Introduction to Algorithms	Problem Set 2 (3 problems)
CS 4820 Fall 2019	Due 11:59pm Thursday, September 19

There are three questions on this homework, a coding problem and two written questions.

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.

Some Comments on Dynamic Programming Problems. In writing up dynamic programming algorithms for other problems all previous advice applies; but there are some additional things needed for full credit as well. If your solution consists of a dynamic programming algorithm, you must clearly specify the set of sub-problems you are using and the recurrence you are using in addition to describing what they mean in English as well as any notation you define. You must also explain why your recurrence leads to the correct solution of the sub-problems: this is the heart of a correctness proof for a dynamic programming algorithm. Finally, you should describe the complete algorithm that finds the solution value as well as the solution itself, while making use of the recurrence and sub-problems. A description of a DP algorithm consisting of a piece of pseudo-code **without these explanations will not get full credit**.

(1) Implementing Kruskal. In this problem we consider the classical Kruskal's minimum-cost spanning tree algorithm (algorithm A from the lecture on Monday, September 9th). Given n nodes V, and m possible edges E, each $e \in E$ with its cost $c_e \geq 0$. **Implement the algorithm till there are 3 components left**. The output of this algorithm will be a set of $n-3$ edges that give a graph on V with 3 components. **Your algorithm should output the sizes of the three component in non-decreasing order.**

Your algorithm needs to run in $O(m \log m)$ time. To do this, it is helpful to keep an array called componentName where componentName[v] for a node v is the name of component containing node v. Then, testing if an edge (v, w) connects nodes in different components is simply a test of whether componentName[v] $\neq$componentName[w]. The hard part is to make sure you can update the componentName array as components merge. The main idea is that when two components merge, the merged component should inherit the name from the larger of the two, this way any node v will change its component name componentName[v] at most $\log_2 n$ times, as each change increases the value by at least a factor of 2. See page claim (4.23) on 153 of the book.

Implement the algorithm in Java. 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. **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 testcases 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 specification correctly. After the deadline elapses, your 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 one integer, $1 \leq n \leq 10^5$, representing the number of nodes in the graph. Nodes are labeled with numbers $0, 1, \ldots, n-1$.
- The second line has one integer, $1 \leq m \leq 4 \times 10^5$, representing the number of edges in the graph.
- Each of the following m lines contains 3 integers, $0 \leq v_0, v_1 < n, 0 \leq w < 2 \times 10^9$, describing an edge connecting node v_0 and node v_1 with cost of w. You may assume no two edges have the same cost.

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

- The first line contains the size of smallest component s_0.
- The second line contains the size of second smallest component s_1.
- The third line contains the size of largest smallest component s_2. Please use $\backslash n$ for your line endings

For example, when the input is the following graph on 9 nodes and 15 edges with the weights next to the edges then the algorithm should select the red edges and hence result in components

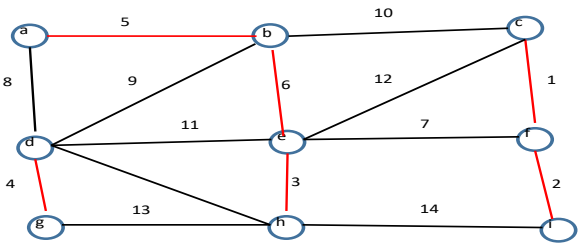

Figure 1: Example input graph for problem 2.

$\{a, b, e, h\}, \{d, g\}, \{c, f, i\}$, so that the sizes of the resulting component in sorted order are 2, 3, 4.

(2) Pairing Students. You are helping a group of students get involved in open source coding. The plan is to have pairs of students spend some time t each week coding for the project together. We will assume that you have n students, and every student prefers to work in a pair, if possible. To facilitate the process of organizing these pairs, you've asked each student i to specify a single time interval $I_i = [s_i, f_i]$ that available to them every week. We call a pair (i, j) of two students an *eligible pair* if their intervals of time overlap by at least t. Give an algorithm that selects as many eligible pairs as possible such that every paired student is part of at most one of the selected pairs, and runs in time at most $O(n^2)$.

(3) Shift Scheduling. Consider the following problem where Alice is considering shifts to work. She wants to plan a sequence of n shifts, labeled as shifts $i = 1, \ldots, n$. For each shift i, if she works that shift, she gets a known rewards r_i. However, by law (and common sense) she cannon work two consecutive periods, that is, needs to take a break between any two shifts. To illustrate the problem, consider the following example. There are 5 shifts, and the rewards in order are $8; 6; 4; 9; 2$. So Alice can work shifts 1,3,5 and earn 8+4+2, a total of 14 reward. But she can do better, if she works shifts 1 and 4 only, earning a total reward of 8+9=17.

Give a linear time algorithm that finds the maximum reward that Alice can achieve.