ITCS 6114 – Algorithms and Data Structures
Programming Project – 1

The project is due before midnight on Friday, October 5. It should be submitted on canvas.

Instructions:

- You can use any Programming language to solve the first 2 questions, third question is not a programming question, you can submit third question's answer in a word document.
- Your submission should include entire source code of all the problems (including output text files).
- Submit a single zip file on canvas, do not submit 3 answers separately.

Question 1:

Given an unsorted array, Write a program that prints the sorted Array and then prints the first minimum and second minimum difference between any successive elements in its sorted form.

Your Output (sorted array, first and second minimum) should also be redirected to a text file.

Return 0 if the array contains less than 2 elements.

Example 1:

Input: [1,12,9,6,4]

Output: [1, 4,6,9,12] 2 3. (2 is first minimum difference and 3 is second minimum difference)

Explanation: The sorted form of the array is [1, 4,6,9,12]

First Minimum – [4, 6] – 2

Second Minimum – [1, 4] – 3

Example 2:

Input: [5,13,2,9,18]

Output: [2,5,9,13,18] 3 4. (3 is first minimum difference and 4 is second minimum difference)

Explanation: The sorted form of the array is [2,5,9,13,18]

First Minimum – [2, 5] – 3

Second Minimum – [5, 9] – 4

Note: If 2 or more pairs have same difference, output the pair that occurs first in the sorted array.

Output Text file should contain this information:

- Sorted Array
- First Minimum difference and the pair of elements
- Second Minimum difference and the pair of elements

Question 2:

Given an array of N integers, find the length of the longest continuous subarray so that the median of the elements in that sub array is greater than or equal to a given number x.

Example 1:

Input: arr = [3,1,7,1,4,2]., X = 3

Output: 5. (Length of Longest Subarray is 5)

Explanation:

Longest sub array is [1,2,3,4,7] of length 5

Median of longest sub array is 3 >= X

Example 2:

Input: arr = [2,1,3,6,1,1], X = 3

Output: 3. (Length of Longest Subarray is 3)

Explanation:

Longest sub array is [2,3,6] of length 3

Median of longest sub array is 3 >= X

Example 3:

Input : arr = [3,1,2,4,9], X = 2

Output: 5. (Length of Longest Subarray is 5)

Explanation:

Longest sub array is [1,2,3,4,9]

Median of longest sub array is 3 >= X

Example 4:

Input : arr = [7,1,3,5,2], X = 4

Output: 4

Explanation:

Longest sub array is [2,3,5,7]

Median of longest sub array is 4 >= X

Question 3:

You're doing some stress-testing on various models of glass jars to determine the height from which they can be dropped and still not break. The setup for this experiment, on a particular type of jar, is as follows. You have a ladder with n rungs, and you want to find the highest rung from which you can drop a copy of the jar and not have it break. We call this the *highest safe rung*.

It might be natural to try binary search: drop a jar from the middle rung, see if it breaks, and then recursively try from rung $n/4$ or $3n/4$ depending on the outcome. But this has the drawback that you could break a lot of jars in finding the answer.

If your primary goal were to conserve jars, on the other hand, you could try the following strategy. Start by dropping a jar from the first rung, then the second rung, and so forth, climbing one higher each time until the jar breaks. In this way, you only need a single jar-at the moment it breaks, you have the correct answer-but you may have to drop it n times (rather than log n as in the binary search solution).

So here is the trade-off: it seems you can perform fewer drops if you're willing to break more jars. To understand better how this trade-off works at a quantitative level, let's consider how to run this experiment given a fixed "budget" of $k \geq 1$ jars. In other words, you have to determine the correct answer-the highest safe rung-and can use at most k jars in doing so.

(a) Suppose you are given a budget of $k = 2$ jars. Describe a strategy for finding the highest safe rung that requires you to drop a jar at most $f(n)$ times, for some function $f(n)$ that grows slower than linearly. (In other words, it should be the case that $\lim_{n\to\infty} \frac{f(n)}{n} = 0$.)

(b) Now suppose you have a budget of $k > 2$ jars, for some given k. Describe a strategy for finding the highest safe rung using at most k jars. If $f_k(n)$ denotes the number of times you need to drop a jar according to your strategy, then the functions f_1, f_2, f_3,… should have the property that each grows asymptotically slower than the previous one: $\lim_{n\to\infty} \frac{f_k(n)}{f_{k-1}(n)} = 0$ for each k.