I. Binary search

```
public class Binarysearch
{
/* assuming sorted array the search starts in the middle, the search item is continuously compared to the
middle value; if not found the next middle value is compared*/

public void display()
{
int [] list1={78,6,0,1,2,3,33,100,22}; //Won't work as array is un- sorted. Use Sequential search here or sort
first.
int item=0;
      int bottom=0;                  // lower bound of subarray
      int top = list1.length- 1;        // upper bound of subarray
      int middle;                    // middle of subarray
      boolean found = false;           // to stop if item found
      int location = -1;              // index of item in array
                                     //also not found

      while (bottom <= top && !found)
      {

        middle = (bottom + top)/2;
        if (list1[middle] == item)    // success
        {
          found = true;
          location = middle;
        }
        else if (list1[middle] < item) // not in bottom half
          bottom = middle + 1;
        else                          // item cannot be in top half
          top = middle - 1;

          System.out.println("Middle " +middle);
          System.out.println("Bottom " +bottom);
          System.out.println("Top " +top);

      }
      System.out.print(location);
    }
public static void main(String [] args)
{
  Binarysearch b= new Binarysearch();
  b.display();
}
```

II. Selection sort

Process:

A) **Find the largest value and swap with the last element on the list.**
B) **Search all elements to find the largest and swap with the second last element**

Example 1

To begin sorting the data
7 1 9 3 5 4
we find the largest element, 9, and swap it with 4, the element at the ***right end of the list***. The result, after the first pass of a selection sort is
7 1 4 3 5 9 '----/
On the second pass, all the items except the last are examined to see which of these is the largest and this item is then placed at the right end of this sublist. This pattern continues on subsequent passes; on each one, the largest value among the unsorted items is placed at the top of the sublist.

Example 2

Using the set of data shown in Example 1, the output shows the results of successive passes of selection sort.

```
After Pass
1       7  1  4  3  5  9
2       5  1  4  3  7  9
3       3  1  4  5  7  9
4       3  1  4  5  7  9
5       1  3  4  5  7  9
```

Only five passes are required to sort six items. Once all but one of the items are placed in their correct positions, the remaining item must also be in its correct position.

The method selectSort uses a selection sort to arrange an array of **double** values in **ascending** order.

```
double [] list={1.7,23.0,2.3,78.90,7.0,5.67,1.0,8.0};
    {
      for (int top = list.length - 1 ; top > 0 ; top--)
      {

        int largeLoc = 0;                  //location of largest value
        for (int i = 1 ; i <= top ; i++)
          if (list [i] > list [largeLoc])  // compare 2 items
            largeLoc = i;                  // if current item is larger then swap
        double temp = list [top];
        list [top] = list [largeLoc];
        list [largeLoc] = temp;
   }
}
```