**Practical no :3**

**NAME:MANSI KIRLOSKAR**

**ROLL NO:38**

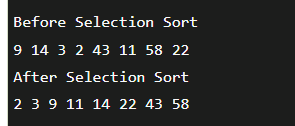
**GREEDY ALGORITHM**

**1]SELECTION SORT**

**CODE:**

1. **public** **class** SelectionSortExample {
2. **public** **static** **void** selectionSort(**int**[] arr){
3. **for** (**int** i = 0; i < arr.length - 1; i++)
4. {
5. **int** index = i;
6. **for** (**int** j = i + 1; j < arr.length; j++){
7. **if** (arr[j] < arr[index]){
8. index = j;//searching for lowest index
9. }
10. }
11. **int** smallerNumber = arr[index];
12. arr[index] = arr[i];
13. arr[i] = smallerNumber;
14. }
15. }
17. **public** **static** **void** main(String a[]){
18. **int**[] arr1 = {9,14,3,2,43,11,58,22};
19. System.out.println("Before Selection Sort");
20. **for**(**int** i:arr1){
21. System.out.print(i+" ");
22. }
23. System.out.println();
25. selectionSort(arr1);//sorting array using selection sort
27. System.out.println("After Selection Sort");
28. **for**(**int** i:arr1){
29. System.out.print(i+" ");
30. }
31. }
32. }

OUTPUT:

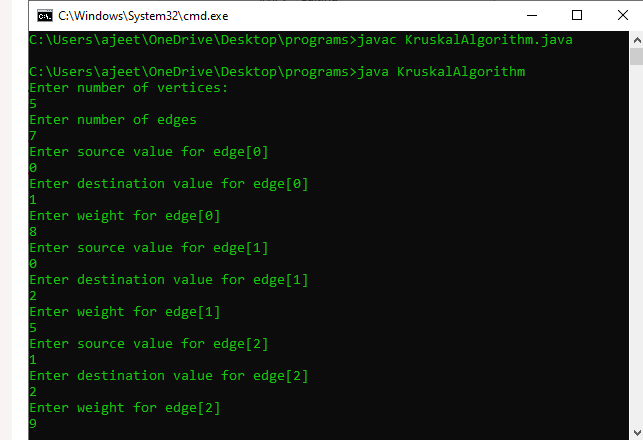


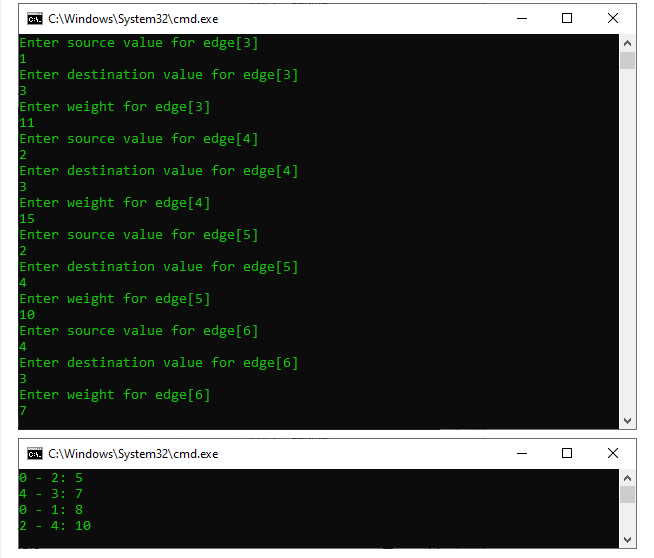
**2] KRUSKAL ALGORITHM**

**CODE:**

1. **import** Java.util.\*;
3. // create KruskalAlgorithm class to create minimum spanning tree of the given graph
4. **class** KruskalAlgorithm {
5. //create class Edge to create an edge of the graph that implements Comparable interface
6. **class** Edge **implements** Comparable<Edge> {
7. **int** source, destination, weight;
9. **public** **int** compareTo(Edge edgeToCompare) {
10. **return** **this**.weight - edgeToCompare.weight;
11. }
12. };
14. // create class Subset for union
15. **class** Subset {
16. **int** parent, value;
17. };
19. //initialize vertices, edges and edgeArray
20. **int** vertices, edges;
21. Edge edgeArray[];
23. // using constructor to create a graph
24. KruskalAlgorithm(**int** vertices, **int** edges) {
25. **this**.vertices = vertices;
26. **this**.edges = edges;
27. edgeArray = **new** Edge[**this**.edges];
28. **for** (**int** i = 0; i < edges; ++i)
29. edgeArray[i] = **new** Edge();  //create edge for all te edges given by the user
30. }
32. // create applyKruskal() method for applying Kruskal's Algorithm
33. **void** applyKruskal() {
35. // initialize finalResult array to store the final MST
36. Edge finalResult[] = **new** Edge[vertices];
37. **int** newEdge = 0;
38. **int** index = 0;
39. **for** (index = 0; index < vertices; ++index)
40. finalResult[index] = **new** Edge();
42. // using sort() method for sorting the edges
43. Arrays.sort(edgeArray);
45. // create an array of the vertices of type Subset for the subsets of vertices
46. Subset subsetArray[] = **new** Subset[vertices];
48. // aloocate memory to create vertices subsets
49. **for** (index = 0; index < vertices; ++index)
50. subsetArray[index] = **new** Subset();
52. // it is used to create subset with single element
53. **for** (**int** vertex = 0; vertex < vertices; ++vertex) {
54. subsetArray[vertex].parent = vertex;
55. subsetArray[vertex].value = 0;
56. }
57. index = 0;
59. // use for loop to pick the smallers edge from the edges and increment the index for next iteration
60. **while** (newEdge < vertices - 1) {
61. // create an instance of Edge for next edge
62. Edge nextEdge = **new** Edge();
63. nextEdge = edgeArray[index++];
65. **int** nextSource = findSetOfElement(subsetArray, nextEdge.source);
66. **int** nextDestination = findSetOfElement(subsetArray, nextEdge.destination);
68. //if the edge doesn't create cycle after including it, we add it in the result and increment the index
69. **if** (nextSource != nextDestination) {
70. finalResult[newEdge++] = nextEdge;
71. performUnion(subsetArray, nextSource, nextDestination);
72. }
73. }
74. **for** (index = 0; index < newEdge; ++index)
75. System.out.println(finalResult[index].source + " - " + finalResult[index].destination + ": " + finalResult[index].weight);
76. }
78. // create findSetOfElement() method to get set of an element
79. **int** findSetOfElement(Subset subsetArray[], **int** i) {
80. **if** (subsetArray[i].parent != i)
81. subsetArray[i].parent = findSetOfElement(subsetArray, subsetArray[i].parent);
82. **return** subsetArray[i].parent;
83. }
85. // create performUnion() method to perform union of two sets
86. **void** performUnion(Subset subsetArray[], **int** sourceRoot, **int** destinationRoot) {
88. **int** nextSourceRoot = findSetOfElement(subsetArray, sourceRoot);
89. **int** nextDestinationRoot = findSetOfElement(subsetArray, destinationRoot);
91. **if** (subsetArray[nextSourceRoot].value < subsetArray[nextDestinationRoot].value)
92. subsetArray[nextSourceRoot].parent = nextDestinationRoot;
93. **else** **if** (subsetArray[nextSourceRoot].value > subsetArray[nextDestinationRoot].value)
94. subsetArray[nextDestinationRoot].parent = nextSourceRoot;
95. **else** {
96. subsetArray[nextDestinationRoot].parent = nextSourceRoot;
97. subsetArray[nextSourceRoot].value++;
98. }
99. }
101. //main() method starts
102. **public** **static** **void** main(String[] args) {
104. **int** v, e;
105. //create scanner class object to get input from user
106. Scanner sc = **new** Scanner(System.in);
108. //show custom message
109. System.out.println("Enter number of vertices: ");
111. //store user entered value into variable v
112. v = sc.nextInt();
114. //show custom message
115. System.out.println("Enter number of edges");
117. //store user entered value into variable e
118. e = sc.nextInt();
120. KruskalAlgorithm graph = **new** KruskalAlgorithm(v, e);        // use for creating Graph
122. **for**(**int** i = 0; i < e; i++){
123. System.out.println("Enter source value for edge["+ i +"]");
124. graph.edgeArray[i].source = sc.nextInt();
126. System.out.println("Enter destination value for edge["+ i +"]");
127. graph.edgeArray[i].destination = sc.nextInt();
129. System.out.println("Enter weight for edge["+i+"]");
130. graph.edgeArray[i].weight = sc.nextInt();
131. }
133. // call applyKruskal() method to get MST
134. graph.applyKruskal();
135. }
136. }

**OUTPUT:**





**3]PRIM’S ALGORITHM:**

**CODE**:

1. **import** java.lang.\*;
2. **import** java.util.\*;
3. **import** java.io.\*;
5. //creating MinimumSpanningTreeExample class to implement Prim's algorithm in Java
6. **class** MinimumSpanningTreeExample {
7. // Define the count of vertices available in the graph
8. **private** **static** **final** **int** countOfVertices = 9;
10. // create findMinKeyVertex() method for finding the vertex v that has minimum key-value and that is not added MST yet
11. **int** findMinKeyVertex(**int** keys[], Boolean setOfMST[])
12. {
13. // Initialize min value and its index
14. **int** minimum\_index = -1;
15. **int** minimum\_value = Integer.MAX\_VALUE;
17. //iterate over all vertices to find minimum key-value vertex
18. **for** (**int** vertex = 0; vertex < countOfVertices; vertex++)
19. **if** (setOfMST[vertex] == **false** && keys[vertex] < minimum\_value) {
20. minimum\_value = keys[vertex];
21. minimum\_index = vertex;
22. }
24. **return** minimum\_index;
25. }
27. // create showMinimumSpanningTree for printing the constructed MST stored in mstArray[]
28. **void** showMinimumSpanningTree(**int** mstArray[], **int** graphArray[][])
29. {
30. System.out.println("Edge \t\t Weight");
31. **for** (**int** j = 1; j < countOfVertices; j++)
32. System.out.println(mstArray[j] + " <-> " + j + "\t \t" + graphArray[j][mstArray[j]]);
33. }
35. // create designMST() method for constructing and printing the MST. The graphArray[][] is an adjacency matrix that defines the graph for MST.
36. **void** designMST(**int** graphArray[][])
37. {
38. // create array of size total number of vertices, i.e., countOfVertices for storing the MST
39. **int** mstArray[] = **new** **int**[countOfVertices];
41. // create keys[] array for selecting an edge having minimum weight in cut
42. **int** keys[] = **new** **int**[countOfVertices];
44. // create setOfMST array of type boolean for representing the set of vertices included in MST
45. Boolean setOfMST[] = **new** Boolean[countOfVertices];
47. // set the value of the keys to infinite
48. **for** (**int** j = 0; j < countOfVertices; j++) {
49. keys[j] = Integer.MAX\_VALUE;
50. setOfMST[j] = **false**;
51. }
53. // set value 0 to the 1st vertex because first vertes always include in MST.
54. keys[0] = 0; // it select as first vertex
55. mstArray[0] = -1; // set first value of mstArray to -1 to make it root of MST
57. // The vertices in the MST will be equal to the countOfVertices
58. **for** (**int** i = 0; i < countOfVertices - 1; i++) {
59. // select the vertex having minimum key and that is not added in the MST yet from the set of vertices
60. **int** edge = findMinKeyVertex(keys, setOfMST);
62. // Add the selected minimum key vertex to the setOfMST
63. setOfMST[edge] = **true**;
65. // change the key value and the parent index of all the adjacent vertices of the selected vertex. The vertices that are not yet included in Minimum Spanning Tree are only considered.
66. **for** (**int** vertex = 0; vertex < countOfVertices; vertex++)
68. // The value of the graphArray[edge][vertex] is non zero only for adjacent vertices of m setOfMST[vertex] is false for vertices not yet included in Minimum Spanning Tree
69. // when the value of the graphArray[edge][vertex] is smaller than key[vertex], we update the key
70. **if** (graphArray[edge][vertex] != 0 && setOfMST[vertex] == **false** && graphArray[edge][vertex] < keys[vertex]) {
71. mstArray[vertex] = edge;
72. keys[vertex] = graphArray[edge][vertex];
73. }
74. }
76. // print the constructed Minimum Spanning Tree
77. showMinimumSpanningTree(mstArray, graphArray);
78. }
79. //main() method start
80. **public** **static** **void** main(String[] args)
81. {
83. MinimumSpanningTreeExample mst = **new** MinimumSpanningTreeExample();
84. **int** graphArray[][] = **new** **int**[][]{{ 0, 4, 0, 0, 0, 0, 0, 8, 0 },
85. { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
86. { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
87. { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
88. { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
89. { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
90. { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
91. { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
92. { 0, 0, 2, 0, 0, 0, 6, 7, 0 }};
94. // Print the Minimum Spanning Tree solution
95. mst.designMST(graphArray);
96. }
97. }

**OUTPUT:**

