Algorithms Qualifier Exam

Fall 2019

Instructions:

- Some of the problems have multiple parts and the later parts of the problems are more difficult than the first parts. So even if you can't complete all the parts of a problem, try at least to complete the first part.
- Provide proofs for all answers.
- Write your proofs clearly and concisely – points will be deducted otherwise.
- Pseudo-code is not required for algorithms – just a clear description with all the key ideas and proof of correctness will suffice.

Problem 1 (10 points): Local Minimum Algorithm

You are given an array $A[1 \ldots n]$ with $A[1] \geq A[2]$ and $A[n-1] \leq A[n]$. A local minimum occurs at i if $A[i-1] \geq A[i] \leq A[i+1]$. **Show how to find a local minimum in time asymptotically better than O(n).** (Note that you only need to output any one local minimum and not all of them.)

Problem 2 (10 points total): Dynamic Programming String Problem

A *palindrome* is a string whose reverse is the same string. Any string can be decomposed into a sequence of palindromes. For example the string BUBBASEESABANNA can be broken into palindromes in the following ways (and many others): BUB · BASEESAB · ANANA

B · U · BB · A · SEES · ABA · NAN · A

Problem 2a (5 points): Describe and analyze an O(n^3) time algorithm to find the smallest number of palindromes that make up a given input string.

(For example, given the input string BUBBASEESABANNA, your algorithm would return 3.)

Problem 2b (5 points): Describe and analyze an $O(n^2)$ time algorithm for this problem. (Obviously, this would give credit for 2a as well.)

Problem 3 (15 points total): Sorting By Deleting

3a (5 points): Show that the following problem can be solved in polynomial time: We have a list of numbers (throughout this problem you may assume that each number appears only once, i.e., no ties) and want to get them in ascending order. Unfortunately, we cannot move them around; the only thing we can do is delete numbers, and we try to delete as few as possible to obtain a list sorted in ascending order. For example, (a) if the list is 1,3,2, then it is optimal to delete either 3 (resulting in 1,2) or 2 (resulting in 1,3), and (b) if the list is 4,5,1,2,3,8,6,7,9, then it is optimal to delete 4,5,8. (Hint: use dynamic programming.)

Problem 3b (5 points): Prove that the problem generalized to a directed graph is NP-hard: We are given a directed graph, where the nodes are labeled with numbers. The goal is to delete as few nodes as possible so that the following property holds: if v and w have not been deleted and (v,w) is an edge, then the label of v must be smaller than the label of w. Note: Part 3a is the special case where the nodes are arranged on a line and there is an edge between every pair of nodes, pointing from left to right. (Hint: think about the VERTEX-COVER problem.)

Problem 3c (5 points): Also prove that the generalized Problem 2b can be approximately solved within a multiplicative factor of 2 in deterministic polynomial time. (Hint: again, think about the VERTEX-COVER problem.)

Problem 4 (10 points): Dynamic Data Structure Algorithm

Describe an efficient data structure that represents an ordered list of elements under the following three types of operations:

- **access(k):** Return the kth element of the list (in its current order).
- **insert(k,x):** Insert x(a new element) after the kth element in the current
- version of the list.
- **reverse(i,j):** Reverse the order of the ith through jth elements.

The list starts out empty, and each operation should run in O(log n) amortized time, where n is the (current) number of elements in the list. For example, if the current list is [a,b,c,d,e], then access(4) returns d, and after reverse(2,4), the represented list becomes [a,d,c,b,e], and then access(4) returns b. (Hints: First consider how to implement access and insert, and then think about a special case of reverse in which the [i,j] range is all elements. Use these ideas to solve problem 5.)

Problem 5 (15 points total): Offline and Online Least Common Ancestor Algorithms

The *least common ancestor* (LCA) of nodes v and w in an n node rooted tree T is the node furthest from the root that is an ancestor of both v and w.

Problem 5a (5 points): Give an O(log(n)) amortized algorithm for the offline LCA problem: The *offline LCA problem* is given a n node rooted tree T and a set of m query pairs of nodes, compute the LCA of each pair of nodes.

Problem 5b (5 points): Give an $\alpha(n)$ amortized algorithm for the offline LCA problem, where $\alpha()$ is the inverse of Ackerman's function. (Hint: Make use of a union-find data structure, which given n disjoint sets can in total time $O(m+n)\alpha(m+n))$ process m operations including the union of pairs of these sets and finding the set containing any given element.)

Problem 5c (5 points): Give an efficient algorithm for the online LCA problem: Show how to preprocess a n node tree in O(n log n) time so as to allow dynamic LCA queries to be answered in O(log n) amortized time. (Hints: Use the techniques of persistent data structures. Make use of a somewhat inefficient union-find data structure, which given n disjoint sets can in total time O(m+n)log(m+n)) process m operations including the union of pairs of these sets and finding the set containing any given element.)