1) Bubble Sort

```
import java.util.*;
public class bubble
{
        public static void main(String args[])
        {
                int n,i,j,t;
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the size of array:");
                n=sc.nextInt();
                int a[]=new int[n];
                for(i=0;i<n;i++)
                {
                        System.out.println("Enter the value:");
                        a[i]=sc.nextInt();
                }
                for(i=0;i<n;i++)
                {
                        for(j=0;j<n-1-i;j++)
                        {
                                if(a[j]>a[j+1])
                                {
                                        t=a[j];
                                        a[j]=a[j+1];
                                        a[j+1]=t;
                                }
                        }
                }
                System.out.println("SORTED ARRAY:\n");
                for(i=0;i<n;i++)
                {
                        System.out.println(a[i]);
                }
        }
}
```

Output:

```
 Enter the size of array:

5

Enter the value:

4

Enter the value:

7

Enter the value:

9

Enter the value:

1

Enter the value:

5

SORTED ARRAY:


1

4

5

7

9
```

2) Modified Bubble Sort

```
import java.util.*;
public class bubble
{
	public static void main(String args[])
	{
		int n,i,j,t;
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the size of array:");
```

```
        n=sc.nextInt();
        int a[]=new int[n];
        for(i=0;i<n;i++)
        {
            System.out.println("Enter the value:");
            a[i]=sc.nextInt();
        }
        for(i=0;i<n;i++)
        {
            boolean swap=false;
            for(j=0;j<n-1-i;j++)
            {
                if(a[j]>a[j+1])
                {
                    t=a[j];
                    a[j]=a[j+1];
                    a[j+1]=t;
                    swap=true;
                }
            }
            if(swap=false) break;
        }
        System.out.println("SORTED ARRAY:\n");
        for(i=0;i<n;i++)
        {
            System.out.println(a[i]);
        }
    }
}
```

Output:

```
 Enter the size of array:

5

Enter the value:

48

Enter the value:
```

```
76

Enter the value:

92

Enter the value:

1

Enter the value:

45

SORTED ARRAY:


1

45

48

76

92
```

3) Insertion Sort

```
import java.util.*;
public class InsertionSort
{
        public static void main(String args[])
        {
                int n,i,j,temp;
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the number of element:");
                n=sc.nextInt();
                int a[]=new int[n];
                for(i=0;i<n;i++)
                {
                        System.out.println("Enter the value:");
                        a[i]=sc.nextInt();
                }
                for(i=1;i<n;i++)
                {
                        temp=a[i];
                        for(j=i-1;j>=0;j--)
                        {
                                if(a[j]<temp) break;
                                a[j+1]=a[j];
                        }
                        a[j+1]=temp;
                }
                System.out.println("Sorted array by insertion sort:");
                for(i=0;i<n;i++)
                {
                        System.out.println(a[i]);
                }
        }
}
```

Output:

```
 Enter the size of array:
5
Enter the value:
48
Enter the value:
70
Enter the value:
99
Enter the value:
1
Enter the value:
39
SORTED ARRAY:
```

```
1

39

48

70

99
```

4) Merge Sort

```
import java.util.*;
class MergeSort
{

    void merge(int arr[], int l, int m, int r)
    {
        int n1 = m - l + 1;
        int n2 = r - m;
        int L[] = new int[n1];
        int R[] = new int[n2];
        for (int i = 0; i < n1; ++i)
        {
            L[i] = arr[l + i];
        }
        for (int j = 0; j < n2; ++j)
        {
            R[j] = arr[m + 1 + j];
        }
        int i = 0, j = 0;
        int k = l;
        while (i < n1 && j < n2)
        {
            if (L[i] <= R[j])
            {
                arr[k] = L[i];
                i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}
void sort(int arr[], int l, int r)
{
    if (l < r)
    {
        int m =l+ (r-l)/2;
        sort(arr, l, m);
        sort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

static void printArray(int arr[])
{
    int n = arr.length;
    for (int i = 0; i < n; ++i)
    {
        System.out.print(arr[i] + " ");
        }
        System.out.println();
    }
    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the size of array:");
        int n=sc.nextInt();
        int arr[] = new int[n];
        for(int i=0;i<n;i++)
        {
            System.out.println("Enter the value:");
            arr[i]=sc.nextInt();
        }
        System.out.println("Given Array");
        printArray(arr);

        MergeSort ob = new MergeSort();
        ob.sort(arr, 0, arr.length - 1);

        System.out.println("\nSorted array");
        printArray(arr);
    }
}
```

OUTPUT:

```
Enter the size of array:
5
Enter the value:
5
Enter the value:
4
Enter the value:
3
Enter the value:
2
Enter the value:
1
```

Given Array
5 4 3 2 1

Sorted array
1 2 3 4 5

5) Quick Sort

```
import java.util.*;
public class QuickSort3
{
        public static void quickSort(int a[],int l, int u)
        {
                int j;
                if(l<u)
                {
                        j=partition(a,l,u);
                        quickSort(a,l,j-1);
                        quickSort(a,j+1,u);
                }
        }
        public static int partition(int a[], int l, int u)
        {
                int pivot,i,j,temp;
                pivot=a[u];
                i=l-1;
                for (j=l;j<u;j++)
        {
                        if (a[j]<=pivot)
                        {
                        i++;
                        temp = a[i];
                        a[i] = a[j];
                        a[j] = temp;
                        }
        }
        temp = a[i+1];
        a[i+1] = a[u];
```

```
                a[u] = temp;
                return i+1;
        }
        public static void main(String args[])
        {
                int n,i;
                Scanner sc = new Scanner(System.in);
                System.out.println("enter the size of array");
                n=sc.nextInt();
                int a[]=new int[n];
                for(i=0;i<=n-1;i++)
                {
                        System.out.println("enter the value");
                        a[i]=sc.nextInt();
                }
                quickSort(a,0,n-1);
                System.out.println("Sorted array is:");
                for(i=0;i<=n-1;i++)
                {
                        System.out.println(a[i]);
                }
        }
}
```

OUTPUT:

```
enter the size of array
5
enter the value
11
enter the value
21
enter the value
12
enter the value
32
enter the value
34
Sorted array is:
```

```
11
12
21
32
34
```

6) Heap Sort

```
import java.util.*;
public class heapsort
{
    public void sort(int arr[])
    {
        int n = arr.length;
        for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
       for (int i = n - 1; i > 0; i--)
        {
            int temp = arr[0];
                arr[0] = arr[i];
            arr[i] = temp;
            heapify(arr, i, 0);
        }
    }
    void heapify(int arr[], int n, int i)
    {
        int largest = i; // Initialize largest as root
        int l = 2 * i + 1; // left = 2*i + 1
        int r = 2 * i + 2; // right = 2*i + 2
        if (l < n && arr[l] > arr[largest])
                largest = l;
        if (r < n && arr[r] > arr[largest])
                largest = r;
        if (largest != i)
        {
                int swap = arr[i];
                arr[i] = arr[largest];
                arr[largest] = swap;
```

```
                                heapify(arr, n, largest);
                        }
                }
                static void printArray(int arr[])
                {
                         int n = arr.length;
                        for (int i = 0; i < n; ++i)
                                System.out.print(arr[i] + " ");
                        System.out.println();
                }
                public static void main(String args[])
                {
                        Scanner sc=new Scanner(System.in);
                        System.out.println("Enter the size of array:");
                        int n=sc.nextInt();
                        int arr[] = new int[n];
                        for(int i=0;i<n;i++)
                        {
                                System.out.println("Enter the value:");
                                arr[i]=sc.nextInt();
                        }
                        heapsort ob = new heapsort();
                        ob.sort(arr);
                        System.out.println("Sorted array is");
                        printArray(arr);
                }
        }
```

OUTPUT:

```
Enter the size of array:
5
Enter the value:
5
Enter the value:
6
Enter the value:
4
Enter the value:
```

```
7
Enter the value:
3
Sorted array is
3 4 5 6 7
```

7) Count Sort

```java
import java.util.*;
class CountingSort {

    static void countSort(int[] arr)
    {
        int max = Arrays.stream(arr).max().getAsInt();
        int min = Arrays.stream(arr).min().getAsInt();
        int range = max - min + 1;
        int count[] = new int[range];
        int output[] = new int[arr.length];
        for (int i = 0; i < arr.length; i++) {
            count[arr[i] - min]++;
        }

        for (int i = 1; i < count.length; i++) {
            count[i] += count[i - 1];
        }

        for (int i = arr.length - 1; i >= 0; i--) {
            output[count[arr[i] - min] - 1] = arr[i];
            count[arr[i] - min]--;
        }

        for (int i = 0; i < arr.length; i++) {
            arr[i] = output[i];
        }
    }

    static void printArray(int[] arr)
```

```
        {
                for (int i = 0; i < arr.length; i++) {
                        System.out.print(arr[i] + " ");
                }
                System.out.println("");
        }
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the size of array:");
                int n=sc.nextInt();
                int arr[] = new int[n];
                for(int i=0;i<n;i++)
                {
                        System.out.println("Enter the value:");
                        arr[i]=sc.nextInt();
                }

                countSort(arr);
                printArray(arr);
        }
}
```

OUTPUT:

```
Enter the size of array:
5
Enter the value:
5
Enter the value:
3
Enter the value:
7
Enter the value:
1
Enter the value:
9
1  3 5 7 9
```

8) Radix Sort

```
import java.util.Arrays;

class RadixSort {

  void countingSort(int array[], int size, int place) {
    int[] output = new int[size + 1];
    int max = array[0];
    for (int i = 1; i < size; i++) {
      if (array[i] > max)
        max = array[i];
    }
    int[] count = new int[max + 1];

    for (int i = 0; i < max; ++i)
      count[i] = 0;
    for (int i = 0; i < size; i++)
      count[(array[i] / place) % 10]++;
    for (int i = 1; i < 10; i++)
      count[i] += count[i - 1];
    for (int i = size - 1; i >= 0; i--) {
      output[count[(array[i] / place) % 10] - 1] = array[i];
      count[(array[i] / place) % 10]--;
    }
    for (int i = 0; i < size; i++)
      array[i] = output[i];
  }
  int getMax(int array[], int n) {
    int max = array[0];
    for (int i = 1; i < n; i++)
      if (array[i] > max)
        max = array[i];
    return max;
  }
  void radixSort(int array[], int size) {
    int max = getMax(array, size);
    for (int place = 1; max / place > 0; place *= 10)
      countingSort(array, size, place);
  }
  public static void main(String args[]) {
    int[] data = { 11, 43, 64, 23, 1, 45, 88 };
    int size = data.length;
    RadixSort rs = new RadixSort();
    rs.radixSort(data, size);
    System.out.println("Sorted Array in Ascending Order: ");
    System.out.println(Arrays.toString(data));
  }
}
```

OUTPUT:

```
Sorted Array in Ascending Order:
[1 ,11 ,23 ,43 ,45 , 64, 88]
```

9) Bucket Sort

```
import java.util.*;

public class Main
{
    public static int[] bucket_sort(int[] arr, int max_value)
    {
        int[] bucket = new int[max_value + 1];
        int[] sorted_arr = new int[arr.length];

        for (int i= 0; i <arr.length; i++)
            bucket[arr[i]]++;

        int pos = 0;
        for (int i = 0; i < bucket.length; i++)
            for (int j = 0; j < bucket[i]; j++)
                sorted_arr[pos++] = i;

        return sorted_arr;
    }


    static int maxValue(int[] arr)
    {
        int max_value = 0;
        for (int i = 0; i < arr.length; i++)
```

```
            if (arr[i] > max_value)
                max_value = arr[i];
        return max_value;
    }

    public static void main(String args[])
    {
        int[] arr ={8, 5, 3, 1, 9, 6, 0, 7, 4, 2, 10};
        int max_value = maxValue(arr);

        System.out.print("\nOriginal : ");
        System.out.println(Arrays.toString(arr));

        System.out.print("\nSorted : ");
        System.out.println(Arrays.toString(bucket_sort(arr,max_value)));

    }
}
```

OUTPUT:

```
Original: [8, 5, 3, 1, 9, 6, 0, 7, 4, 2, 10]
Sorted: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

		Swaps	Comparisons	Shifts	Total time(ms)
Bubble sort	Best case	-	190	-	0.0814
	Worst case	190	190	-	0.9842
	Random case	56	190	-	0.638
Modified bubble sort	Best case	-	190	-	0.0628
	Worst case	190	190	-	0.7894
	Random case	50	190	-	0.1295
Insertion sort	Best case	-	19	-	0.0354
	Worst case	-	19	190	0.6432
	Random case	-	19	61	0.0851
Merge sort	Best case	-	48	88	0.0039
	Worst case	-	40	88	0.0892
	Random case	-	44	88	0.0418
Quick sort	Best case	-	20	-	0.0013
	Worst case	19	20	-	0.2169
	Random case	11	20	-	0.0012
Heap sort	Best case	-	40	-	0.1565
	Worst case	60	40	-	0.2364
	Random case	45	40	-	0.2010
Count sort	Best case	-	-	-	0.0173
	Worst case	-	-	-	0.0974
	Random case	-	-	-	0.0712
Radix sort	Best case	-	-	-	0.0023
	Worst case	-	-	-	0.0189
	Random case	-	-	-	0.00138
Bucket sort	Best case	-	-	-	0.00563
	Worst case	-	-	-	0.0808
	Random case	-	-	-	0.0153

Sorting technique	*Best case*	*Worst case*	*Average case*
Basic bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$
Modified bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$
Merge sort	O(n logn)	O(n logn)	O(n logn)
Quick sort	O(n logn)	$O(n^2)$	O(n logn)
Heap sort	O(n logn)	O(n logn)	O(n logn)
Count sort	O(n+k)	O(n+k)	O(n+k)
Radix sort	O(m+n)	O(m+n)	O(m+n)
Bucket sort	O(n)	O(n)	O(n)

Q2)

A) DFS

```
import java.util.*;

class Graph {
  private LinkedList<Integer> adjLists[];
  private boolean visited[];

  // Graph creation
  Graph(int vertices) {
    adjLists = new LinkedList[vertices];
    visited = new boolean[vertices];

    for (int i = 0; i < vertices; i++)
      adjLists[i] = new LinkedList<Integer>();
  }

  // Add edges
  void addEdge(int src, int dest) {
    adjLists[src].add(dest);
  }

  // DFS algorithm
  void DFS(int vertex) {
    visited[vertex] = true;
    System.out.print(vertex + " ");

    Iterator<Integer> ite = adjLists[vertex].listIterator();
    while (ite.hasNext()) {
      int adj = ite.next();
      if (!visited[adj])
        DFS(adj);
    }
  }

  public static void main(String args[]) {
    Graph g = new Graph(7);

    g.addEdge(0, 1);
    g.addEdge(0, 3);
    g.addEdge(0,4);
    g.addEdge(0,6);
    g.addEdge(1,6);
    g.addEdge(1,0);
    g.addEdge(2,5);
    g.addEdge(2, 3);
    g.addEdge(3,2);
    g.addEdge(3,5);
    g.addEdge(3,6);
    g.addEdge(3,0);
    g.addEdge(4,0);
    g.addEdge(5,2);
    g.addEdge(5,6);
    g.addEdge(6,0);
    g.addEdge(6,1);
    g.addEdge(6,3);
    g.addEdge(6,5);

    System.out.println("Following is Depth First Traversal");

    g.DFS(0);
  }
}
```

```
OUTPUT:
Following is Depth First Traversal
0 1  6 3 2 5 4
```

B)

BFS

```
import java.util.*;

public class GraphBFS {
  private int V;
  private LinkedList<Integer> adj[];

  // Create a graph
  GraphBFS(int v) {
    V = v;
    adj = new LinkedList[v];
    for (int i = 0; i < v; ++i)
      adj[i] = new LinkedList();
  }

  // Add edges to the graph
  void addEdge(int v, int w) {
    adj[v].add(w);
  }

  // BFS algorithm
  void BFS(int s) {
    boolean visited[] = new boolean[V];

    LinkedList<Integer> queue = new LinkedList();

    visited[s] = true;
    queue.add(s);

    while (queue.size() != 0) {
      s = queue.poll();
      System.out.print(s + " ");

      Iterator<Integer> i = adj[s].listIterator();
      while (i.hasNext()) {
        int n = i.next();
        if (!visited[n]) {
          visited[n] = true;
          queue.add(n);
        }
      }
    }
  }

  public static void main(String args[]) {
    GraphBFS g = new GraphBFS(7);

    g.addEdge(0, 4);
    g.addEdge(1,4);

    g.addEdge(1, 3);

    g.addEdge(1,6);

    g.addEdge(1,5);

    g.addEdge(1,2);

    g.addEdge(2,1);

    g.addEdge(2,5);

    g.addEdge(3,1);

    g.addEdge(4,1);

    g.addEdge(4,5);

    g.addEdge(4,6);

    g.addEdge(5,1);

    g.addEdge(5,2);

    g.addEdge(5, 4);

    g.addEdge(5,6);

    g.addEdge(6,1);

    g.addEdge(6,4);

    g.addEdge(6,5);


    System.out.println("Following is Breadth First Traversal ");


    g.BFS(0);
  }
}
```

OUTPUT:

```
Following is Breadth First Traversal
```

0 4 1 5 6 3 2