Introduction to Algorithms	**Problem Set 3 (3 problems)**
CS 4820 Fall 2019	**Due 11:59pm Tuesday, October 8**

There are three questions on this homework, a coding problem and two written questions. **No late days allowed for this homework** due to the upcoming prelim. We will post solutions as soon as the homeworks are in, and will release grades before the prelim.

Your homework submissions for written questions need to be typeset (hand-drawn figures are OK). See the course web page for suggestions on typing formulas. The solution to each written question needs to be uploaded to CMS as a separate pdf file. To help provide anonymity in your grading, do not write your name on the homework (CMS will know it's your submission).

Solutions to the coding problem need to be submitted to the **online autograder** at `https://cs4820.cs.cornell.edu/`.

Collaboration is encouraged while solving the problems, but

1. list the names of those with whom you collaborated with (as a separate file submitted on CMS);
2. you must write up the solutions in your own words;
3. you must write your own code.

Remember that when a problem asks you to design an algorithm, you must also prove the algorithm's correctness and analyze its running time. The running time must be bounded by a polynomial function of the input size.

(1) Counting Significant Inversions. (10 points) Given a list of possibly non-unique numbers $x_1, \ldots, x_n$, we say that two members of this list x_i and x_j form an inversion if $i < j$ and yet $x_i > x_j$. So if the list is sorted in increasing order then it has no inversions. In contrast, if n different numbers are listed in decreasing order then the i-th number in an decreasing list will form an inversion with every number ahead of it, so the total number of inversions will be $0 + 1 + 2 + \cdots + (n-1) = n(n-1)/2$, i.e., all pairs will form inversions. Section 5.3 of the book offers an example of this problem (on page 222), and offers a divide and conquer algorithm running in $O(n \log n)$ time that can count the number of inversions in a given sequence of n numbers in Section 5.3. In this problem, we say that a pair x_i and x_j form a *significant* inversion if $i < j$ and yet $x_i > x_j + 1$. Design a $O(n \log n)$ algorithm that finds the number of significant inversions in a given sequence, and implement the algorithm in Java. The time limit of this problem will be 1 second.

The only libraries you are allowed to `import` are the ones in `java.util.*`, and ones dealing with reading and writing inputs, such as `java.io.Reader`, `java.io.Writer`, etc. We recommend using BufferedReader and BufferedWriter for reading/writing standard input/output.

Warning: be aware that the running time of calling a method of a built-in Java class is usually not constant time, and take this into account when you think about the overall running time of your code. For instance, if you used a LinkedList, and use the `indexOf` method, this will take time linear in the number of elements in the list.

We have set up an **online autograder** at `https://cs4820.cs.cornell.edu/`. You can upload solutions as many times as you like before the homework deadline.[1] **You need to have the main method of your code named Main, must not be "public", must not be part of a package**. When you upload a submission, the autograder will automatically compile and run it on a number of public test-cases[2] that we have prepared for you, checking the result for correctness and outputting any problems that may occur. You should use this facility to verify that you have interpreted the assignment

[1]The deadline shown on the auto-grader is the latest time at which you will be allowed to submit your code, including the maximum number of slip days. If you submit your code past the actual deadline, you will automatically be charged slip days accordingly, so please be careful.

[2]They will also be available on CMS.

specification correctly. After the deadline elapses, your last submission will be tested against a new set of *private* test cases, which your grade for the assignment will be based on.

Input / output formatting and requirements

Your algorithm is to read data from `stdin` in the following format:

- The first line has a single integer, $1 \le n \le 10^5$, representing the length of the list.
- The following line contains n integers in the list, $0 \le x_1, x_2, \ldots, x_n \le 10^9$. Note that the list could contain same numbers, and it's not necessarily in increasing order or decreasing order.

Your algorithm should output data to `stdout` in the following format:

- The first line contains a single integer, the total number of significant inversions in the list $x_1, \ldots, x_n$, ending with $\backslash n$.
- Note that the output could be as large as $10^5 \times 10^5 = 10^{10}$, so be careful with the integral type you stored it. Typically, it should be `long` instead of `int`.

(2) Finding the best interval. (10 points) Suppose you have perfect foresight for the future prices of a stock, and you know the next n periods the stock price will be $x_1, x_2, \ldots, x_n$. You want to choose two dates $1 \le i \le j \le n$ to make as much money buying a stock on day i and selling it on day j. So selecting a pair $i \le j$ would make you $x_j - x_i$ in income. Unfortunately n is really huge and you are doing this on a device that is quite restricted in memory. We can find the best $i \le j$ pair by trying all pairs in $O(n^2)$ time. In this problem we are asking you to give an $O(n \log n)$ algorithm to solve this problem but **you are only allowed to use up to $O(\log n)$ additional memory**. Note that this does not include the input array's $O(n)$ memory usage. You may assume n is a power of 2.

Hint: if you do multiple recursive calls these calls can reuse memory.

(3) Perfect Hashing. (15 points) On Monday September 30 we will cover universal hashing: a randomized hash function is a function $h : U \to \mathbb{Z}_p$ selected at random, such that $\Pr[h[u] = i] = 1/p$ for all $u \in U$ and all $i \in \mathbb{Z}_p$, and also $\Pr[h[u] = h[v]] = 1/p$ for any two elements $u \ne v$. Here $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ is the set of integers mod p and U is our universe of elements we want to hash.

In class we considered storing a set $S \subset U$ of $n = |S|$ elements using this hash function (storing elements u in an array at position $h[u]$), and showed that if $n \le p$ than the expected number of elements colliding with any element u is at most 1. This problem considers extensions of this idea.

(a) (5 points) Even with the random choice of a hash function h as we have done in class, with high probability there will be some collisions. Suppose $n = p$, and let n_i be the number of elements $u \in S$ that hash to i, that is $n_i = |\{u \in S \mid h(u) = i\}|$. Show that $\Pr[\sum_{i=0}^{p-1} n_i^2 > 4p] \le 1/2$.
Hint: Consider a variable $X_{u,v}$ for any pair of elements $u, v \in S$ that is 1 if u and v collide (that is $h(u) = h(v)$). Note that $\sum_i n_i^2 = \sum_{u,v \in S} X_{u,v}$ assuming we have for any two elements $u \ne v$ we have two random variables $X_{u,v}$ and $X_{v,u}$ and we also have $X_{v,v}$ for any element $v \in S$. Do you see why? Include an explanation of this fact, if you use it.

(b) (5 points) One way to avoid collision is to use a $p \ge n^2$. Show that using universal hashing with such a large table, with probability at least 1/2 there will be no collisions at all.

(c) (5 points) Consider a dictionary that we will store a set of S elements doing a lot of lookups, but the set S will not change. For this case, it may be useful to design a hashing scheme that is perfect in the sense that there are really no collisions, and yet doesn't use as much space as the solution we have in part (b).
One idea is called 2-level hashing. Start with a random hash function with $p \ge |S|$, and repeatedly try selecting a function h until you find one that satisfies the inequality $\sum_i n_i^2 \le 4p$. This will be our level 1 table.
Now take each position i with $n_i > 1$, and use a second level hash function, h_i that has a table size $p_i \ge n_i^2$ to hash these elements to a new table. For each i repeatedly select h_i until there are no

collisions in this table. Now we have a two level table, where for elements u that don't collide we store at $h[u]$ of the first table, for those that do collide, we set $i = h[u]$ and store u at a separate table T_i at location $h_i[u]$ of table T_i. The total space used is approximately $n + \sum_i n_i^2 \leq 5n$, linear in $|S|$. The most time consuming part of finding such perfect dual hashing scheme is the many computations of $h(u)$ and $h_i(u)$ for various proposed h functions and elements $u \in S$. What is the expected number of such computations needed to find a perfect hash function? (you may assume that $n \leq p \leq 2n$ in the 1st level hash table, and $n_i^2 \leq p_i \leq 2n_i^2$ for all i for the second level hash tables for need, and that finding such a prime p and p_i is an $O(1)$ lookup in a prime table.