UNIT-V

SORTINGS

Sortings is the process, in which the elements are arranged in ascending or descending order.

Selection sort :-In selection sort algorithm, set lower index value to min variable. then find the minimum value in the array. Then swap the minimum value with the value of minimum index. At next iteration leave the value of minimum index position and sort the remaining value by following same steps.

ex: Consider the following array as an example.

I pass

a[0]	30		4
a[1]	40	swap(4,30)	40
a[2]	4		30
a[3]	5		5
a[4]	10		10

II pass

a[0]	4		4
a[1]	40		5
a[2]	30	swap(40,5)	30
a[3]	5		40
a[4]	10		10

III pass

a[0]	4		4
a[1]	5		5
a[2]	30		10
a[3]	40	swap(30,10)	40
a[4]	10		30

IV pass

a[0]	4		4
a[1]	5		5
a[2]	10		10
a[3]	40	swap(40,30)	30
a[4]	30		40

Working of the selection sort

Say we have an array unsorted A [0],A[1],A[2].....A[n-1] and A[n] as input. Then the following steps are followed by selection sort algorithm to sort the values of an array.

Step 1 - Set variable MIN to location 0.

Step 2 - Search the minimum element in the array.

Step 3 - Swap with value at location MIN with minimum value in the array.

Step 4 - Increment MIN variable to point to next element.

Step 5 - Repeat the process until the array is sorted.

```
import java.io.*;
class sorting
{
int a[] = new int[10];
int n=0,i,j,t,min;
public void read()throws IOException
{
DataInputStream inn=new DataInputStream(System.in);
System.out.println("\n enter how many numbers do you want to insert");
n=Integer.parseInt(inn.readLine());
for(i=0;i<n;i++)
a[i]=Integer.parseInt(inn.readLine());
}
public void sort()
{
 for(i=0;i<n-1;i++)
 {
 min=i;
  for(j=i+1;j<n;j++)
    {
     if(a[j]<a[min])
      {

       t=a[j];
       a[j]=a[min];
       a[min]=t;
       }
     }
    }
 }
 public void display()
       {
            System.out.println ("after sorting ");
             for(i=0;i<n;i++)
             System.out.println (a[i]);
           }
 }
 class ssort
     {
           public static void main(String arg[])throws IOException
           {
              sorting obj=new sorting();
              obj.read();
              obj.sort();
              obj.display();
             }
           }
```

output:-

```
D:\dsprog>javac ssort.java
D:\dsprog>java ssort
 enter how many numbers do you want to insert
5
54
38
95
46
85
After sorting
38
46
54
85
95
```

Insertion sort

The insertion sort works as follows

1.The first element of the array is assumed to sorted.

2. The second element is compared with the first element and it is inserted before or after the first element according to the order.Thus the first two elements are sorted.

3. The third element is compared with firsr two elements and it is inserted either before first and second element or between first and second elements or after first and second elements thus the first three elements are sorted.

4. Simililarly the 'n' elements will be sorted in ' n-1' passes.

Table :

ex: Consider the following array as an example.

I pass

a[0]	50		40
a[1]	40	swap(50,40)	50
a[2]	30		30
a[3]	20		20
a[4]	10		10

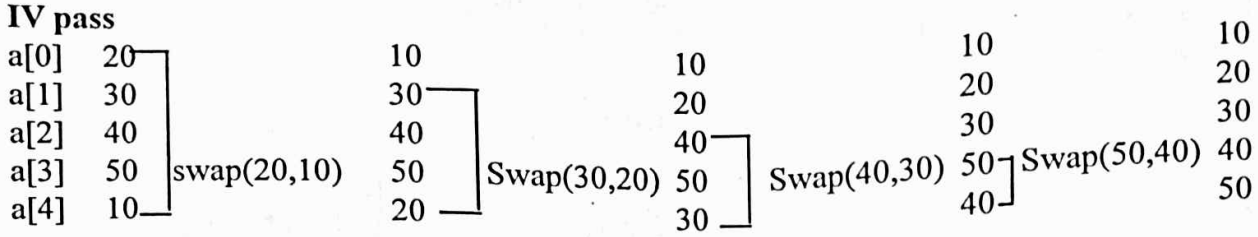
II pass

a[0]	40		30		30
a[1]	50		50		40
a[2]	30	swap(40,30)	40	Swap(50,40)	50
a[3]	20		20		20
a[4]	10		10		10

III pass

a[0]	30		20		20	20
a[1]	40		40		30	30
a[2]	50		50	Swap(40,30)	50	Swap(50,40) 40
a[3]	20	swap(20,30)	30		40	50
a[4]	10		10		10	10

IV pass




```
import java.io.*;
class Sorting
{
int a[] = new int[10];
int n,i,j;
public void read( )throws IOException
{
  DataInputStream inn=new DataInputStream(System.in);
  System.out.println("\n Enter how many numbers do you want to insert");
  n=Integer.parseInt(inn.readLine());
  for(i=0;i<n;i++)
  a[i]=Integer.parseInt(inn.readLine());
}
public void sort()
{
            for(i=1;i<n;i++)
            {
            int small=a[i];
                j=i-1;
            while((j>-1)&&(a[j]>small))
                    {
                    a[j+1]=a[j];
                    j--;
                    }
            a[j+1]=small;

            }
}
public void display()
      {
      System.out.println ("after sorting ");
      for(int i=0;i<n;i++)
      System.out.println(a[i]);
      }
}
class Isort
{
public static void main (String arg[])throws IOException
{
Sorting obj=new Sorting();
```

```
        obj.read();
        obj.sort();
        obj.display();
}
}
```

Output:-

```
D:\dsprog>javac Isort.java
D:\dsprog>java  Isort
 Enter how many numbers do you want to insert
5
21
36
35
24
65
After sorting
21
24
35
36
65
```

Bubble Sort:- Bubble sorting is an algorithm in which we are comparing first two values and put the larger one at higher index. Then we take next two values compare these values and place larger value at higher index. This process do iteratively untill the largest value is reached at last index. Then start again from zero index up to n-1 index. The algorithm follows the same steps iteratively untill elements are sorted. Say we have an array unstored A[0],A[1], A[2]----------A[n-1] and A[n] as input. Then the following steps are followed by bubble sort algorithm to sort the values of an array.

1. Compare A[0] and A[1].
2. If A[0]>A[1] then Swap A[0] and A[1].
3. Take next A[1] and A[2],Compare these values.
4. If A[1]>A[2] then Swap A[1] and A[2]

.....

.....

at last compare A[n-1] and A[n]. If A[n-1]>A[n] then swap A[n-1] and A[n]. As we see the highest value is reached at nth position. At next iteration leave nth value. Then apply the same steps repeatedly on A[0],A[1],A[2]--------A[n-1] elements untill the values of array is sorted.

Let the array be A [5] = { 55,45,35,25,15 }

I Pass

A[0]	55 ⌐ ex	45	45	45	45
A[1]	45 ⌐	55 ⌐ ex	35	35	35
A[2]	35	35	55 ⌐ ex	25	25
A[3]	25	25	25 ⌐	55 ⌐ ex	15
A[4]	15	15	15	15	(55)

II Pass

A[0]	45 ⌐ ex	35	35	35
A[1]	35 ⌐	45 ⌐ ex	25	25
A[2]	25	25	45 ⌐ ex	15
A[3]	15	15	15 ⌐	(45)
A[4]	55	55	55	(55)

III Pass

A[0]	35 ⌐ ex	25	25
A[1]	25 ⌐	35 ⌐ ex	15
A[2]	15	15	(35)
A[3]	45	45	(45)
A[4]	55	55	(55)

VI Pass

A[0]	25 ⌐ ex	(15)
A[1]	15 ⌐	(25)
A[2]	35	(35)
A[3]	45	(45)
A[4]	55	(55)

```
import java.io.*;
class sorting
{
        int a[]=new int[10];
        int n,i,j,t;
        void  read()throws IOException
         {
        DataInputStream inn=new DataInputStream(System.in);
          System.out.println("enter how many numbers do you want to sort");
          n=Integer.parseInt(inn.readLine());
          System.out.println("enter"+n+" numbers");
          for(i=0;i<n;i++)
              a[i]=Integer.parseInt(inn.readLine());
          }
```

```
        void sort()
        {
          for(i=0;i<n;i++)
                {
                        for(j=0;j<n-1;j++)
                        {
                        if(a[j]>a[j+1])
                                {
                                t=a[j];
                                a[j]=a[j+1];
                                a[j+1]=t;
                                }
                        }
                }
        }
        void display()
          {
           System.out.println("Data   after sorting is");
            for(i=0;i<n;i++)
            System.out.println(a[i]);
          }
}
   class bsort
          {
            public static void main (String arg[])throws IOException
            {
                sorting obj=new sorting();
                obj.read();
                obj.sort();
                obj.display();
              }
            }
```

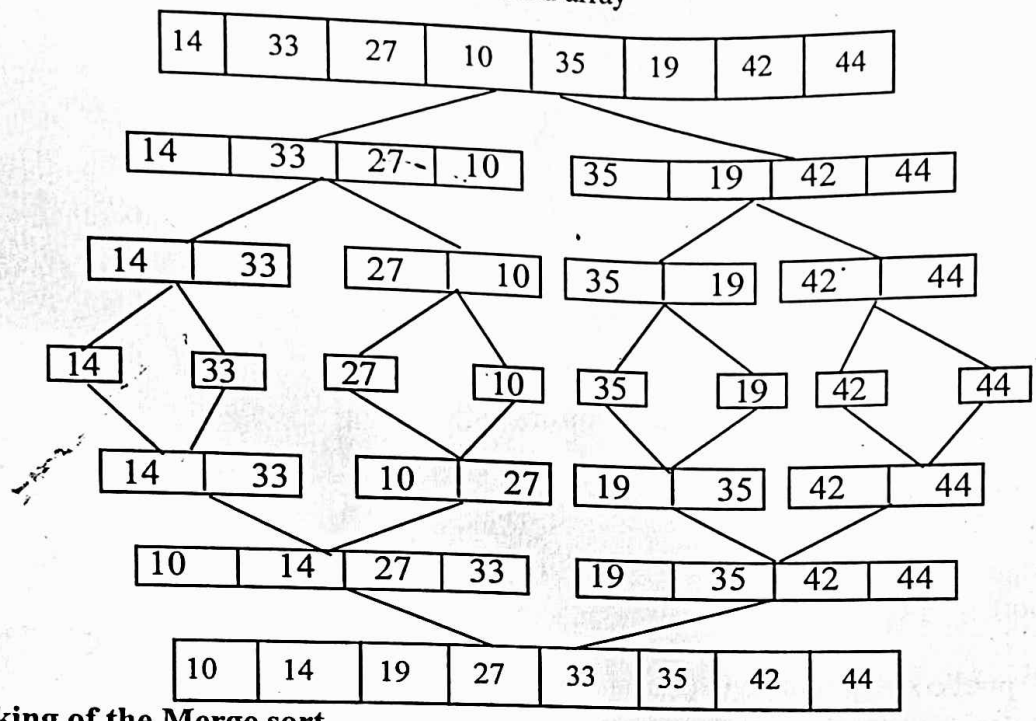
output:-

```
D:\>javac bsort.java
D:\>java bsort
Enter how many numbers do you want to sort
4
enter4 numbers
95
68
51
94
Data after sorting is
51
68
94
95
```

Mergesort:

Merge Sort follows the rule of **Divide and Conquer.** In Merge Sort the unsorted list is divided into 'N' sub-lists, Each having one element, because a list consisting of one element is always sorted. Then, it repeatedly merges these sub-lists, finally only one sorted list is produced.

Example for Merge Sort:- Consider an unsorted array



Working of the Merge sort..

To understand Merge Sort, We take an unsorted array as the following.

14	33	27	10	35	19	42	44

We know that Merge Sort first divide the whole array iteratively into equal halves.An array of 8 items is divided into two arrays of size 4.

14	33	27	10

35	19	42	44

This does not change the sequence of appearance of items in the original. Now we divide these two arrays into halves.

14	33

27	10

35	19

42	44

We further divide these arrays and we achieve single value which can no more be divided.

14	33	27	10	35	19	42	44

Now, we combine them in exactly the same manner as they were broken down.

We first compare the element for each list and then combine them into another list in a sorted manner. We see that 14 and 33 are in sorted positions. We compare 27 and 10 and in the target list of 2 values we put 10 first, followed by 27. We change the order of 19 and 35 whereas 42 and 44 are placed sequentially.

14	33	10	27	19	35	42	44

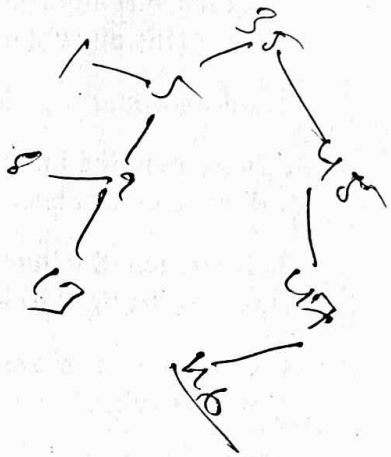
In the next iteration of the combining phase, we compare lists of two data values, and merge them into a list of four data values placing all in a sorted order.

10	14	27	33	19	35	42	44

After the final merging, the list should look like this

10	14	19	27	33	35	42	44

```
import java.io.*;
public class MergeSort
{
  public static void sort(int[] a, int low, int high)
  {
    int N = high - low;
    if (N <= 1)
       return;
    int mid = low + N/2;
    sort(a, low, mid);
    sort(a, mid, high);
    int[] temp = new int[N];
    int i = low, j = mid;
    for (int k = 0; k < N; k++)
    {
      if (i == mid)
        temp[k] = a[j++];
      else if (j == high)
        temp[k] = a[i++];
      else if (a[j]<a[i])
        temp[k] = a[j++];
      else
        temp[k] = a[i++];
    }
    for (int k = 0; k < N; k++)
      a[low + k] = temp[k];
  }
  public static void main(String[] args) throws IOException
{
DataInputStream inn=new DataInputStream(System.in);
    int n, i;
    System.out.println("Enter n Value");
    n = Integer.parseInt(inn.readLine());
    int arr[] = new int[ n ];
     System.out.println("Enter the  elements");
    for (i = 0; i < n; i++)
      arr[i] =Integer.parseInt(inn.readLine());
```



```
        sort(arr, 0, n);
        System.out.println("\nElements after sorting ");
      for (i = 0; i < n; i++)
        System.out.print(arr[i]+" ");
      System.out.println();
   }
}
```

output:-

```
D:\>javac MergeSort.java
D:\> java MergeSort

Enter number of integer elements
5

Enter 5 integer elements
67
32
45
1
4

Elements after sorting
1 4 32 45 67
```

Quick sort

In quick sort algorithm pick an element from array of elements. This element is called the pivot. This algorithem follows the bellow steps.

1.Take the starting element index into 'i' and ending element index into 'j'.

2.Increment the index 'i' value until greatest element is found, i.c compare pivot with array values from left to right.

3.Decrement the index 'j' value until smallest element is found i.e compare pivot with array values from right to left.

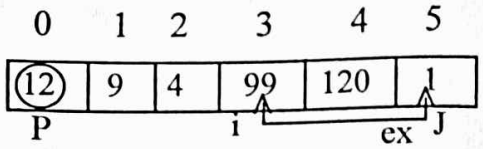
4. Compare the index of i ,j .If (i < j)is true swap a[i] with a[j].Repet the steps 2 & 3 until (i < j) is false.

5.If (i < j) is false then swap a[j] with p and alsopivot element is divide the array into sub arrays.

6. Again repeat the steps 1,2,3,4,5 until all the elements are sorted.

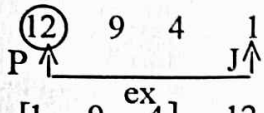
Working of quick sort algorithm :

Input: 12 9 4 99 120 1



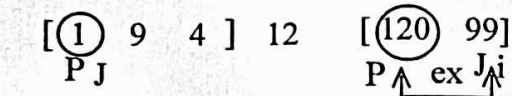
(3<5)
(i < J) T swap (a[i] ,a[j])

12 9 4 1 120 99



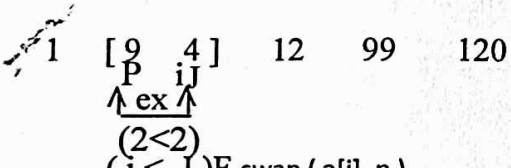
(4<3)
(i < J) F swap (P, a[j])

[1 9 4] 12 [120 99]



(1<0) (5<5)
(i < J) F swap (a[j] ,p) (i < J) F swap (a[j] ,p)

1 [9 4] 12 [99] 120



(i < J)F swap (a[j] ,p)

1 [4] 9 12 99 120

1 4 9 12 99 120

The sorted list is:-1 4 9 12 99 120

```
import java .io. *;
class Quicksort1
{
public static void main ( String arg[ ] )throws IOException
{
DataInputStream inn=new DataInputStream(System.in);
int  i;
System.out.println("enter an array size ");
int n=Integer.parseInt(inn.readLine());
int array [ ]  =  new int  [ n ];
System.out.println("Enter array elements ");
 for (i=0; i<n;++i)
  array [i ] =Integer.parseInt(inn.readLine());
System.out.println("values before the sort");
for ( i =  0;  i <n;  i++)
System.out.print( array [ i ]+"     ");
```

```
System.out.println( );
quick_srt( array , 0,  n-1);
System.out.print ("values after the sort:\n");
for (i   =  0;   i  <n;   i++)
System.out.print(array [ i ]+"    ");
System.out.println ( );
}
public static void quick_srt ( int array [ ], int low, int   n )
{
int lo  =  low;
int  hi   =n;
if   (lo == n)
{
return;
}
int mid  =  array [(lo+hi)/2];
while( lo < hi )
{
   while((lo < hi)&&(array[lo]<mid) )
   {
    lo++;
   }
   while((lo < hi)&&(array[hi]>mid) )
   {
     hi--;
   }
 if ( lo  <  hi )
  {
   int T  =  array [ lo ];
int T  = hi;
hi = lo;
lo = T;
}
quick_srt( array ,  low,  lo );
quick_srt(array,(lo==low)? ( lo+1):lo,  n) ;
}
}
```

OUTPUT:-

```
D:\dsprog>javac Quicksort1.java

D:\dsprog>java  Quicksort1
enter an array size     :

5
enter5 array elements   :
35
```

87
32
94
24
values before the sort :

35 87 32 94 24

values after the sort:

24 32 35 87 94

HEAP SORT :-The following are the steps to perform heap sort.

1.Build the heap tree based on given array elements.

2.Start delete min operations , storing each deleted element at the end of the heap array.

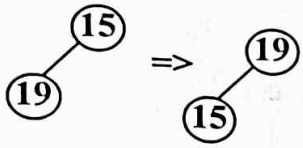
The minimum element in the root level then it is called minheap similarly the maximum element in the root and the smaller elements are childrens then it is called max heap

Ex; 15,19,10,7,17,6

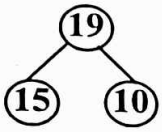
1.Building the heap tree

Step 1:

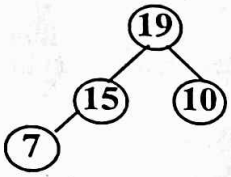
Step 2:



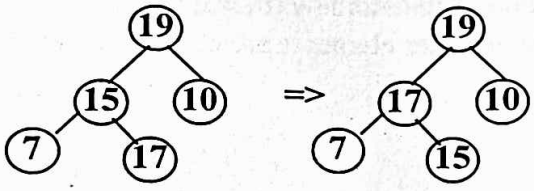
Step 3:



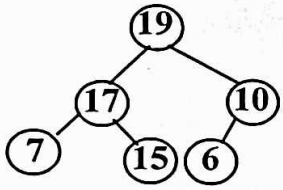
Step 4:



Step5:



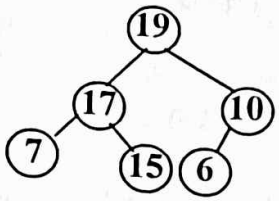
Step 6



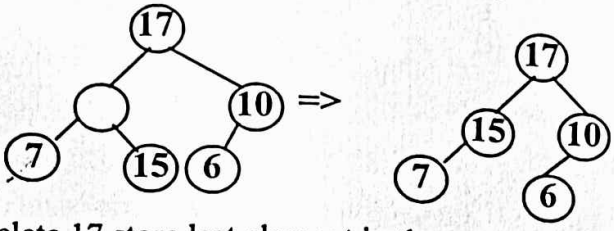
2.Deletion of a heap sort;

Sorting based on delete max operation

19	17	10	7	15	6

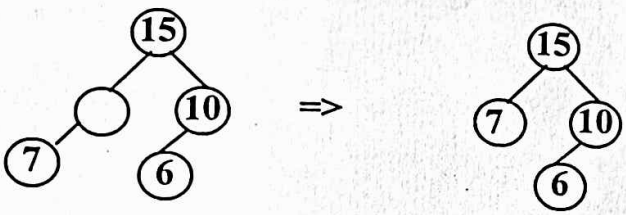


Step 1: Delete 19 store last element in the array



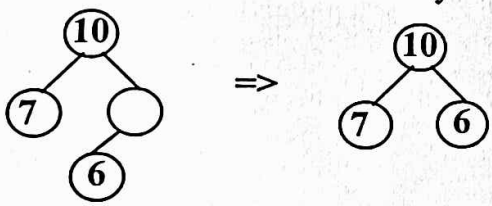
17	15	10	7	6	19

Step 2: Delete 17 store last element in the array



15	7	10	6	17	19

Step 3: Delete 15 store last element in the array

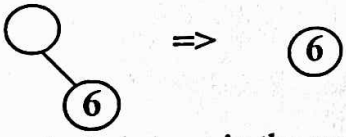


10	7	6	15	17	19

Step 4: Delete 10store last element in the array

7	6	10	15	17	19

Step 5: Delete 7 store last element in the array



6	7	10	15	17	19

Step 6: Delete 6 store last element in the array

The sorted array is

6	7	10	15	17	19

programe for heapsort:-

```
public class HeapSort
{
  private static int N;
  public static void sort(int arr[])
  {
    heapify(arr);
    for (int i = N; i > 0; i--)
    {
      swap(arr,0, i);
      N = N-1;
      maxheap(arr, 0);
    }
  }
  public static void heapify(int arr[])
  {
    N = arr.length-1;
    for (int i = N/2; i >= 0; i--)
      maxheap(arr, i);
  }
  public static void maxheap(int arr[], int i)
  {
    int left = 2*i ;
    int right = 2*i + 1;
    int max = i;
    if (left <= N && arr[left] > arr[i])
      max = left;
    if (right <= N && arr[right] > arr[max])
      max = right;
    if (max != i)
    {
      swap(arr, i, max);
      maxheap(arr, max);
    }
  }
  public static void swap(int arr[], int i, int j)
  {
    int tmp = arr[i];
    arr[i] = arr[j];
    arr[j] = tmp;
  }
  public static void main(String[] args) throws IOException
  {
    DataInputStream inn=new  DataInputStream(System.in);
    System.out.println("Heap Sort Test\n");
    int n, i;
```

```
        System.out.println("Enter n value");
        n = Integer.parseInt(inn.readLine());
        int arr[] = new int[ n ];
        System.out.println("\nEnter elements");
        for (i = 0; i < n; i++)
           arr[i] = Integer.parseInt(inn.readLine());
        sort(arr);

        System.out.println("\nElements after sorting ");
        for (i = 0; i < n; i++)
           System.out.print(arr[i]+" ");
        System.out.println();
    }
}
```

output:--

```
D:\>javac HeapSort.java
D:\>java  HeapSort
Enter number of integer elements
6
Enter 6 integer elements
78
64
25
91
64
35
Elements after sorting
25 35 64 64 78 91
```

SEARCHINGS:-

Searchings refers to search a particular element in the list. In this operation we approach mainly two types of methods like linear search and binary search.

Linear search

In this we search the given element is there (or) not in sequential process like beginning to ending. If element is found then display "Element Found" otherwise display "Element Not Found" the following algorithm is used to given the element is there (or) not

ALGORITHM:-

Linear search(ele)

Step 1:- START

Step 2:-set I=0and flag=0

Step 3:- Repeat the steps 4&5

Step 4:- If(A[I]=ELE) THEN

flag=1

Step 5:-I=I+1

Step 6:-if (flag=1)then

Write ("Element is FOUND")

Else

Write("Element is NOT FOUND")

Step 7:-STOP

```
import java .io.*;
class Isearch
{
public static void main(String arg[])throws IOException
{
int []a=new int[10];
int i, n, key, c=0;
DataInputStream  inn=new DataInputStream(System.in);
System.out.println ("Enter the size");
n=Integer.parseInt(inn.readLine());
System.out.println("Enter elements");
for(i=0;i<n;i++)
a[i]=Integer.parseInt(inn.readLine());
System.out.println("Enter elements for search");
key=Integer.parseInt(inn.readLine());
for(i=0;i<n;i++)
{
if(key==a[i])
{
c++;
break;
}
```

```
}
if(c==1)
System.out.println("Element found");
else
System.out.println("Element not  found");
}
}
```

Output:-

```
D:\dsprog>javac Isearch.java
D:\dsprog>java  Isearch
Enter the size
5
Enter elements
53
24
95
76
31
Enter elements for search
95
Element found
```

Binary Search

In binary search first find out middle of the element based on the low and high values.In this low value is a beginning array position and high value is ending elememt position and next you check the condition search element is middle element or not, if it is true then display "Element is Found " otherwise you can check another condition,if search element is less than middle element then our searching process with lower half otherwise serching process with upper half.This process will continue until searching element is found or not .The following algoritham is used to search the given element is there or not.

Algorithm:-

```
Binary_Search( ele )
Step1:-START
Step2:-Set L=0,h=max-1,flag=0;
Step3:-Repeat steps 4 until condition is false
Step:4:while(i < = h)
        m=(i+h)/2
        if( ele = a[m] ) then flag = 1
        if (ele < a[m] ) then h=m-1
        if( ele > a[m] ) then L=m+1
Step5:-if(flag=1) then
        write("Element Found");
```

else
Write("Element not Found ");
Step6:- STOP

```
import java.io.*;
class bsearch
{
public static void main(String arg[])throws IOException
{
int []a=new int[10];
int i,n,key,c=0,low=0,high,mid;
DataInputStream inn=new DataInputStream(System.in);
System.out.println("Enter the size");
n=Integer.parseInt(inn.readLine());
high=n-1;
System.out.println("Enter the Elements");
for(i=0;i<n;i++)
a[i]=Integer.parseInt(inn.readLine());
System.out.println("Enter element for search");
key=Integer.parseInt(inn.readLine());
while(low<=high)
{
mid=(low+high)/2;
if(a[mid]>key)
high=mid-1;
if(a[mid]<key)
low=mid+1;
if(a[mid]==key)
{
c++;
break;
}
}
if(c==1)
System.out.println("Element found");
else
System.out.println("Element not found");
}
}
```