public static string FreqSort(string s2)

{

try

{

Dictionary<char, int> val = new Dictionary<char, int>();

var result = new StringBuilder();

int j = 2;

//fetch every character of a string

if (s2.Length > 0)

{

foreach (char i in s2)

{

//check if key doesnt exist in the dictionary, if not add

if (!val.ContainsKey(i))

{

//if yes, add character and its count as one

val.Add(i, 1);

}

// if key character is present, just increment the value

//increment the value every time a character is repeated in the string

else

{

val[i]++;

}

}

// fetch key value pairs such that the count is sorted in descending order as per our initial requirement

// to fetch value having the maximum count

foreach (KeyValuePair<char, int> entry in val.OrderByDescending(key => key.Value))

{

//Using this condition because we want the character to be repeated as many times as the frequency

// without this condition, the output may be something like led or lde but we want it to be lled

for (int k = 0; k < entry.Value; k++)

{

//append the key value to result as per its count(value)

result.Append(entry.Key);

}

}

// return result as a string

return result.ToString();

}

else

{

return null;

}

}

catch (Exception)

{

throw;

}

}

public static int[] Intersect1(int[] nums1, int[] nums2)

{

try

{ //sort both the arrays

Array.Sort(nums1);

Array.Sort(nums2);

int j = 0;

int[] largearr;

int[] smallarr;

int len1 = nums1.Length;

int len2 = nums2.Length;

List<int> intList = new List<int>();

//check which array has larger length

if (len1 > 0 && len2 > 0)

{

if (len1 >= len2)

{

largearr = nums1.ToArray();

smallarr = nums2.ToArray();

}

else

{

largearr = nums2.ToArray();

smallarr = nums1.ToArray();

}

//traverse throw the larger array and compare each element of larger array with smaller array

//stire the common elements in the list

foreach (int i in largearr)

{

if (i == smallarr[j])

{

intList.Add(i);

j++;

}

}

int[] result = intList.ToArray();

return result;

}

else

{

return null;

}

}

catch

{

throw;

}

}

public static int[] Intersect2(int[] nums1, int[] nums2)

{

try

{

Dictionary<int, int> val = new Dictionary<int, int>();

List<int> l1 = new List<int>();

int[] largearr;

int[] smallarr;

int len1 = nums1.Length;

int len2 = nums2.Length;

int j = 2;

//Transverse through the elements of the first array

if (len1 > 0 && len2 > 0)

{

if (len1 >= len2)

{

largearr = nums1.ToArray();

smallarr = nums2.ToArray();

}

else

{

largearr = nums2.ToArray();

smallarr = nums1.ToArray();

}

foreach (int i in largearr)

{

//if the key doesnt exist then, add the key and value for the dictionary

if (!val.ContainsKey(i))

{

val.Add(i, 1);

}

//if the key is found then remove the initial count of 1

//and add the new count of 2, making sure that we increment the value of j in case the number is repeated again

else

{

val.Remove(i);

val.Add(i, j);

j = j + 1;

}

}

//transverse through the array2

foreach (int i in smallarr)

{

//every time the key is repeated, the value will be decremented and the intersection element will be added to the list

if (val.ContainsKey(i))

{

val[i]--;

l1.Add(i);

}

}

//list converted to an array considering the return type

return l1.ToArray();

}

else

{

return null;

}

}

catch

{

throw;

}

}

public static bool ContainsDuplicate(char[] arr, int k)

{

try

{

var dict = new Dictionary<char, int>();

int d = 0;

//Transverse through the array

if (arr.Length > 0)

{

for (int i = 0; i < arr.Length; i++)

{

//Check if key exists in dictionary, if it does then update the value of the dictionary such that you consider

// the difference between the current and previous i-th value which was found (basically for the same character)

if (dict.ContainsKey(arr[i]))

{

dict[arr[i]] = i - dict[arr[i]];

//save the difference in the d term

d = dict[arr[i]];

// compare the absolute difference with the target (k-th value)

if (d > k)

{

//return false if difference greater than target and break and the control goes back to for loop

return false;

break;

}

}

else

//if dictionary doesnt contain the key then add the char as key and the index value of its occurence as value

{

dict.Add(arr[i], i);

}

}

//corner case for handling arrays where is no difference or the target value is zero

if (d == 0 || k == 0)

{

Console.WriteLine("No match found");

}

// if difference is less than or equal to target, print true as per the program requirement

else if (d <= k)

return true;

}

else

{

return false;

}

}

catch (Exception)

{

throw;

}

return default;

}