L20

Tuesday, January 25, 2022 3:12 PM

Sorting and Searching Techniques

Sorting - Bubble, Insertion and Selection

Q1. Write a Java program to Sort elements using Bubble Sort

Sorting specifies the way to arrange data in a particular order either in ascending or descending.

Bubble sort is an internal sorting technique in which **adjacent elements** are compared and exchanged if necessary.

The working procedure for **bubble sort** is as follows:

1. Let us consider an array of **n** elements (i.e., **a[n]**) to be sorted.
2. Compare the first two elements in the array i.e., **a[0]** and **a[1]**, if **a[1]** is less than **a[0]** then **interchange** the two values.
3. Next compare **a[1]** and **a[2]**, if **a[2]** is less than **a[1]** then interchange the values.
4. Continue this process till the last two elements are **compared** and **interchanged**.
5. Repeat the above steps for **n - 1** passes.

Let us consider an example of array numbers "**50 20 40 10 80**", and sort the array from **lowest number** to **greatest number** using bubble sort.

In each step, elements written in **bold** are being **compared**. Number of elements in the array are **5**, so **4** passes will be required.
Pass - 1 :
(**50 20** 40 10 80) -> (20 50 40 10 80) // Compared the first two elements, and swaps since 50 > 20.(20 **50 40** 10 80) -> (20 40 50 10 80) // Swap since 50 > 40.(20 40 **50 10** 80) -> (20 40 10 50 80) // Swap since 50 > 10.(20 40 10 **50 80**) -> (20 40 10 50 80) // Since the elements are already in order (50 < 80), algorithm does not swap them.
Total number of elements in the given array are **5**, so in **Pass - 1** total numbers compared are **4**. After completion of **Pass - 1** the **largest element** is moved to the **last position** of the array.

Now, **Pass - 2** can compare the elements of the array from **first position** to **second last position**.
Pass - 2 :
(**20 40** 10 50 80) -> (20 40 10 50 80) // Since the elements are already in order (20 < 50), algorithm does not swap them.(20 **40 10** 50 80) -> (20 10 40 50 80) // Swap since 40 > 10.(20 10 **40 50**80) -> (20 10 40 50 80) // Since the elements are already in order (40 < 50), algorithm does not swap them.
In **Pass - 2** total numbers compared are **3**. After completion of **Pass - 2** the **second largest element** is moved to the **second last position** of the array.

Now, **Pass - 3** can compare the elements of the array from **first position** to **third last position**.
Pass - 3 :
(**20 10** 40 50 80) -> (10 20 40 50 80) // Swap since 20 > 10.(10 **20 40**50 80) -> (10 20 40 50 80) // Since these elements are already in order (20 < 40), algorithm does not swap them.
In **Pass - 3** total numbers compared are **2**. After completion of **Pass - 3** the **third largest element** is moved to the **third last position** of the array.

Now, **Pass - 4** can compare the **first** and **second** elements of the array.
Pass - 4 :
(**10 20**40 50 80) -> (10 20 40 50 80) // Since these elements are already in order (10 < 20), algorithm does not swap them.
In **Pass - 4** total numbers compared are **1**. After completion of **Pass - 4** all the elements of the array are **sorted**. So, the result is **10 20 40 50 80**.

Write code to **sort** the array elements by using **bubble sort** technique.

Write a class BubbleSorting with a method bubbleSort(int[] array). The method receives an array of int type.

For example, if the array of elements 11, 15, 12, 10 are passed as arguments to the bubbleSort(..) method, then the output should be:
10
11
12
15

Note: Make sure to use the **println()** method and not the **print()** method.

q11039/BubbleSorting.jav | q11039/BubbleSortingMa

```
1  package q11039;
2
3  public class BubbleSorting{
4      public static void bubbleSort(int[] array){
5
6          int arraySize = array.length;
7
8          int temp;
9
10         for(int i = 0; i<arraySize-1; i++){
11
12             for(int j = 0; j<arraySize-i-1; j++){
13
14                 if(array[j] > array[j+1]){
15                     temp = array[j];
16                     array[j] = array[j+1];
17                     array[j+1] = temp;
18                 }
19             }
20         }
21
22         for(int k = 0; k<arraySize; k++){
23
24             System.out.println(array[k]);
25         }
26     }
27 }
```

Q2. Program to Sort elements using Insertion Sort

Insertion sort is one that **sorts** a set of elements by inserting an element into the existing sorted elements.

The working procedure for **insertion sort** is as follows:
1. Let us consider an array of **n** elements (i.e., **a[n]**) to be sorted.
2. The **first element a[0]** in the array is itself trivially sorted.
3. The **second element a[1]** is compared with first element **a[0]** and it will be inserted either before or after first element, so that **first** and **second** elements are sorted.
4. The **third element a[2]** is compared with **a[0]** and **a[1]** and it will be inserted into its proper place by checking conditions, so that first three elements are sorted.
5. Repeat the same process for **n - 1** passes.

Let us consider an example of array numbers "**50 20 40 10 30**", and sort the array from **lowest number** to **greatest number** using insertion sort.

In each step, elements written in **color** is **compared** with elements written in **bold**. Number of elements in the array are **5**, so **4** passes will be required.
Pass - 1 :
(**50**20 40 10 30) -> (**20 50** 40 10 30) // The second element **a[1]** is compared with the first element **a[0]** and swaps since 50 > 20, so first 2 elements are sorted.
Now, **Pass - 2** can compare **a[2]** with **a[0]** and **a[1]**.
Pass - 2 :
(**20 50**40 10 30) -> (**20 40 50** 10 30) // Since 40 > 20 and 40 < 50, so 40 is inserted in between 20 and 50.
Now, **Pass - 3** can compare **a[3]** with **a[0]**, **a[1]** and **a[2]**.
Pass - 3 :
(**20 40 50**10 30) -> (**10 20 40 50** 30) // Since 10 < 20, so it is inserted before 20.
Now, **Pass - 4** can compare **a[4]** with **a[0]**, **a[1]**, **a[2]** and **a[3]**.
Pass - 4 :
(**10 20 40 50**30) -> (10 20 30 40 50) // Since 30 > 10, 30 > 20 but 30 < 40, so 30 is inserted in between 20 and 40 .
After completion of **Pass - 4** all the elements of the array are **sorted**. So, the result is **10 20 30 40 50**.

Write a class InsertionSorting with a **public** method insertionSort. The method receives one parameter array of int type. Write a code to sort the array elements using insertion sort technique.

For example:
Cmd Args : 10 23 15 8 5
5
8
10
15
23
Note: Make sure to use the **println()** method and not the **print()** method.

q11040/InsertionSorting.j q11040/InsertionSortingM

```
1  package q11040;
2
3  public class InsertionSorting{
4      public static void insertionSort(int[] array){
5
6          int sizeOfArray = array.length;
7
8          for(int  i = 1; i < sizeOfArray; i++){
9
10             int key = array[i];
11             int j = i-1;
12
13             while(j >= 0 && key < array[j]){
14
15                 array[j+1] = array[j];
16                 --j;
17             }
18
19             array[j+1] = key;
20         }
21
22         for(int k = 0; k<sizeOfArray; k++){
23
24             System.out.println(array[k]);
25         }
26     }
27 }
```

Q3. Program to Sort elements using Selection Sort - Largest element method

Selection sort process can be done in **two** ways, one is the largest element method and the other is smallest element method.

The working procedure for selection sort using the **largest element method** is as follows:
1. Let us consider an array of **n** elements (i.e., **a[n]**) to be sorted.
2. In the first step, the **largest element** in the list is searched. Once the largest element is found, it is exchanged with the element which is placed at the **last position**. This completes the first pass.
3. In the next step, it searches for the **second largest element** in the list and it is interchanged with the element placed at **second last position**. This is done in second pass.
4. This process is repeated for **n - 1** passes to sort all the elements.

Let us consider an example of array numbers "**80 10 50 20 40**", and sort the array from **lowest number** to **greatest number** using selection sort by the largest element.

Pass - 1 :
(80 10 50 20 40) -> (40 10 50 20 80) // First finds the largest element and it is exchanged with the last position element.
After completion of **Pass - 1**, the largest element is moved to the end of the array.

Now, **Pass - 2** can find the next largest element with out considering the last position element.
Pass - 2 :
(40 10 50 20 80) -> (40 10 20 50 80) // Largest in **40 10 50 20** is **50** and it is replaced with next last position of the array.
After completion of **Pass - 2** the **second largest element** is moved to the **second last position** of the array.

Now, **Pass - 3** can find the next largest element with out considering the **last two position elements** because they are already sorted.
Pass - 3 :
(40 10 20 50 80) -> (20 10 40 50 80) // Largest in **40 10 20** is **40** and it is replaced with next last position of the array.
After completion of **Pass - 3** the **third largest element** is moved to the **third last position** of the array.

Now, **Pass - 4** can find the next largest element with out considering the **last three position elements** because they are already sorted.

Pass - 4 :
(20 10 40 50 80) -> (10 20 40 50 80) // Largest in **20 10** is **20** and it is replaced with next last position of the array.
After completion of **Pass - 4** all the elements of the array are **sorted**. So, the result is **10 20 40 50 80**.

Write a class SelectionSortingLargestElement with a **public** method selectionSortLargestEle. The method receives one parameter array of type int. Write a code to sort the array elements by using selection sort - largest element method.

For example:
Cmd Args : 63 83 33 53
33
53
63
83
Note: Make sure to use the **println()** method and not the **print()** method.

q11041/SelectionSortingL | q11041/SelectionSortingM

```
1  package q11041;
2
3  public class SelectionSortingLargestElement{
4      public static void selectionSortLargestEle(int[] array){
5
6          int sauravHathi = array.length;
7
8          for(int i = 0; i < sauravHathi-1; i++){
9
10             int lowest = i;
11
12             for(int j = i+1; j<sauravHathi; j++){
13
14                 if(array[j] < array[lowest]){
15                     lowest = j;
16                 }
17             }
18
19             int temp = array[i];
20             array[i] = array[lowest];
21             array[lowest] = temp;
22         }
23
24         for(int k = 0; k<sauravHathi; k++){
25
26             System.out.println(array[k]);
27         }
28     }
29 }
```

Q4. Program to Sort elements using Selection Sort - Smallest element method

The working procedure for **selection sort smallest element method** is as follows:
1. Let us consider an array of **n** elements (i.e., **a[n]**) to be sorted.
2. In the first step, the **smallest element** in the list is searched. Once the smallest element is found, it is exchanged with the element which is placed at the **first position**. This completes the first pass.
3. In the next step, it searches for the **second smallest element** in the list and it is interchanged with the element placed at **second position**. This is done in second pass.
4. This process is repeated for **n - 1** passes to sort all the elements.

Let us consider an example of array numbers "**80 10 50 20 40**", and sort the array from **lowest number** to **greatest number** using selection sort smallest element method.

Pass - 1 :
(80 10 50 20 40) -> (10 80 50 20 40) // First finds the smallest element and it is exchanged with the first position element.
After completion of **Pass - 1**, the smallest element is moved to the starting position of the array.

Now, **Pass - 2** can find the next smallest element with out considering the first position element.
Pass - 2 :
(10 80 50 20 40) -> (10 20 50 80 40) // Smallest in **80 50 20 40** is **20** and it is replaced with next first position of the array.
After completion of **Pass - 2** the **second smallest element** is moved to the **second position** of the array.

Now, **Pass - 3** can find the next smallest element with out considering the **first two position elements** because they are already sorted.
Pass - 3 :
(10 20 50 80 40) -> (10 20 40 80 50) // Smallest in **50 80 40** is **40** and it is replaced with next position of the array.
After completion of **Pass - 3** the **third smallest element** is moved to the **third position** of the array.

Now, **Pass - 4** can find the next smallest element with out considering the **first three position elements** because they are already sorted.
Pass - 4 :
(10 20 40 80 50) -> (10 20 40 50 80) // Smallest in **80 50** is **50** and it is replaced with next position of the array.
After completion of **Pass - 4** all the elements of the array are **sorted**. So, the result is **10 20 40 50 80**.

Write a class SelectionSortingSmallestElement with a **public** method selectionSortSmallestEle. The method receives one parameter array of type int. Write code to **sort** the array elements by using **selection sort - smallest element** method.

For example:
Cmd Args : 35 25 45 65
25
35
45
65
Note: Make sure to use the **println()** method and not the **print()** method.

q11042/SelectionSortingS | q11042/SelectionSortingM

```java
package q11042;

public class SelectionSortingSmallestElement{
    public static void selectionSortSmallestEle(int[] array){

        int smallest;
        int subscribe = array.length;

        for(int i = 0; i < subscribe-1; i++){

            smallest = i;

            for(int j = i+1; j < subscribe; j++){

                if(array[j] < array[smallest]){

                    smallest = j;
                }
            }

            int temp = array[smallest];
            array[smallest] = array[i];
            array[i] = temp;
        }

        for(int k = 0; k < subscribe; k++){

            System.out.println(array[k]);
        }
    }
}
```

Q5. Write a Java program to Sort elements using Merge Sort

Write code to **sort** the array elements by using **merge sort** technique.

Write a class MyMergeSort with **main** method.

q11043/MyMergeSort.jav

```java
package q11043;
import java.util.Scanner;
public class MyMergeSort {
    private int[] array;
    private int[] tempMergArr;
    private int length;
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.print("Enter no of elements in the array: ");
        int n = s.nextInt();
        int[] inputArr = new int[n];
        System.out.print("Enter elements in the array seperated by space: ");
        for(int i = 0; i < n; i++) {
            inputArr[i] = s.nextInt();
        }
        MyMergeSort mms = new MyMergeSort();
        mms.sort(inputArr);
        for(int i:inputArr){
            System.out.print(i);
            System.out.print(" ");
        }
    }
    public void sort(int inputArr[]) {
        this.array = inputArr;
        this.length = inputArr.length;
        this.tempMergArr = new int[length];
        doMergeSort(0, length - 1);
    }

    private void doMergeSort(int lowerIndex, int higherIndex) {

        if(lowerIndex < higherIndex){

            int middle = lowerIndex + (higherIndex - lowerIndex) / 2;
            doMergeSort(lowerIndex, middle);
            doMergeSort(middle+1, higherIndex);
            mergeParts(lowerIndex, middle, higherIndex);
        }


    }

    private void mergeParts(int lowerIndex, int middle, int higherIndex) {

        for(int i = lowerIndex; i <= higherIndex; i++){

            tempMergArr[i] = array[i];
        }
        int i = lowerIndex;
        int j = middle + 1;
        int k = lowerIndex;

        while(i <= middle && j <= higherIndex){

```

```
        if (tempMergArr[i] <= tempMergArr[j]){

            array[k] = tempMergArr[i];
            i++;
        } else {

            array[k] = tempMergArr[j];
            j++;
        }
        k++;
    }

    while(i <= middle){

        array[k] = tempMergArr[i];
        k++;
        i++;
    }
```

Searching - Linear, Binary

Q1. Program to Search an element using Linear Search

Searching specifies the way to **search** an element from the list of elements.

Linear search (or) Sequential search is to scan each entry in the list in a **sequential** manner until the desired element is found. i.e., it means to find a particular **key element** in a list of elements in a sequential manner.

Linear search is a searching technique in which it **sequentially** checks each element of the list for the **target value** until a match is found (or) until all the elements have been searched.

The working procedure for **linear search** is as follows:

1. Let us consider an array of **n** elements and a **key element** which is going to be search in the list of elements.
2. Compare the **key element** with the first element **a[0]**, if it is **matched** then stop the process and print the **index** of the key element where it is found, otherwise **repeat** the same process with **a[1]**.
3. Compare the **key element** with the second element **a[1]**, if it is **matched** then stop the process and print the **index** of the key element where it is found, otherwise **repeat** the same process with **a[2]**.
4. Continue this process until a match is found (or) until all the elements have been searched.

Let us consider an example of array numbers "**50 20 40 10 80**", and the key element is to find is 10.

Search - 1 :
Compare **10** with value of a[0] i.e., **50**, both are not equal so repeat the same process with **a[1]**.
Search - 2 :
Compare **10** with value of a[1] i.e., **20**, both are not equal so repeat the same process with **a[2]**.
Search - 3 :
Compare **10** with value of a[2] i.e., **40**, both are not equal so repeat the same process with **a[3]**.
Search - 4 :
Compare **10** with value of a[3] i.e., **10**, both are equal so stop the process and print **index** value where it found, i.e., position 3.

Write a class LinearSearch with a **public** method linearSearch that takes two parameters an array of type int[] and a key of type int. Write code to **search** key element within the array elements by using **linear search** technique.

Examples for your understanding:
Cmd Args : 10 20 30 40 20
Search element 20 is found at position : 1
Cmd Args : 15 25 18 9
Search element 9 is not found

q11044/LinearSearch.java | q11044/LinearSearchMair

```java
package q11044;

public class LinearSearch{

    public static int linearS(int[] array, int key){

        int sizeOfArray = array.length;

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

            if(array[i] == key){

                return i;
            }
        }
        return -1;
    }
    public static void linearSearch(int[] array, int key){

        int subscribe = linearS(array, key);

        if(subscribe == -1){

            System.out.println("Search element "+key+" is not found");
        } else{

            System.out.println("Search element "+key+" is found at position : "+subscribe);
        }
    }
}
```

Q2. Write a Java program to Search an element using Binary Search

Binary search is **faster** than **linear search**, as it uses **divide and conquer** technique and it works on the sorted list either in ascending or descending order.

Binary search (or) Half-interval search (or) Logarithmic search is a search algorithm that finds the position of a **key element** within a sorted array.

Binary search compares the **key element** to the **middle element** of the array; if they are **unequal**, the half in which the **key element** cannot lie is eliminated and the search continues on the remaining half until it is successful.

The working procedure for **binary search** is as follows:

1. Let us consider an array of **n** elements and a **key element** which is going to be search in the list of elements.
2. The main principle of binary search has first divided the list of elements into two halves.
3. Compare the **key element** with the **middle element**.
4. If the comparison result is **true** the print the **index position** where the **key element** has found and stop the process.
5. If the **key element** is greater than the **middle element** then search the key element in the second half.
6. If the **key element** is less than the **middle element** then search the key element in the first half.
7. Repeat the same process for the sub lists depending upon whether **key** is in the **first half** or **second half** of the list until a match is found (or) until all the elements in that half have been searched.

Let us consider an example of array numbers "**50 20 40 10 80**", and the key element is to find is **10**.

Search - 1 :
First **Sort** the given array elements by using any one of the sorting technique.
After sorting the elements in the array are **10 20 40 50 80** and initially **low = 0**, **high = 4**.
Search - 2 :
Compare **10** with middle element i.e., **(low + high) / 2** = **(0 + 4) / 2** = **4 / 2** = **2**, **a[2]** is **40**.
Here **10** < **40** so search the element in the left half of the element **40**. So **low = 0**, **high = mid - 1 = 2 - 1 = 1**.
Search - 3 :
Compare **10** with middle element i.e., **(low + high) / 2** = **(0 + 1) / 2** = **1 / 2** = **0**, **a[0]** is **10**.
Here **10 == 10** so print the index 0 where the element has found and stop the process

Write a class BinarySearch with a **public** method binarySearch that takes two parameters an array of type int[] and a key of type int. Write a code to search the key element within the array elements by using binary search technique.

Examples for your understanding:
Cmd Args : 10 1 2 3 4 5 4
Search element 4 is found at position : 4
Cmd Args : 10 8 12 11 9
Search element 9 is not found

q11045/BinarySearch.java | q11045/BinarySearchMair

```java
package q11045;

public class BinarySearch{
    public static int binaryS(int[] array, int key){
        int low = 0;
        int high = array.length - 1;

        while(low <= high){

            int mid = low +(high - low) / 2;
            if(array[mid] == key){
                return mid;
            }

            if(array[mid] < key){

                low = mid + 1;
            } else{

                high = mid - 1;
            }
        }
        return -1;
    }
    public static void binarySearch(int[] array, int key){

        int subscribe = binaryS(array, key);

        if(subscribe == -1){

            System.out.println("Search element "+key+" is not found");
        } else{

            System.out.println("Search element "+key+" is found at position : "+subscribe);
        }
    }
}
```