**Name : Murali srinivasan S  
Dept : CSE**

**Test : Practice\_set\_05**

1. Bubble Sort   
     
   **Code :  
   import java.util.Arrays;**

**public class BubbleSort {**

**/\***

**\* -----OTHER NAMES-----**

**\* Other names for Bubble Sort : (Sinking Sort , Exchange Sort)**

**\***

**\* ------COMPLEXITIES------**

**\* Time Complexity : (1) Best Case = O(n) ; (2) Worst Case = O(n^2)**

**\* Space Complexity : O(1) (means no extra space required to sort the numbers and this property is also known as "in-place sorting")**

**\***

**\* -----WORKING PROCESS-----**

**\* How it works? : (Consider the given array needs to be sorted in ascending order)**

**\***

**\* - On each pass in the bubble sort algorithm , it will check the current element with the next element.**

**\* - If the current element is greater than the next element then swap them.**

**\* - Otherwise, make the next element as the current element.**

**\* - At the end of each pass the largest element in the unsorted part of the array will be placed in it's appropriate position.**

**\* - This means on every pass, the unsorted part will gradually reduce by one and the sorted part will gradually increase by one.**

**\* (Example)**

**\* - On end of first pass, [(3,4,2,1), (5)] (note that the first part is unSorted and the second pass is sorted)**

**\* - On end of second pass, [(2,3,1), (4,5)]**

**\* - If the array given to us is in the complete opposite order what we want it to be, then we this algorithm will make a maximum of (n-1) passes to sort the array of the desired order.**

**\***

**\* -----SPECIAL CHARACTERISTICS-----**

**\* Algorithm's nature : Stable sorting Algorithm**

**\***

**\* - Stabel Sorting Algorithm : Order should be the same when the values in the array are same.**

**\*/**

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

**int[] arr = {5,4,3,2,1};**

**bubbleSort(arr);**

**System.out.println(Arrays.toString(arr));**

**}**

**static void bubbleSort(int[] arr){**

**int n = arr.length;**

**boolean isSorted = true;**

**for(int i=0; i<n; i++){**

**for(int j=1; j<n-i; j++){**

**if(arr[j-1] >= arr[j]){**

**swap(arr, j-1, j);**

**isSorted = false;**

**}**

**}**

**if(isSorted){**

**break;**

**}**

**}**

**}**

**static void swap(int[] arr, int a, int b){**

**int temp = arr[a];**

**arr[a] = arr[b];**

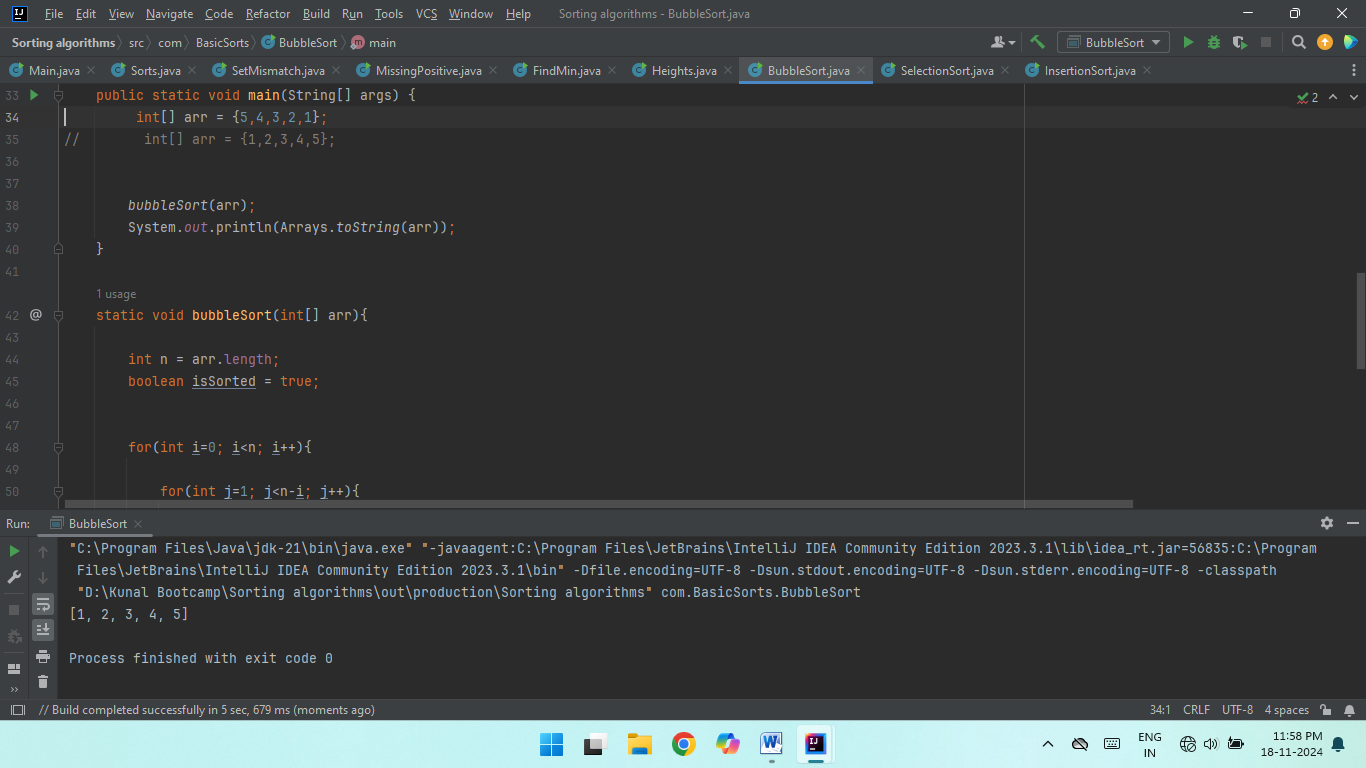
**arr[b] = temp;**

**}**

**}**

**Complexities :**Time Complexity : O(n^2)

Space Complexity : O(1)

**Output :**  


1. Quick Sort   
    **Code :  
     
   package Sortings;**

**import java.util.Arrays;**

**public class QuickSort {**

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

**int[] nums = {5,4,3,2,1};**

**quickSort(nums, 0, nums.length-1);**

**System.out.println(Arrays.toString(nums));**

**}**

**static void quickSort(int[] nums, int low, int hi){**

**// Base Case:**

**if(low >= hi){**

**return;**

**}**

**int s = low;**

**int e = hi;**

**int mid = s + (e-s)/2;**

**int pivot = nums[mid];**

**while (s <= e){**

**// also a reason, why? If it is already sorted it will not swap.**

**while (nums[s] < pivot){**

**s++;**

**}**

**while (nums[e] > pivot){**

**e--;**

**}**

**// This check is mandatory because the while loop can only sense this violation when the whole body of it gets executed.**

**if(s <= e){**

**int temp = nums[s];**

**nums[s] = nums[e];**

**nums[e] = temp;**

**s++;**

**e--;**

**}**

**}**

**// now my pivot is at the correct index, please sort two halves now.**

**quickSort(nums, low, e);**

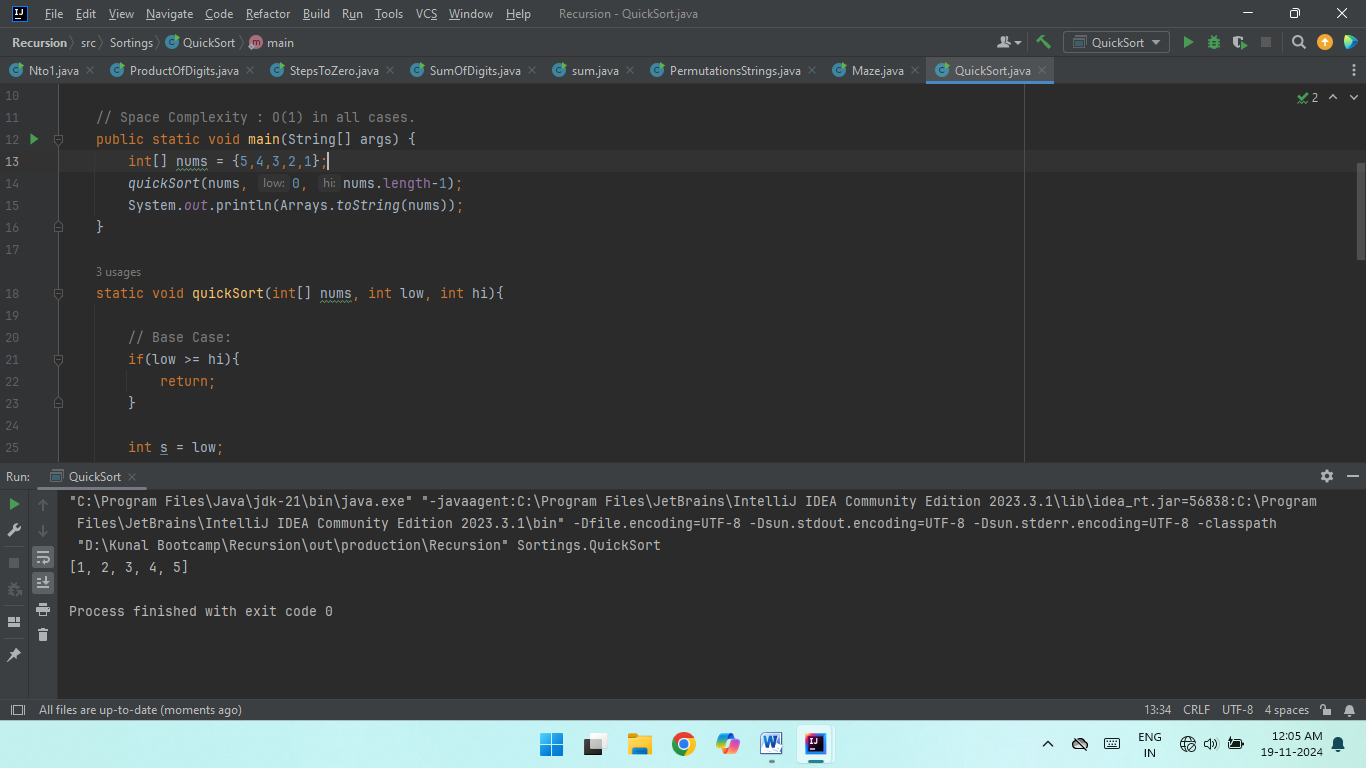
**quickSort(nums, s, hi);**

**}**

**}**

**Complexities :**Time Complexity : O(n^2)

Space Complexity : O(1)

**Output :**  


1. Non Repeating Characters

**Code :  
  
class Solution {**

**static char nonRepeatingChar(String s) {**

**HashMap<Character, Integer> mp = new HashMap<>();**

**for(int i=0; i<s.length(); i++){**

**if(mp.containsKey(s.charAt(i))){**

**mp.put(s.charAt(i), mp.get(s.charAt(i)) + 1);**

**} else { mp.put(s.charAt(i), 1); }**

**}**

**// for(Character ch : mp.keySet()){**

**// if(mp.get(ch) == 1) { return (char)ch; }**

**// }**

**for(int j=0; j<s.length(); j++){**

**char ch = s.charAt(j);**

**if(mp.get(ch) == 1) { return ch; }**

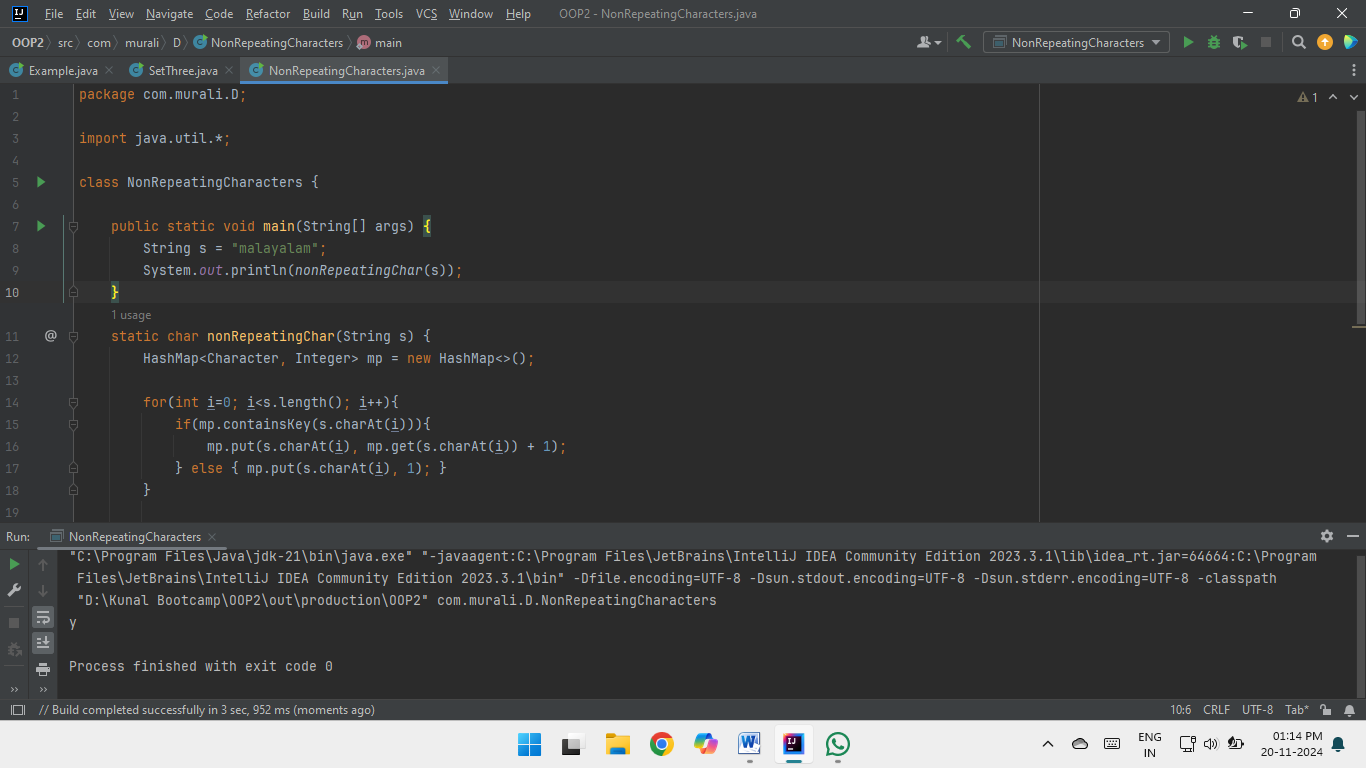
**}**

**return '$';**

**}**

**}**

**Complexities :**Time Complexity : O(n)

Space Complexity : O(1)  
  
**Output :**

1. Edit Distance

**Code :  
  
class Solution {**

**public int minDistance(String word1, String word2) {**

**return editDist(word1, word2);**

**}**

**public int editDist(String s, String t){**

**int n = s.length();**

**int m = t.length();**

**int[][] memo = new int[n+1][m+1];**

**for(int[] mo : memo){ Arrays.fill(mo, -1); }**

**return editDistRec(s, t, n, m, memo);**

**}**

**public int editDistRec(String s, String t, int n, int m, int[][] memo){**

**if(n == 0){ return m; }**

**if(m == 0){ return n; }**

**if(memo[n][m] != -1){ return memo[n][m]; } // Make sure the indices are appropriately specified.**

**if(s.charAt(n-1) == t.charAt(m-1)){**

**memo[n][m] = editDistRec(s, t, n-1, m-1, memo);**

**} else{**

**int insert = editDistRec(s, t, n, m-1, memo);**

**int remove = editDistRec(s, t, n-1, m, memo);**

**int replace = editDistRec(s, t, n-1, m-1, memo);**

**int min\_rm\_rp = Math.min(remove, replace);**

**memo[n][m] = 1 + Math.min(insert, min\_rm\_rp); // That '+1' is important.**

**}**

**return memo[n][m];**

**}**

**}**

**Complexities :**Time Complexity : O(n\*m)

Space Complexity : O(n\*m)  
  
**Output :**// Image to be added

1. K largest elements

**Code :  
  
class Solution{**

**static List<Integer> kLargest(int arr[], int k) {**

**// write code here**

**List<Integer> ls = new ArrayList<>();**

**Arrays.sort(arr);**

**for(int i=arr.length-1; i>=arr.length-k; i--){**

**ls.add(arr[i]);**

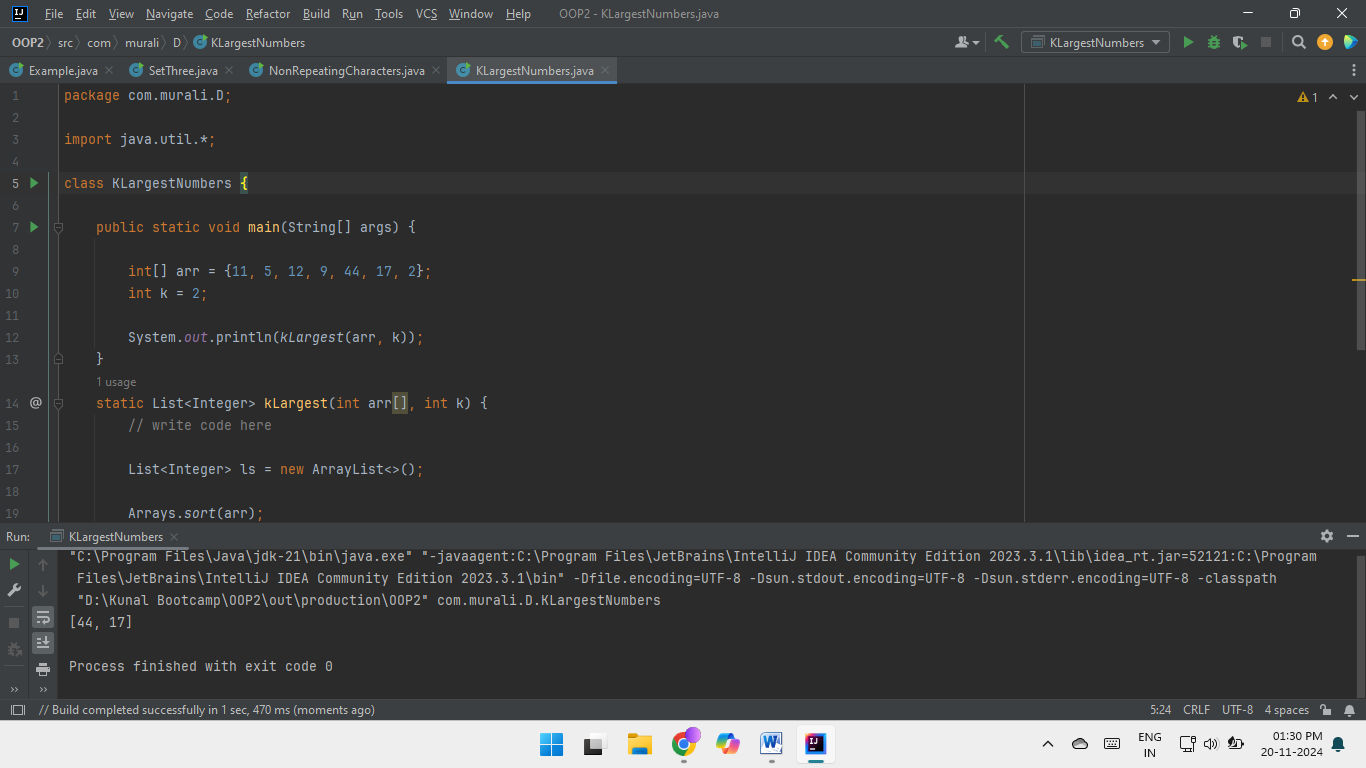
**}**

**return ls;**

**}**

**}**

**Complexities :**Time Complexity : O(n)

Space Complexity : O(1)  
  
**Output :**

1. Form the Largest Number

**Code :**

**class Solution {**

**public static boolean myCompare(String s1, String s2) {**

**return (s1 + s2).compareTo(s2 + s1) > 0;**

**}**

**public static String findLargest(int[] arr) {**

**// Convert the array of integers to an array of strings**

**ArrayList<String> numbers = new ArrayList<>();**

**for (int ele : arr) {**

**numbers.add(Integer.toString(ele));**

**}**

**// Sort using custom comparator**

**Collections.sort(numbers, (s1, s2) -> myCompare(s1, s2) ? -1 : 1);**

**// if all numbers are zero.**

**if (numbers.get(0).equals("0")) { return "0"; }**

**StringBuilder res = new StringBuilder();**

**for (String num : numbers) { res.append(num); }**

**return res.toString();**

**}**

**}**

**Complexities :**Time Complexity : O(n \* log n)

Space Complexity : O(1)  
  
**Output :**

