# Birla Institute of Technology & Science, Pilani

## Second Semester 2017-2018, DSA (CS F211)

## Lab Assignment #4

1.Given an array of size N. Sort the array such that numbers which appeared more frequently in the array should appear before the numbers which appeared less frequently. If two numbers have the same frequency, number which appeared before in the original array (The one having smaller index) should appear before in the sorted array.

**Input:** N

N space separated integers denoting the array A

**Output:** Sorted array

**Constraints:** 1 <= N <= 10000; 1 <= A[i] <= 10000;

**Example:**

**Sample-Input:**

13

5 8 1 5 3 1 5 8 5 3 1

**Sample-Output:**

5 5 5 5 1 1 1 8 8 3 3

2. There are **N** persons playing a game. Each person owns some money which is displayed by them. And you are watching them as a spectator suddenly thought of an amount **X** and want to check if any two persons combined sum of money is equal to this amount **X** or not? You have to answer Yes or No in O(nlogn). Assume money to be an integer

**Input Format:**

N X

N integers separated by space denoting the money owned by each person

**Sample Input:**

6 10

1 5 4 19 10 6

**Sample Output:**

Yes

**Explanation:** There exists two values 4 and 6 such that 4 + 6 = 10

3. Given an array A consisting of N distinct integers A = [$a_0$,$a_1$,…….$a_{n-1}$]. You can swap any two numbers of the array any number of times. An array is considered *perfect* if the sum of $|a_i - a_{i-1}|$ for all 1 < i < N is as minimal as possible. Output the minimum number of swaps that should be performed in order to make the array *perfect*.

**Input:**

N

N space separated integers denoting the array A

**Output:**

Print the Minimum number of swaps that should be performed in order to make the array *perfect*.

**Constraints:**

1 <= n <= 100000; 1 <= A[i] <= 1000000000;

**Example:**

**Sample-Input:**

4

2 5 3 1

**Sample-Output:**

2

**Explanation:**

The array becomes *perfect* when we change it to [1 2 3 5] and in the process only two swaps were performed (2 with 5 and then 1 with 5).

4. You are provided with the time at which a student enters the class and also the time at which he leaves the class. You have to find the time at which number of students in the class is maximum.

**Input:**

N (Number of Students)

N space separated integers denoting the entry times of students

N space separated integers denoting the exit times of students

**Output:**

Print the time at which number of students in the class is maximum.

**Example:**

**Sample-Input:**

5

1 2 9 5 5

4 5 12 9 12

(First student arrives at 1 and leaves at 4,

Second student arrives at 2 and leaves at 5, and so on)

**Sample-Output:**

5

**Explanation:**

There are maximum 3 students at time 5.

5.You are given an Array A consisting of N integers. Sort the array according to the number of factors each number has (Ascending Order). If two numbers have the same number of factors

then number which appeared before in the original array (The one having smaller index) should appear before in the sorted array.

**Note: Don't use the inbuilt sorting function.**

**Input:** N

N space separated integers denoting the array A

**Output:** Sorted array

**Constraints:** 1 <= N <= 10000; 1 <= A[i] <= 10000;

**Example:**

**Sample-Input:**

7

5 11 10 20 9 16 23

**Sample-Output:**

20 16 10 9 5 11 23

**Explanation:**

Number of factors for **20** = 6, **16** = 5, **10** = 4, **9** = 3

and for **5, 11, 23** = 2 (same number of factors, hence sorted in increasing order of index)

6. Write a program to reverse the order of words in a sentence. Take the sentence given as an array of characters in the program itself. The array itself must be changed with the order of words reversed. Don't use any extra array. Assume there are no leading or trailing spaces in the sentence.

e.g. Array "Data structures and Algorithms" should become

"Algorithms and structures Data".

7. Design an efficient in-place algorithm called Partition-Even-Odd (Arr) that partitions an array **Arr** in even and odd numbers. The algorithm must terminate with **Arr** containing all its even elements preceding all its odd elements.

For example: Arr = 7, 17, 74, 21, 7, 9, 26, 10

The result might be Arr = 74, 26, 10, 21, 7, 9, 17, 7

8.Given an Array sort the array such that all numbers whose index%2 is equal to 0, be sorted in ascending order and all other numbers sorted in descending order. The resultant sorted array should contain sorted numbers whose index%2 equal to 0 followed by reverse sorted odd-placed (numbers with index%2 equal to 1) numbers.

**Note: Indexing is 0-based**

**Input:**

N

N space separated integers denoting the array

**Output:**

Sorted Array

**Sample-Input:**

9

3 1 2 4 5 9 13 14 12

**Sample-Output:**

2 3 5 12 13 14 9 4 1

**Explanation:**

Odd-place elements: 1, 4, 9, 14

Even-place elements in increasing order:

2, 3, 5, 12, 13

Odd-Place elements in decreasing order:

14, 9, 4, 1

9.Given an Integer N output the smallest palindrome larger than N

**Input:**

N

**Output:**

Smallest palindrome larger than N.

**Constraints:**

N has at most 1000000 digits.

**Sample-Input 1:**

2133

**Sample-Output 1:**

2222

**Sample-Input 2:**

808

**Sample-Output 2:**

818

10. Given an unsorted array, sort the given array by using only the following operation on array.

Reverse (arr, i): Reverse array from 0 to i.