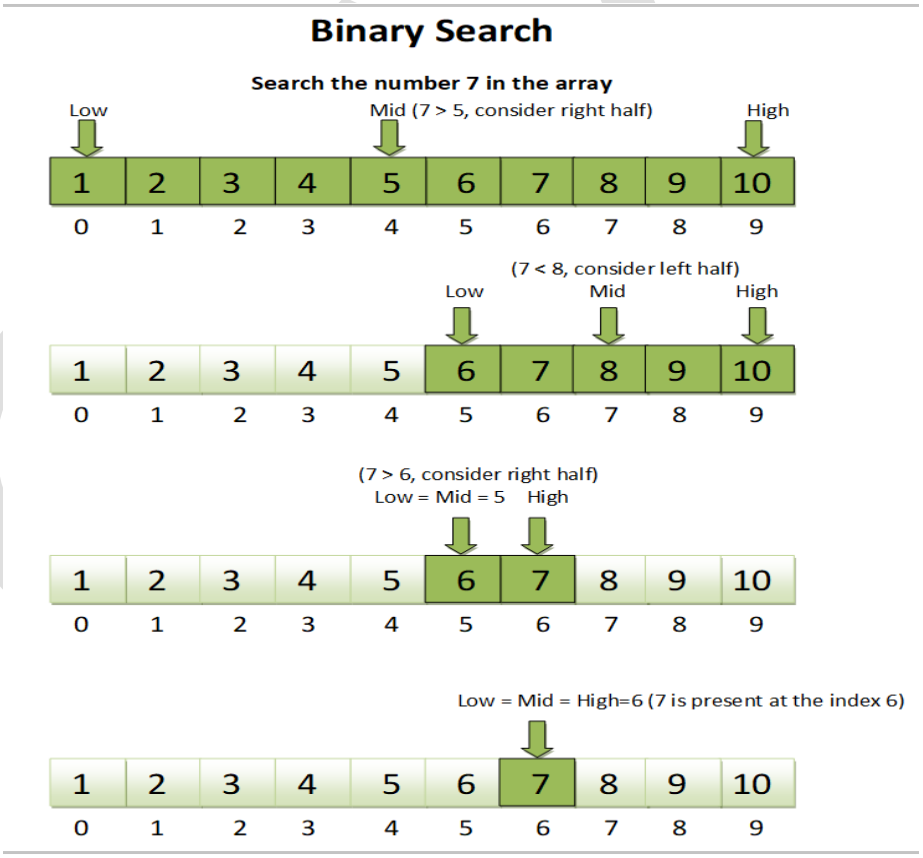
Ans: . Searching: Searching is a process of finding an element in the given list. If the element is found, the searching is successful. Otherwise the searching is unsuccessful. Element which we search is known as "key element".

There are two types of searching techniques. They are:

1. Linear search
2. Binary search

Binary Search: Binary search follows "Divide & conquer technique", where it compares key element with middle element in the list recursively until element is found. For binary search we give elements in "sorted order". ***Time complexity of binary search is O(logn).***

Example for Binary Search:



Output: Element found at position 7

Program on Binary Search:

```
class binary
{
public static void main(String args[])
{
int[] a= {1,2,3,4,5,6,7,8};
int n=a.length;
int key=7;
int left,right,mid,flag=0;
left=0; right=n-1; mid=(left+right)/2;
System.out.println(n+"   "+key);
  while(left<right)
  {
    if(key==a[mid])

   {
      flag=1;
      break;
     }
    else if(key<a[mid])
    {
      right= mid-1;
    }
   else if(key > a[mid])
    {
      left = mid+1;
    }
   mid=(left+right)/2;
  }
System.out.println(flag);
   if(flag==1)
     System.out.println("Element found");
   else
        System.out.println("Element not found");
}
}
```

Output: Element found

3Q. Define Mergesort. Explain with example.

A. Sorting: Sorting is the process of arranging elements either in ascending or descending order

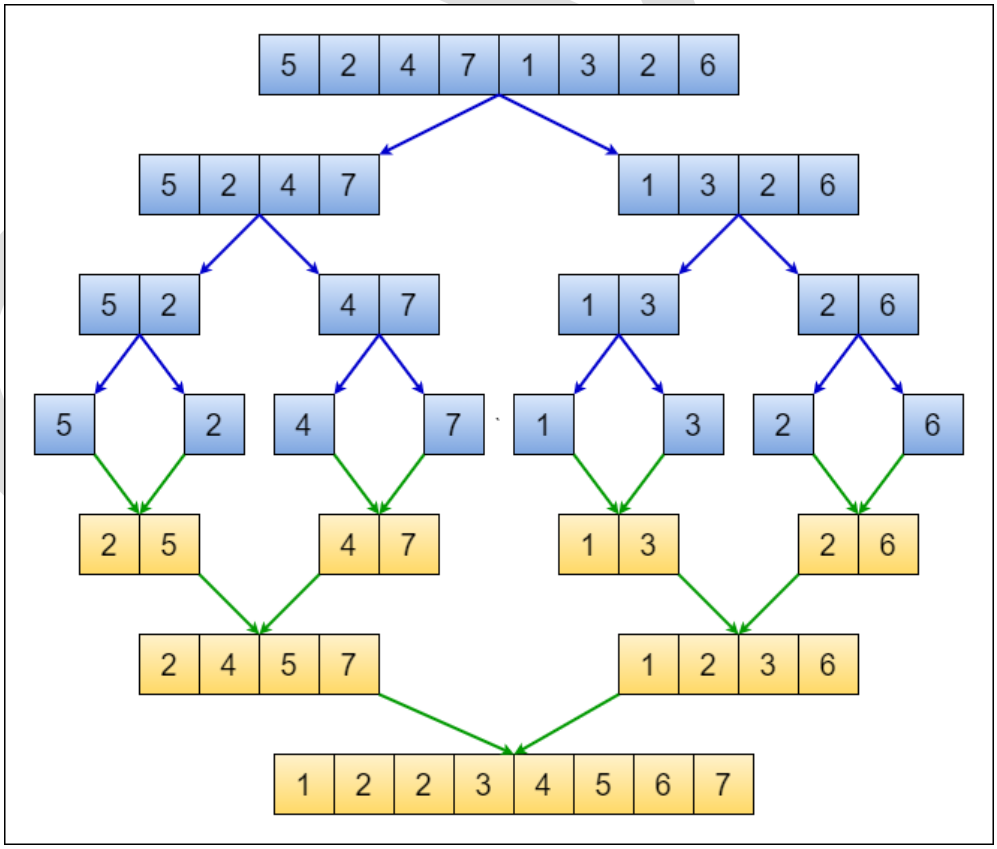
Different types of sorting techniques are:-

- Bubble sort
- Merge sort
- Quick sort
- Selection sort
- Heap sort

Merge sort: Merge Sort follows "Divide & Conquer Technique".

- Merge sort is used to combine two lists into a sorted list. Input lists must be in ascending order and with no duplicate values.
- ***Time complexity of merge sort is O(nlogn)***

Example of Merge Sort:



Program on Merge Sort:

```
class merge
{
   public static void main(String arg[])
       {
         int[] a= { 4,6,8,11};
        int[] b = { 1,3,5,7};
       int m,n,i,j,k;
        n = a.length; m=b.length;
       int[] c = new int[n+m];
       i=0;j=0;k=0;
     while(i<n&&j<m)
     {
       if(a[i]<b[j])
      {
         c[k]=a[i];
          k++;i++;
       }
       else
        {
          c[k]=b[j];
        k++;j++;
        }
      }
        while(i<n) // loop 9
        {
        c[k]=a[i];
        k++; i++;
       }
       while(j<m)  // loop 10
        {
          c[k]=b[j];
           k++; j++;
        }
     System.out.println("Sorted list using merge sort");
       for(k=0;k<n+m;k++)
        System.out.print("\t"+c[k]);
       }  }
```

Output: Sorted list using merge sort

1 3 4 5 6 7 8 11

Q. Explain Quick sort with example

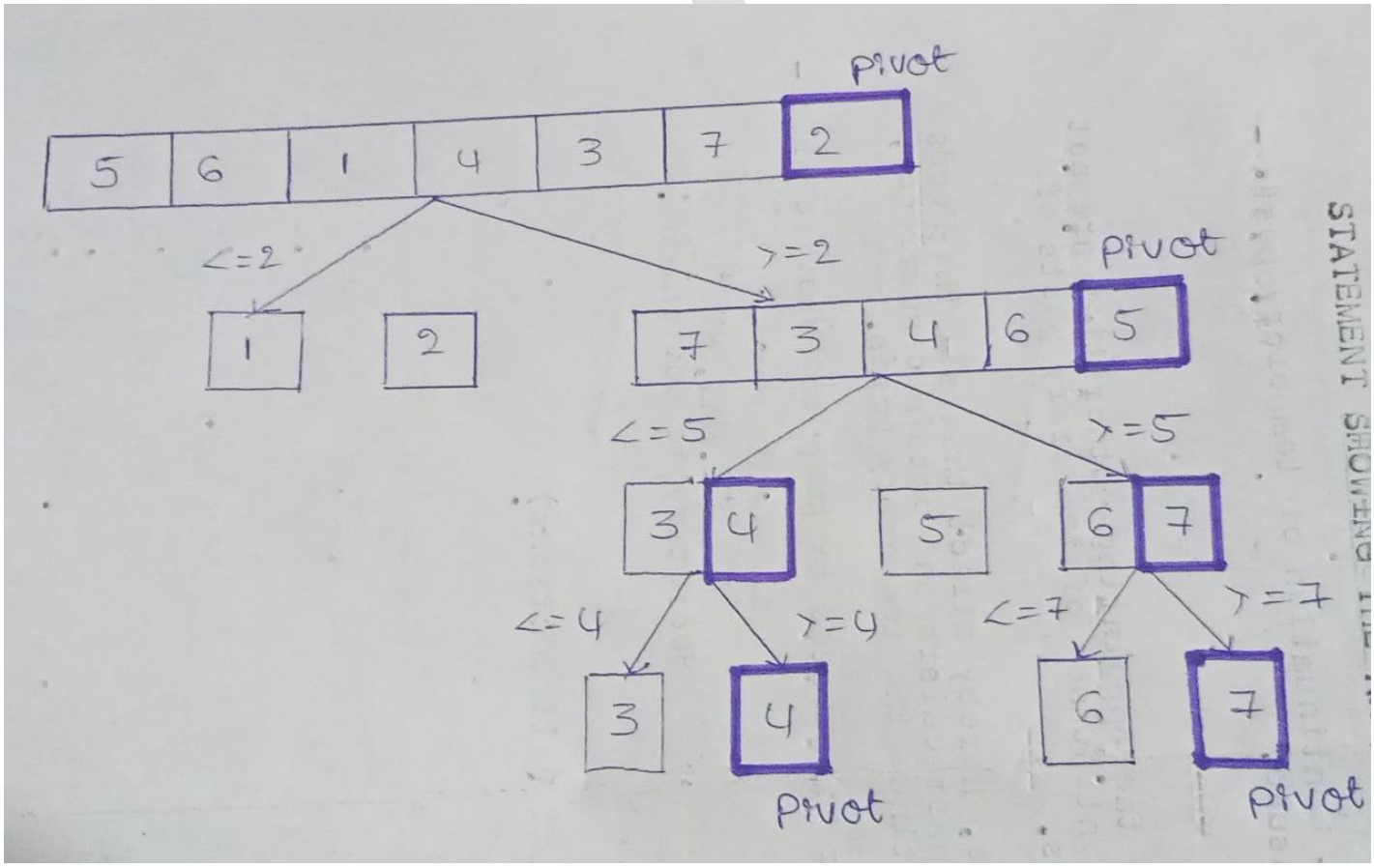
Ans: Quick sort algorithm follows divide and conquer technique.

- ✓ Quick sort also known as "partition exchange sort".
- ✓ ***Time Complexity of quick sort is O(nlogn).***

Algorithm:

1. In each pass (iteration) quicksort algorithm divides the list into 3 parts i.e.,
   1. Elements less than the pivot element
   2. Elements greater than pivot element
   3. Pivot element
2. Pivot element can be any element in the list (i.e., 1st element or last element or middle element)
3. Apply Quicksort algorithm recursively to get the final sorted list.

Example of Quick Sort: Consider elements 5,6,1,4,3,7,2



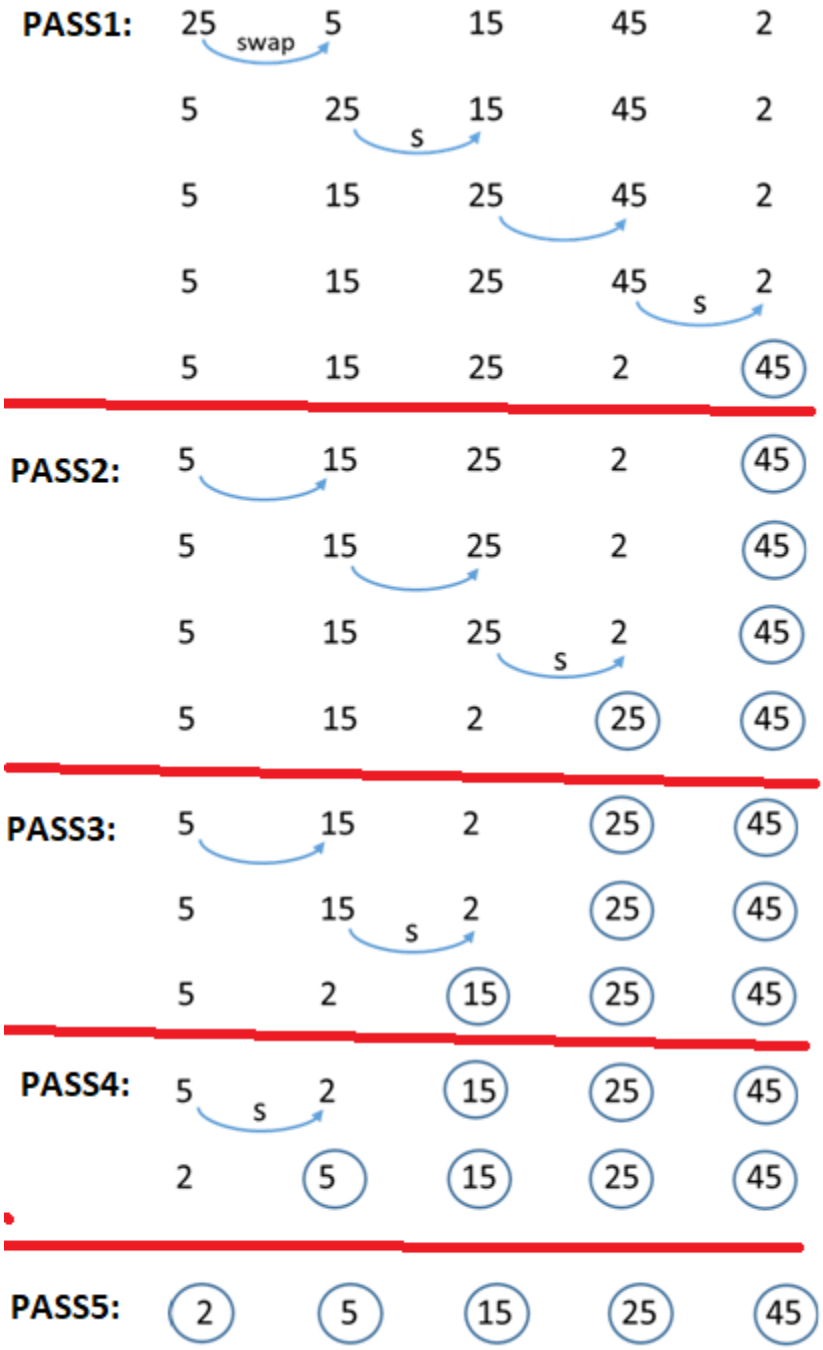
Q. Explain bubble sort with example.

Ans: Bubble Sort: Bubble sort is the simple sorting algorithm that uses "swapping techniques" to place elements in sorted order. ***The time complexity of bubble sort is $O(n^2)$.***

Algorithm:

1. In every pass it compares the elements and swaps them if they are in wrong order (unsorted order)
2. In pass 1 it compares 'n' elements
3. In pass 2 it compares n-1 elements
4. Similarly in pass n it compares only one elements

Example of Bubble Sort: Consider elements: 25, 5, 15, 45, 2



Q. Explain insertion sort with example

A. Insertion sort: Insertion sort places one value in the sorted position in every pass. In insertion sort given list is divided into sorted list and unsorted list. ***Time complexity of insertion sort is O(n^2).***

Algorithm:

Step1: Initially, we take only the first element then it can be considered as "sorted list"

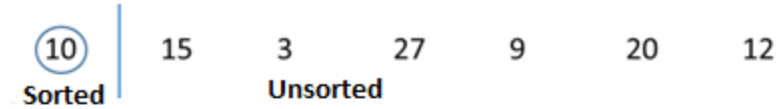
Step 2: In the next pass or iteration consider the second element and place it in the sorted position.

Step 3: Repeat step2, until all elements of list are completed.

Example: consider the elements 10, 15, 3, 27, 9, 20, 12

Original list: 10, 15, 3, 27, 9, 20, 12

Pass 1: Assume 10 in correct position

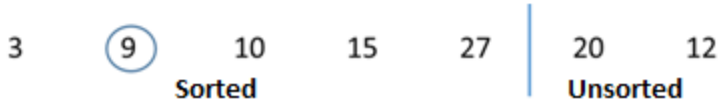


Pass 2: place 15 in sorted list

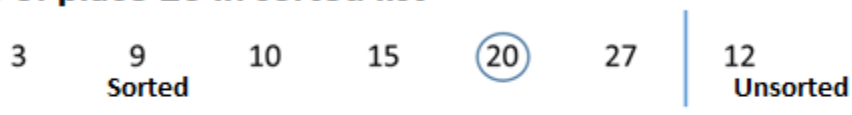
Pass 3: place 3 in sorted list

Pass 4: place 27 in sorted list

Pass 5: place 9 in sorted list



Pass 6: place 20 in sorted list



Pass 7: place 12 in sorted list

Q. Explain selection sort with example

A. Selection sort: Selection sort is used to arrange elements either in ascending or descending order. In selection sort given list is divided into sorted list and unsorted list. ***Time complexity of selection sort is O(n^2).***

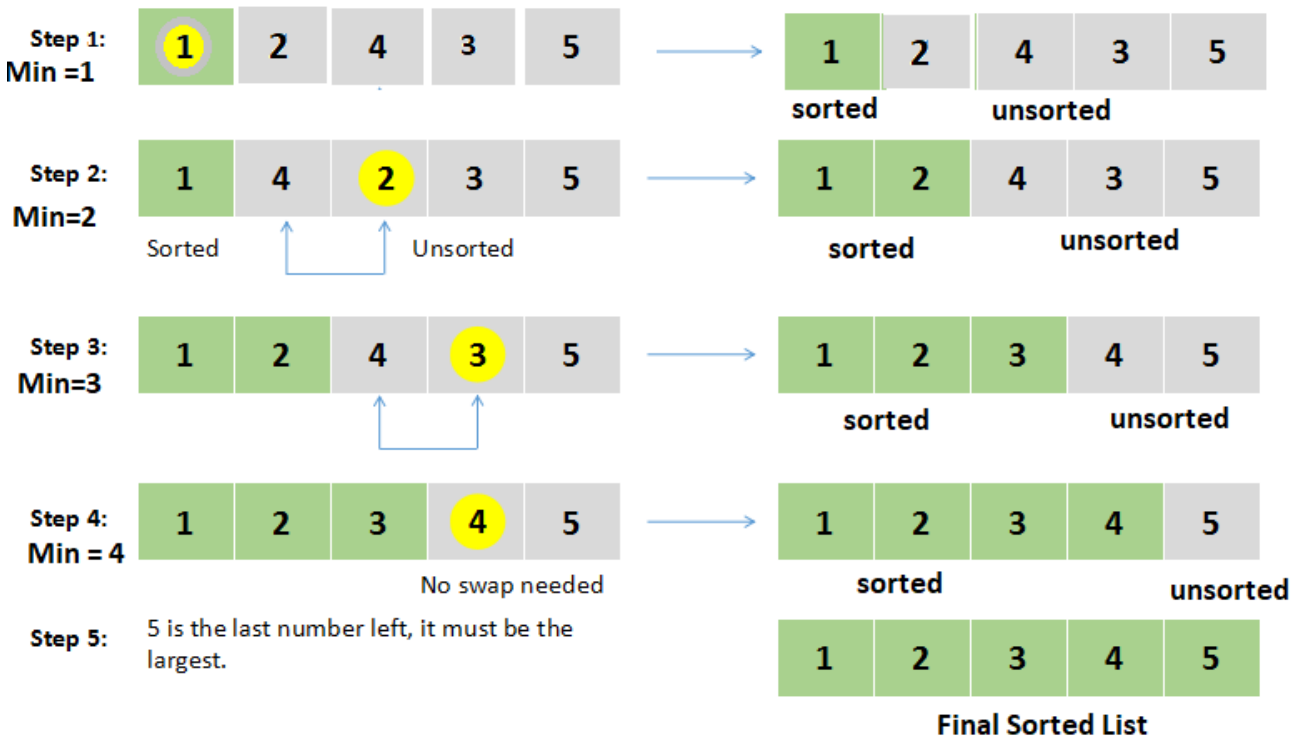
Algorithm:

Step1: In selection sort, in every pass (iteration) we select either largest or smallest element in the list and place it in correct position.

Step2: Find the smallest element in the list and place it in first position in the sorted list.

Step3: Repeat step2, until all elements of list are completed. In the each step sorted list is increase by 'one' and unsorted list decrease by 'one'.

Example: consider the elements : 1 2 4 3 5



Q. Explain Heapsort with example.

Ans. Heapsort:- Heapsort algorithm is developed using heaptree . Heap sort is a sorting technique which uses a max. heap or min. heap tree to arrange the elements in sorted order.

- ✓ For ascending order we use min. heap tree
- ✓ For descending order we use max. heap tree.
- ✓ **Time Complexity of heap sort is O(nlogn)**

Min. Heaptree: In a heap tree if the values of the parent node is less than its child nodes, it is known as min. heaptree.

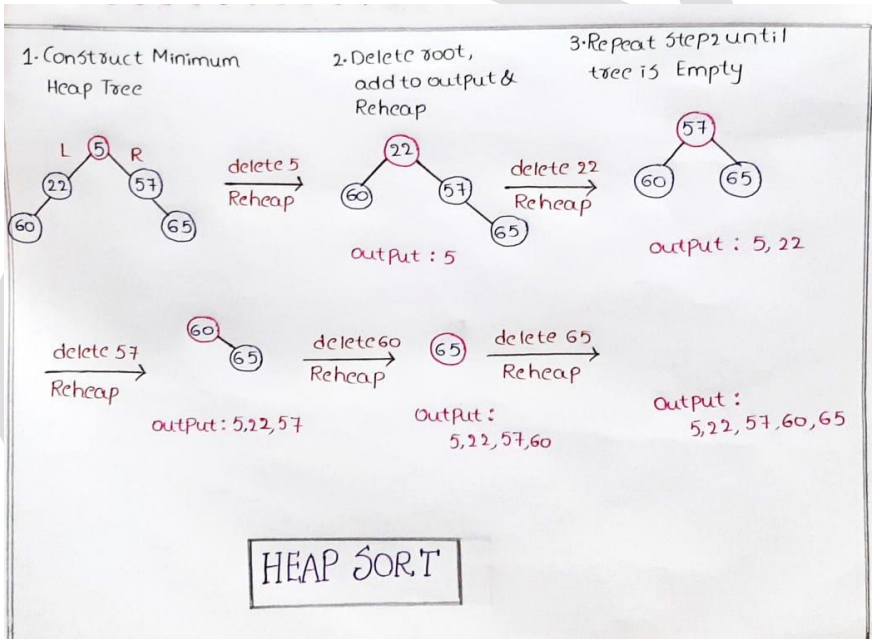
Algorithm:

Step 1: Construct min.heap tree for given elements.

Step 2: Delete root node, add to output , and reheap the tree

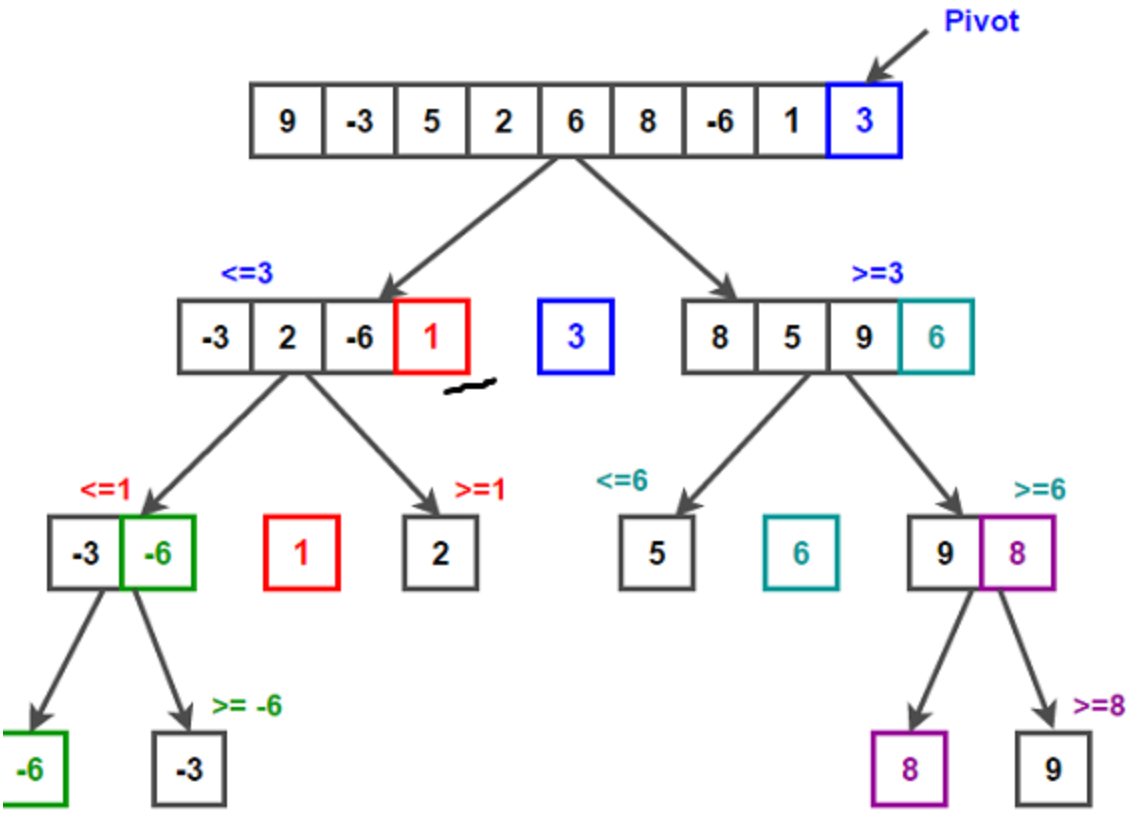
Step 3: Repeat Step2 until tree is empty.

Example: Consider elements 22, 57,5,60,65



---------- XXXXXXXX ---------- for exam upto here only

Quick sort Example 2:



Quick Sort Program:

```
public class quicksort
{
        public static void main(String args[ ])
        {
                int a[ ]={2,3,4,1,5,6,9,7,8};
             int n=a.length;
                qsort(a,0,n-1);
                System.out.println("Elements after Quick sort:");
                for(int i=0;i<n;i++)
                        System.out.print(" "+a[i]);
                }
public static void qsort(int a[ ], int low, int r)
{
        int l=low;
        int h = r;
        if(l>=r)
            return;
        int m=a[(l+h)/2];
        while(l<h)
       {
      while(l<h && a[l]<m)
         l++;
        while(l<h && a[h]>m)
         h--;
        if(l<h)
        {
           int t = a[l];
           a[l]=a[h];
           a[h]=t;
        }
        if(h>l)
        {
         int t = h;
           h=l;
            l=t;
        }
                qsort(a,low,l);
                qsort(a,l+1,r);
        }
   }}
```

Output:

```
 Elements after Quick sort:
 1 2 3 4 5 6 7 8 9
```

Bubble sort Program:

```
import java.io.*;
class bubble
{
        public static void main(String arg[])
        {
          int[] a= { 4,1,5,2,3};
         int n=a.length;
        for(int i=0;i<n-1;i++)
        {
          for(int j=0;j<(n-1)-i;j++)
          {
            int temp;
            if(a[j]>a[j+1])
            {
              temp=a[j];
              a[j]=a[j+1];
               a[j+1]=temp;
             }
            }
         }

     System.out.println("Sorted list using Bubble sort");
       for(int k=0;k<n;k++)
        System.out.print("\t"+a[k]);

        }
}
```

Selection Sort:

```
class selection
{
	public static void main(String arg[])
	{
		int a[]={4,3,1,5,2};
		int n,i,j,temp;
		n=a.length;
			for(i=0;i<n-1;i++)
			for(j=i+1;j<n;j++)
			if(a[i]>a[j])
			{
				temp=a[i];
				a[i]=a[j];
				a[j]=temp;
			}
			System.out.println("After sorting");
			for(i=0;i<n;i++)
			System.out.print("  "+a[i]);


	}
}
```

Output:

```
Final Selection Sort
  1  2  3  4  5
```

Insertion Sort:

```
class insertion
{
   public static void main(String arg[])
  {
    int[] a={4,2,3,5,1};
      int n,i,j,x;
      n=a.length;
     for (j = 1; j < n; j++)
     {
       int key = a[j];
        i = j-1;
        while ( (i > -1) && ( a [i] > key ) )
         {
           a [i+1] = a [i];
             i--;
         }
         a[i+1] = key;
        }
  System.out.println("final list in insertion sort");
    for(i=0;i<5;i++)
      System.out.print(a[i]+"   ");
 }
}
```

Output:

final list in insertion sort
1 2 3 4 5