Algorithms
Swarthmore Honors Exam – Spring 2020

Instructions

• This exam is closed-book and closed-notes. You may not use calculators or other electronic aids.
• Pseudocode is optional if your algorithm descriptions are written very clearly.
• For full credit, when you design an algorithm, you must prove all non-obvious claims that you are making about your algorithm. Depending on your design and description, this might mean a detailed proof of correctness, or a careful analysis of running time, or both.
• You will be graded on correctness of your answers and the clarity of your expression. Be concise and neat.
• There are five problems on this exam, each worth 10 points.

Conventions

• Graphs, whether directed or not, are assumed to have no self-loops and no parallel edges.
• A path in a graph, whether directed or not, must be “simple”: it may not visit any vertex more than once.

1. A FlipSeq is a strange data structure that stores a sequence $S$ of objects and supports the following operations (and no others):
   • $S$.LENGTH( ): returns the length of the sequence stored in $S$. Takes $O(1)$ time.
   • $S$.GET($i$): returns the $i$th element of the sequence stored in $S$. Throws an exception unless $1 \leq i \leq S$.LENGTH( ). Takes $O(1)$ time.
   • $S$.FROMARRAY($A$): stores the contents of the 1-D array $A$ into $S$, replacing whatever was in $S$ previously. Takes $O(n)$ time, where $n =$ length of $A$.
   • $S$.FLIP($i$): reverses the first $i$ elements of the sequence stored in $S$: i.e., if the stored sequence is $x_1, x_2, \ldots, x_n$ before the call, then after the call it becomes
     \[ x_1, x_{i-1}, \ldots, x_1, x_{i+1}, \ldots, x_n. \]
     Throws an exception unless $1 \leq i \leq S$.LENGTH( ). Takes $O(1)$ time.

You don’t have access to the internals and have no idea how any of the above operations works.
You are given as input a FlipSeq $S$ in which is stored a sequence of $n$ elements. Each element occupies $O(1)$ words in memory. The elements come from a totally ordered universe and comparing two elements takes $O(1)$ time. Your goal is to sort the sequence stored inside $S$—i.e., cause that sequence to become ordered in non-decreasing order—by using the above operations in some way.

Design an algorithm for this task that runs in $O(n \log n)$ time and uses at most $O(1)$ words of working memory in addition to the space taken up by $S$ itself.
2. A large group of children have decided to score themselves on their combined abilities at soccer and chess relative to each other. The score of a child \( x \) is the number of children \( y \) such that \( x \) has higher soccer ability and higher chess ability than \( y \). The children would like you to help them efficiently compute everyone’s score.

Suppose there are \( n \) children. Your input is an array \( A[1 \ldots n] \) of objects, each representing a child. You can compare two children \( x \) and \( y \) by calling \( \text{COMPARESoccer}(x, y) \) and \( \text{COMPAREChess}(x, y) \) which tell you which child is better at soccer and at chess (respectively). These calls take \( O(1) \) time each. Assume that no two children are of exactly the same ability at either activity. The desired output is an array \( S[1 \ldots n] \) of scores, where \( S[i] \) is the score of the child represented by \( A[i] \).

Design an efficient algorithm for this problem that runs in \( O(n \log n) \) time.

3. Imagine that you are the CTO of a social network company. To test a feature your company is implementing, you want to select a subset of your \( n \) users so that no two selected users have the same first name and no two selected users have the same geographical location. You would, however, like to select as large a subset as possible subject to these constraints.

Design an algorithm that runs in time polynomial in \( n \) and produces such a maximum-sized subset of users.

For simplicity, assume that each name and each location has been mapped to an integer in \( \{1, 2, \ldots, n\} \) and when you access a user’s record, you can obtain the integers corresponding to their first name and their location in \( O(1) \) time.

Hint: Represent your input as a suitable graph, but think beyond one vertex per user.

4. Consider a (not necessarily complete) binary tree \( T \) with \( n \geq 2 \) leaves. You would like to connect pairs of these leaves using paths through \( T \). When you connect leaves \( x \) and \( y \) (where \( x \neq y \), of course) you earn \( a_{x,y} \) dollars. An edge can be used for at most one such path, so while you would like to earn many dollars, you are constrained by this condition that the paths must be edge-disjoint.

Design an algorithm that, given \( T \) and an \( a_{x,y} \) value for each pair \( \{x, y\} \) of leaves of \( T \), determines the maximum number of dollars you can earn. You may assume that each \( a_{x,y} \) is a positive integer and that addition/subtraction/comparison of two of these integers takes \( O(1) \) time. Your algorithm should run in \( O(n^3) \) time.

5. We are given a collection of integers \( x_1, \ldots, x_n \), where \(-2^n < x_i < 2^n \) for each \( i \). They satisfy \( x_1 + \cdots + x_n = 0 \). We would like to split this collection into many disjoint subcollections in each of which the integers add up to zero. That is, we want to find index sets \( I_1, \ldots, I_k \subseteq \{1, \ldots, n\} \) such that

- for all \( r \neq s \): \( I_r \cap I_s = \emptyset \);
- for each \( r \): \( \sum_{i \in I_r} x_i = 0 \);
- \( k \) is as large as possible.

Prove that the above problem is NP-hard.