

## BubbleSort

```
9  public class BubbleSort {
10      public static void bubbleSort(int[] arr){
11          for(int i = 0; i < arr.length; i++){
12              for(int j = 0; j < arr.length - i - 1; j++){
13                  if(arr[j] > arr[j + 1]){
14                      int temp = arr[j];
15                      arr[j] = arr[j + 1];
16                      arr[j + 1] = temp;
17                  }
18              }
19          }
20      }
21      public static void main(String[] args) {
22          int[] nums = {3, 2, 4, 1, 5};
23          bubbleSort(nums);
24
25          // printing Array after Soring
26          for(int values: nums){
27              System.out.print(values + " ");
28          }
29      }
30  }
31
32
```

## InsertionSortProgram

```
6 // Write a program to sort an Array using Insertion sort.
7
8 public class InsertionSortProgram {
9     public static void insertionSort(int[] arr){
10         for(int i = 1; i < arr.length; i++){
11             int current = arr[i];
12             int j = i - 1;
13             while(j >= 0 && current < arr[j]){
14                 arr[j + 1] = arr[j];
15                 j--;
16             }
17             //Placement
18             arr[j + 1] = current;
19         }
20     }
21     public static void main(String[] args) {
22         int[] arr = {7, 8, 3, 2, 1};
23         insertionSort(arr);
24         System.out.println(Arrays.toString(arr));
25     }
26 }
27
```

## MergeSortAlgorithms

```
11  public class MergeSortAlgorithms {  
    Codeium: Refactor | Explain | Generate Javadoc | X  
12  public static void merge(int[] arr, int start, int end){  
13      // Calculating mid element  
14      int mid = start + (end - start)/2;  
15  
16      // creating a temp Array  
17      int[] temp = new int[end - start + 1];  
18      int i = start;  
19      int j = mid + 1;  
20      int k = 0;  
21      while(i <= mid && j <= end){  
22          if(arr[i] <= arr[j]){  
23              temp[k++] = arr[i++];  
24          }  
25          else{  
26              temp[k++] = arr[j++];  
27          }  
28      }  
29      while(i <= mid){  
30          temp[k++] = arr[i++];  
31      }  
32      while(j <= end){  
33          temp[k++] = arr[j++];  
34      }  
35  
36      // Copying all Elements of temp into Original Array.  
37      for(int m = 0, n = start; m < temp.length; m++, n++){  
38          arr[n] = temp[m];  
39      }  
40  }  
41  
    Codeium: Refactor | Explain | Generate Javadoc | X  
42  public static void mergeSort(int[] arr, int start, int end){  
43      if(start >= end){  
44          return;  
45      }  
46      int mid = start + (end - start)/2;  
47      mergeSort(arr, start, mid);  
48      mergeSort(arr, mid + 1, end);  
49      merge(arr, start, end);  
50  }  
51  
    Codeium: Refactor | Explain | Generate Javadoc | X  
52  public static void main(String[] args) {  
53      int arr[] = { 41, 1, 32, 95, 7 };  
54      mergeSort(arr, 0, arr.length - 1);  
55  
56      // Printing the array  
57      System.out.println(Arrays.toString(arr));  
58  }  
59  }  
60
```

## QuickSortAlgorithms

```
10 public class QuickSortAlgorithms {  
    Codeium: Refactor | Explain | Generate Javadoc | X  
11     public static int partition(int[] arr, int low, int high){  
12         int pivot = arr[low];  
13         int i = low;  
14         int j = high;  
15         while(i < j){  
16             while(arr[i] <= pivot && i <= high - 1){  
17                 i++;  
18             }  
19             while(arr[j] > pivot && j >= low){  
20                 j--;  
21             }  
22             // Swapping ith index and jth index.  
23             if(i < j){  
24                 int temp = arr[i];  
25                 arr[i] = arr[j];  
26                 arr[j] = temp;  
27             }  
28         }  
29         // Swapping high with the pivot element.  
30         int temp = arr[low];  
31         arr[low] = arr[j];  
32         arr[j] = temp;  
33  
34         return j;  
35     }  
    Codeium: Refactor | Explain | Generate Javadoc | X  
36     public static void quicksort(int[] arr, int low, int high){  
37         if(low < high){  
38             int pivot = partition(arr, low, high);  
39             quicksort(arr, low, pivot - 1);  
40             quicksort(arr, pivot + 1, high);  
41         }  
42     }  
    Codeium: Refactor | Explain | Generate Javadoc | X  
43     public static void main(String[] args) {  
44         int n = 8;  
45         int arr[] = { 4, 6, 2, 5, 7, 8, 1, 3 };  
46         System.out.println("Before Quick Sort: ");  
47         for (int i = 0; i < n; i++) {  
48             System.out.print(arr[i] + " ");  
49         }  
50         System.out.println();  
51         quicksort(arr, 0, n - 1);  
52         System.out.println("After Quick Sort: ");  
53         for(int i = 0; i < n; i++){  
54             System.out.print(arr[i] + " ");  
55         }  
56     }  
57 }  
58
```

## BubbleSortUsingDepthOfRecursion

```
9  public class BubbleSortUsingDepthOfRecursion {
10     // This method is used to print the Array.
11     public static void print(int[] arr){
12         for(int i = 0; i < arr.length; i++){
13             System.out.print(arr[i] + " ");
14         }
15     }
16
17     // This is the inner recursion of the bubble sort. i.e. Recursion inside Recursion
18     public static void bubbleSortDepth(int[] arr, int i, int j){
19         if(j == arr.length - i - 1){
20             return;
21         }
22         if(arr[j] > arr[j + 1]){
23             int temp = arr[j];
24             arr[j] = arr[j + 1];
25             arr[j + 1] = temp;
26         }
27         bubbleSortDepth(arr, i, j + 1);
28     }
29
30     // This is the outer Recursion of Bubble Sort
31     public static void bubbleSort(int[] arr, int i, int j){
32         if(i == arr.length - 1){
33             return;
34         }
35         bubbleSortDepth(arr, i, j);
36         bubbleSort(arr, i + 1, 0);
37     }
38
39     // Main Method
40     public static void main(String[] args) {
41         int[] arr = {5, 3, 6, 1, 8, 2, 4};
42         bubbleSort(arr, 0, 0);
43         print(arr);
44     }
45 }
46
```

## RecursiveInsertionSort

```
7 public class RecursiveInsertionSort {
8     public static void insertionSort(int[] arr, int n, int i){
9         if(n == 0){
10             return;
11         }
12         int current = arr[i];
13         int j = i - 1;
14         while(j >= 0 && current < arr[j]){
15             arr[j + 1] = arr[j];
16             j--;
17         }
18         // placement
19         arr[j + 1] = current;
20         insertionSort(arr, n - 1, i + 1);
21     }
22     Codeium: Refactor | Explain | Generate Javadoc | X
23     public static void main(String[] args) {
24         int n = 8;
25         int arr[] = { 4, 6, 2, 5, 7, 8, 1, 3 };
26         System.out.println("Before Quick Sort: ");
27         for (int i = 0; i < n; i++) {
28             System.out.print(arr[i] + " ");
29         }
30         System.out.println();
31         insertionSort(arr, arr.length - 1, 1);
32         System.out.println("After Quick Sort: ");
33         for(int i = 0; i < n; i++){
34             System.out.print(arr[i] + " ");
35         }
36     }
37 }
38
```

## SelectionSort

```
7 public class SelectionSort {  
    Codeium: Refactor | Explain | Generate Javadoc | X  
8     public static void selectionSort(int[] arr){  
9         for(int i = 0; i < arr.length; i++){  
10             int smallest = i;  
11             for(int j = i + 1; j < arr.length; j++){  
12                 if(arr[smallest] > arr[j]){  
13                     smallest = j;  
14                 }  
15             }  
16             int temp = arr[i];  
17             arr[i] = arr[smallest];  
18             arr[smallest] = temp;  
19         }  
20     }  
    Codeium: Refactor | Explain | Generate Javadoc | X  
21     public static void main(String[] args) {  
22         int[] nums = {64, 25, 12, 22, 11};  
23         selectionSort(nums);  
24  
25         // printing Array after Sorting  
26         for(int values: nums){  
27             System.out.print(values + " ");  
28         }  
29     }  
30 }  
31
```