Bubble Sort
++++++++++++++++++++++

```
import java.util.Arrays;

/*

* In First loop arr.length -1 because,
  if we sort arr.length-1 elements then the last element
  automatically gets its required position

* In the second loop condition,
 -1 because to prevent ArrayIndexOutOfBoundsException
 -i because on every pass largest element bubbles at end of array,
 so again comparing with that part is unnecessary
 */

public class Practice1{

 public static void bubbleSort(int[] arr) {
  if(arr.length==0 || arr.length==1) {
   return;
  }
  int temp = 0;
  for(int i = 0;i<arr.length-1;i++) {
   for(int j=0;j<arr.length-1-i;j++) {
    if(arr[j] > arr[j+1]) {
     temp = arr[j];
     arr[j] = arr[j+1];
     arr[j+1] = temp;
    }
   }
  }
 }

 public static void main(String[] args) {
  int[] arr = {14, 78, 45, 34, 56, 100, 78, -9};

  bubbleSort(arr);
  System.out.println(Arrays.toString(arr));
 }
}
```

##

Selection Sort
++++++++++++++++++++++

```
package com.practice;

import java.util.Arrays;
```

```
/*

here in each pass we are selecting
minimum element and placing it in appropriate position

*/

public class Practice1{

 public static void selectionSort(int[] arr) {
  if(arr.length==0) {
   System.out.println("Empty");
   return;
  }
  for(int i=0;i<arr.length-1;i++) {
   int min = i;
   for(int j = i+1;j<arr.length;j++) {
    if(arr[j] < arr[min]) {
     min = j;
    }
   }
   /*if min == i then no need for swapping as it is already in its appropriate position*/
   if(min != i) {
    int temp = arr[i];
    arr[i] = arr[min];
    arr[min] = temp;

   }
  }
 }

 public static void main(String[] args) {
  int[] arr = {14, 78, 45, 34, 56, 100, 78, -9};

  selectionSort(arr);
  System.out.println(Arrays.toString(arr));
 }
}
```

##

Insertion Sort
++++++++++++++++++++++

```
package com.practice;

import java.util.Arrays;

/*

12,| 10, 56, 34, 89, 70
10, 12, | 56, 34, 89, 70
```

10, 12, 56, | 34, 89, 70
10, 12, 34, 56, | 89, 70
10, 12, 34, 56, 89, | 70
10, 12, 34, 56, 70, 89

Here we will select a element store it in temp;
then INSERT at appropriate position in previous sorted array
*/

```
public class Practice1{

 public static void insertionSort(int[] arr) {
  if(arr.length==0) {
   System.out.println("Empty");
   return;
  }
  int temp, j;
  for(int i =1;i<arr.length;i++) {
   temp = arr[i];
   j = i-1;
   while(j >=0 && arr[j] >temp) {
    arr[j+1] = arr[j];
    j--;
   }
   arr[j+1] = temp;
  }
 }

 public static void main(String[] args) {
  int[] arr = {14, 78, 45, 34, 56, 100, 78, -9};

  insertionSort(arr);
  System.out.println(Arrays.toString(arr));
 }
}
```

##

Merge Sort
++++++++++++++++++++++
package com.practice;

```
import java.util.Arrays;

/*
i -> to mark index of left subarray
j -> to mark index of right subarray
k -> to mark index of merged array

*/

public class Practice1{
```

```
public static void merge(int[] arr, int[] left, int[] right) {
 int i=0, j=0, k=0;
 while(i < left.length && j < right.length) {
  if(left[i] < right[j]) {
   arr[k] = left[i];
   k++;
   i++;
  }
  else {
   arr[k] = right[j];
   k++;
   j++;
  }
 }

 while(i < left.length) {
  arr[k] = left[i];
  k++;
  i++;
 }
 while(j < right.length) {
  arr[k] = right[j];
  k++;
  j++;
 }
}

public static void mergeSort(int[] arr) {
 if(arr.length==0) {
  System.out.println("Empty");
  return;
 }
 if(arr.length<2) {
  return;
 }

 int mid = arr.length/2;
 int[] left = new int[mid]; //index 0 to mid-1 elements are part of left sub-array
 int[] right = new int[arr.length-mid]; //index mid to last elements are part of right sub-array

 for(int i=0;i<mid;i++) {
  left[i] = arr[i];
 }
 for(int i=mid;i<arr.length;i++) {
  right[i-mid] = arr[i];
 }

 mergeSort(left);
 mergeSort(right);
 merge(arr, left, right);

}

public static void main(String[] args) {
```

```
 int[] arr = {14, 78, 45, 34, 56, 100, 70, -9, 66};

 mergeSort(arr);
 System.out.println(Arrays.toString(arr));
 }
}
```

##

Quick Sort
++++++++++++++++++++++

```
package com.practice;

import java.util.Arrays;

/*
Partition Algorithm:
select last element as pivot
And using partition algo we will place pivot in such a position
such that all data to the left of pivot as less than or equal to pivot and
all data to the right of pivot are greater than pivot

Partition algorithm return the index of sorted pivot element

then apply recursion on left of pivot and right of pivot

*/

public class Practice1{


 public static int partition(int[] arr, int from, int to) {
  int pivot = arr[to];
  int temp;
  int i = -1;//Used to point end index of left list of pivot
  for(int j = 0;j<to;j++) {
   if(arr[j] < pivot) {
    i++;
    temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
   }
  }
  i++;
  //dont swap arr[i] and pivot as pivot is a variable
  temp = arr[i];
  arr[i] = arr[to];
  arr[to] = temp;

  return i;
 }
```

```
 public static void quickSort(int[] arr, int from, int to) {
  if(arr.length ==0) {
   return;
  }
  //If there is atleast 2 elements
  int i;
  if(from < to) {
   i = partition(arr, from, to);
   quickSort(arr, from, i-1);
   quickSort(arr, i+1, to);
  }
 }


 public static void main(String[] args) {
  int[] arr = {14, 78, 45, 34, 56, 100, 70, -9, 66};

  quickSort(arr, 0, arr.length-1);
  System.out.println(Arrays.toString(arr));
 }
}
```

##

Heap Sort
++++++++++++++++++++++

```
package com.practice;

import java.util.*;

/*
 HeapSort(A){
 1.BuildHeap(A);
 2.for(A.length-1 to 1){
   swap(A[0], A[i])
   Heapify(A, 0)
   }
 }


 */



public class Practice1{

 public static void heapify(ArrayList<Integer> arr, int cur, int size) {
  int left = 2*cur+1;
```

```
 int right = 2*cur+2;
 int largest = cur;
 if(left < size && arr.get(left) > arr.get(largest)) {
  largest = left;
 }
 if(right < size && arr.get(right) > arr.get(largest)) {
  largest = right;
 }
 if(largest != cur) {
  //swap arr[largest] and arr[cur]
  int temp = arr.get(largest);
  arr.set(largest, arr.get(cur));
  arr.set(cur, temp);

  heapify(arr, largest, size);
 }
}

public static void buildHeap(ArrayList<Integer> arr) {
 for(int i = arr.size()/2-1;i>=0;i--) {
  heapify(arr, i, arr.size());
 }
 System.out.println(arr);
}

public static void heapSort(ArrayList<Integer> arr) {
 buildHeap(arr);

 for(int i = arr.size()-1;i>0;i--) {
  //swap arr[i] and arr[0]
  int temp = arr.get(0);
  arr.set(0, arr.get(i));
  arr.set(i, temp);
  heapify(arr, 0, i);
 }
 System.out.println(arr);

}
public static void main(String[] args) {
 ArrayList<Integer> arr = new ArrayList<>(Arrays.asList(14, 78, 45, 34, 56, 100, 78, -9));
 heapSort(arr);
 System.out.println(arr);
 }
}
```