**1. Trace a walkthrough (i.e., show the individual steps) of selection sort with each of these lists:**

**a. 4 7 11 4 9 5 11 7 3 5**The selection sort method will go through the list and find the smallest element and swap with the first element. It will then update the first element so that previously sorted elements will not be included. So, this process will look like the following:

4 7 11 4 9 5 11 7 3 5

3 7 11 4 9 5 11 7 4 5

3 4 11 7 9 5 11 7 4 5

3 4 4 7 9 5 11 7 11 5

3 4 4 5 9 7 11 7 11 5

3 4 4 5 5 7 11 7 11 9

3 4 4 5 5 7 11 7 11 9 (7 already in correct spot)

3 4 4 5 5 7 7 11 11 9

3 4 4 5 5 7 7 9 11 11  
**b. –7 6 8 7 5 9 0 11 10 5 8**-7 6 8 7 5 9 0 11 10 5 8

-7 0 8 7 5 9 6 11 10 5 8

-7 0 5 7 8 9 6 11 10 5 8

-7 0 5 5 8 9 6 11 10 7 8

-7 0 5 5 6 9 8 11 10 7 8

-7 0 5 5 6 7 8 11 10 9 8

-7 0 5 5 6 7 8 11 10 9 8 (8 already in correct spot)

-7 0 5 5 6 7 8 8 10 9 11

-7 0 5 5 6 7 8 8 9 10 11

**2. Trace a walkthrough of merge sort with each of these lists:**

**a. 5 11 7 3 5 4 7 11 4 9**

5 11 7 3 5 4 7 11 4 9

5 11 7 3 5 4 7 11 4 9

5 11 7 3 5 4 7 11 4 9 5 11 7 3 5 4 7 11 4 9 5 11 3 5 7 4 7 4 9 11

3 5 5 7 11 4 4 7 9 11

3 4 4 5 5 7 7 9 11 11

**b. 9 0 11 10 5 8 –7 6 8 7 5**  9 0 11 10 5 8 -7 6 8 7 5

9 0 11 10 5 8 -7 6 8 7 5

9 0 11 10 5 8 -7 6 8 7 5 9 0 11 10 5 8 -7 6 8 7 5 0 9 5 10 11 -7 6 8 5 7 8

0 5 9 10 11 -7 5 6 7 8 8

-7 0 5 5 6 7 8 8 9 10 11

**3. Your task is to remove all duplicates from an array. For example, if the array has the values**

**4 7 11 4 9 5 11 7 3 5**

**then the array should be changed to**

**4 7 11 9 5 3**

**Write an algorithm (and, optionally, a program) to carry out this task.  
If your algorithm makes more than one pass through the original array to perform this task, can you re-write the algorithm so that it only makes one pass?**

public class HW6RemoveDuplicates {

public static void main(String [] args){

//declare an int array with the original values

int[] numArray = {4, 7, 11, 4, 9, 5, 11, 7, 3, 5};

//declare an int variable for the current size of the array

int arraySize = numArray.length;

//use nested for loops to go through the array and check if elements are the same

for(int i = 0; i < arraySize; i++){

for(int j = 0; j < arraySize; j++){

if(i != j){

//if two different elements are the same, replace the current element with the next and move all other elements up one using a for loop

if(numArray[i] == numArray[j]){

for(int k = j+1; k < arraySize; k++){

numArray[k-1] = numArray[k];

}

//decrease current size of array by 1

arraySize = arraySize - 1;

//decrease j by 1

j = j - 1;

}

}

}

}

//use a for loop to go through the size of the new array and print the unique elements

for(int i = 0; i < arraySize; i++){

System.out.print(numArray[i] + " ");

}

}

}In the original for loop, the program only goes through the array once. However, in the other for loops, the program goes through the array more than once when checking to see if elements are the same. One way we could simplify this program would be to convert the array to an ArrayList then use the remove method in a for loop to easily remove any duplicate elements.

**4. Write a recursive version of the linear search algorithm.** public static int RecursiveLinearSearch(int[] x, int start, int desired){

//return -1 if program goes through array and doesn't find desired value

if(start >= x.length){

return -1;

}

//if current index of array contains desired value, return index

else if(x[start] == desired){

return start;

}

//otherwise, call recursive method using same array, start+1, and same desired value

else{

return RecursiveLinearSearch(x, start+1, desired);

}

}//end RecursiveLinearSearch

}

**5. Describe how to modify the selection sort algorithm (shown below) to sort an array of integers in descending (instead of ascending) order.**

selectionSort(array):  
    for each index i from 0 to array.length-2:  
        min ← findIndexOfSmallest(array, i)  
        swap(i, min)  
    end for  
end selectionSort  
  
findIndexOfSmallest(array, first):  
    minIndex ← first  
    for each index j from first+1 to array.length-1:  
        if array[j] < array[minIndex]:  
            minIndex ← j  
        end if  
    end for  
    return minIndex  
end findIndexOfSmallest  
  
swap(array, i1, i2):  
    temp ← array[i1]  
    array[i1] ← array[i2]  
    array[i2] ← temp  
end swap

In order to sort an array of integers in descending order rather than ascending order, you would modify the algorithm to search for the largest value rather than the smallest value. You would then swap the value at the current start index with the maximum value that was found. This can be done by changing findIndexOfSmallest so that the first index is the max index and updating the max index each time a greater integer is found (rather than a smaller integer). As mentioned above, in selectionSort, you would then swap the value at i with the value at maxIndex. The starting index would then increase by 1, and you would repeat this process for the rest of the array until it is sorted from largest to smallest.

**6. Write the Java statements for the following algorithm:**

1. **Generate an array of 100 random integers.**

int[] numArray = new int[100];

for(int i = 0; i < 100; i++){

numArray[i] = (int) (Math.random() \* 1000) + 1;

}

This will create an array of 100 random integers from 1 to 1000.

1. **Using the Arrays.sort static method, sort the array.**Arrays.sort(numArray);

This statement will sort the array from the previous problem.

1. **Using the Arrays.binarySearch static method, search the array of a user-input integer.**

Scanner input = new Scanner(System.in);  
System.out.print("Enter an integer to search for: ");  
int keyValue = input.nextInt();  
int index = Arrays.binarySearch(numArray, keyValue);

1. **If the integer is in the array, display the index where it was found.**

if(index >= 0){  
 System.out.println("Value found at index " + index);  
}

If the binarySearch method returns a value greater than or equal to 0, this means the integer is in the array.

1. **Otherwise, display the index where is should be added.**

else{  
 System.out.println("Value should be added at position " + ((index \* -1) - 1));  
}

If the binarySearch method returns a value less than 0, the value is not in the array. In this case, binarySearch will return -k-1, where k is the position before which the element should be inserted. So, we should print the index \* -1 and subtract 1 to give the position where the value should be inserted.

**7. Modify the selection sort algorithm (shown below) to sort an array of objects that implement the Measurable interface (also shown below).**

**selectionSort(array):  
    for each index i from 0 to array.length-2:  
        min ← findIndexOfSmallest(array, i)  
        swap(i, min)  
    end for  
end selectionSort  
  
findIndexOfSmallest(array, first):  
    minIndex ← first  
    for each index j from first+1 to array.length-1:  
        if array[j] < array[minIndex]:  
            minIndex ← j  
        end if  
    end for  
    return minIndex  
end findIndexOfSmallest  
  
swap(array, i1, i2):  
    temp ← array[i1]  
    array[i1] ← array[i2]  
    array[i2] ← temp  
end swap**

**/\*\*  
   Describes any class whose objects can be measured.  
\*/  
public interface Measurable  
{  
   /\*\*  
      Computes the measure of the object.  
      @return the measure  
   \*/  
   double getMeasure();  
}**

selectionSort(Measurable array[]):  
    for each index i from 0 to array.length-2:  
        min ← findIndexOfSmallest(array, i)  
        swap(i, min)  
    end for  
end selectionSort  
  
findIndexOfSmallest(Measurable array[], int first):  
    minIndex ← first  
    for each index j from first+1 to array.length-1:  
        if array[j].getMeasure < array[minIndex].getMeasure:  
            minIndex ← j  
        end if  
    end for  
    return minIndex  
end findIndexOfSmallest  
  
swap(Measurable array[], int i1, int i2):  
    Measurable temp ← array[i1]  
    array[i1] ← array[i2]  
    array[i2] ← temp  
end swap

Basically, the algorithm should be changed so that each array in the methods contain Measurable objects. Additionally, in the findIndexOfSmallest method, when checking if the current value is less than the minimum, you should use the getMeasure method from Measurable to get the values at the given indices of the array. Additionally, the variable temp should be declared as Measurable in the swap method. Finally, be sure to implement Measurable at the beginning of the class declaration and implement the getMeasure method within the program.