CIS 675 (Fall 2018) Disclosure Sheet

Name: _____
HW # _____

Yes **No** Did you consult with anyone on parts of this assignment, including other students, TAs, or the instructor?

Yes **No** Did you consult an outside source (such as an Internet forum or a book other than the course textbook) on parts of this assignment?

If you answered **Yes** to one or more questions, please give the details here:

By submitting this sheet through my Blackboard account, I assert that the information on this sheet is true.

This disclosure sheet was based on one originally designed by Profs. Royer and Older.

Homework 2:

Due September 26 at midnight Eastern time. Submit your solutions typed and in a pdf document. To receive full credit, explain your answers.

If you collaborate with another student or use outside sources, please list those students' names and the URL/title/etc. of the sources that you referred to. Collaboration is permitted, but you must write up your own solutions.

1. You are given an array A of distinct integers, sorted in decreasing order. We want to determine if there is any index i such that $A[i] = i$ (assume the indexing starts at 0). Design an efficient divide-and-conquer algorithm to solve this. What is its running time? Explain your answer.

2. Suppose that you are given an unsorted array A of length n, in which all values are different real numbers. A *local minimum* of A is an index i such that $A[i] < A[i+1]$ and $A[i] < A[i-1]$ (if i is the first or last element of A, then it need only be less than the single adjacent element). In other words, $A[i]$ is less than all adjacent values in A.

Create an efficient algorithm for finding a local minimum in better than $O(n)$ time. Note that you don't need to return *all* local minima- just one.

(Hint: Suppose you split the array in half, and compare the two boundary elements $A[j]$ and $A[j+1]$. If $A[j] < A[j+1]$, on which side are you *guaranteed* to find a local minimum? What if $A[j+1] < A[j]$? Why?)

What is the running time of your algorithm?.

3. Consider the following sort algorithm for large arrays: HybridSort. HybridSort is a combination of MergeSort and Insertion Sort. The original MergeSort algorithm recurses until the size of each subarray is only 1. HybridSort is the same as MergeSort, except that it only recurses to a depth of at most r, where r is some fixed constant (i.e., the depth of the recursion tree is at most r). At that point, Insertion Sort is used for the resulting subproblems. For example, if after performing recursions to a depth of r, the resulting subarrays have size 5, InsertionSort is used on each of those size-5 subarrays. The sorted results are then returned to the MergeSort recursion

tree and merged as in the original MergeSort.

What is the running time of this algorithm as a function of n?

4. Prove that if G is a forest (acyclic graph) with n nodes, k edges, and c components. Prove that $c = n - k$.

5. Suppose that you are given an undirected graph, and want to find the shortest cycle in the graph (i.e., the cycle with the fewest edges). I propose the following algorithm: run DFS on the graph, and create the DFS tree. Suppose u is an ancestor of v in the tree, and I see a back edge (v, u). This means that there is a cycle containing the edges from u to v in the DFS tree and the back edge (v, u). As I am constructing this tree, keep track of these observed cycles, and at the end, output the shortest such cycle. Will this algorithm work? If yes, prove it. If no, give an example showing where it fails.

Extra Credit: You are laying optical fiber cables to bring internet to a new area of the country. The cables can only be placed between certain pairs of cities. In other words, the cities and possible cable locations can be described as an undirected graph, where each node represents a city and the edges represent the possible locations for placing cable. Each edge e is associated with a distance d_e that tells you the distance between the corresponding cities in miles.

You want to connect City S to City T through a series of cables (where S and T are specified by the user); however, each cable that you have is exactly L miles long, and you cannot connect two cables in between cities. So if there is an edge from City S to City A that is $L - 1$ miles long, and an edge from City A to City T that is $L - 2$ miles long, then you could lay cable from S to A to T. However, if the edge from S to A was $L + 1$ miles long, you would have to find a different route.

Create an algorithm to determine whether you can connect City S to City T, given the structure of the graph and your limited-length cables. Your algorithm should run in time $O(|E|)$, where $|E|$ is the number of edges. Show that your algorithm meets this running time.