**Algorithms**

**Laboratory Task-2**

**Submission Deadline** – As announced in the class

**Submission Guidelines**-

* Rename the file to your id only. If your id is 18-XXXXX-1, then the file name must be 18-XXXXX-1.docx.
* Must submit within the given deadline in VUES to the section named Lab Tak-2
* Must include resources for all the section named ‘Code’ and ‘Output (screenshot)’ in the table.

|  |
| --- |
| Question-1 – **Implement Insertion sort.** |
| **Pseudocode**  INSERTION-SORT(A)  for j = 2 to n  key ← A [j]  j ← i – 1  while i > 0 and A[i] > key  A[i+1] ← A[i]  i ← i – 1  A[j+1] ← key |
| **Code** |
| **Output (Screenshot)** |

|  |
| --- |
| Question-2 – **Implement Counting Sort.** |
| **Description**  Let us understand it with the help of an example.  For simplicity, consider the data in the range 0 to 9.  Input data: 1, 4, 1, 2, 7, 5, 2  1) Take a count array to store the count of each unique object.  Index: 0 1 2 3 4 5 6 7 8 9  Count:0 2 2 0 1 1 0 1 0 0  2) Modify the count array such that each element at each index  stores the sum of previous counts.  Index: 0 1 2 3 4 5 6 7 8 9  Count: 0 2 4 4 5 6 6 7 7 7  The modified count array indicates the position of each object in the output sequence.  3) Output each object from the input sequence followed by decreasing its count by 1.  Process the input data: 1, 4, 1, 2, 7, 5, 2. Position of 1 is 2. Put data 1 at index 2 in output. Decrease count by 1 to place next data 1 at an index 1 smaller than this index. |
| **Pseudocode**  CountingSort(A)  for i = 0 to k do  c[i] = 0  //Storing Count of each element  for j = 0 to n do  c[A[j]] = c[A[j]] + 1  // Change C[i] such that it contains actual position of these elements in output array  for i = 1 to k do  c[i] = c[i] + c[i-1]  //Build Output array from C[i]  for j = n-1 down to 0 do  B[ c[A[j]]-1 ] = A[j]  c[A[j]] = c[A[j]] - 1  end func |
| **Code** |
| **Output (Screenshot)** |

|  |
| --- |
| Question-3 – **Implement Binary Search** |
| **Pseudocode**  // initially called with low = 0, high = N – 1  BinarySearch\_Right(A[0..N-1], value, low, high) {  if (high < low)  return low  mid = (low + high) / 2  if (A[mid] > value)  return BinarySearch\_Right(A, value, low, mid-1)  else  return BinarySearch\_Right(A, value, mid+1, high)  } |
| **Code** |
| **Output (Screenshot)** |