The University of Melbourne
School of Computing and Information Systems
COMP90038 Algorithms and Complexity

Assignment 2, Semester 2, 2020

Released: Wednesday the 14th of October. Deadline: Sunday the 1st of November 23:59

This assignment is marked out of 30 and is worth 20% of your grade for COMP90038.

Objectives

To improve your understanding of the time complexity of algorithms. To develop problem-solving and design skills. To develop skills in analysis and formal reasoning about complex concepts. To improve written communication skills; in particular the ability to use pseudo-code and present algorithms clearly, precisely and unambiguously.

Problems

1. [4 Marks] Consider two sets of integers represented in arrays, $X = [x_1, x_2, \ldots, x_n]$ and $Y = [y_1, y_2, \ldots, y_n]$. Write two versions of a FindSetUnion(X, Y) algorithm to find the union of X and Y as an array. An element is in the union of X and Y if it appears in at least one of X and Y.

   You may make use any algorithm introduced in the lectures to help you develop your solution. That is, you do not have to write the 'standard' algorithms – just use them. Therefore, you should be able to write each algorithm in about 10 lines of code. **You must include appropriate comments in your pseudocode.**

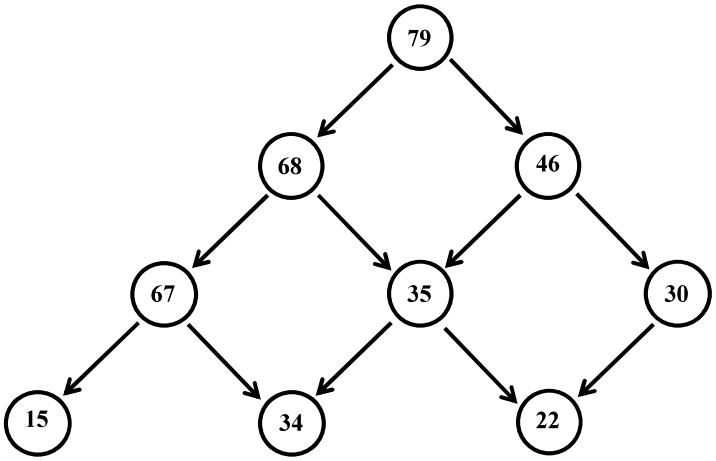
   (a) [2 Marks] Write a pre-sorting based algorithm of FindSetUnion(X, Y). Your algorithm should strictly run in $\mathcal{O}(n \log n)$.

   (b) [2 Marks] Write a Hashing based algorithm of FindSetUnion(X, Y). Your algorithm should run in $\mathcal{O}(n)$.

2. [12 Marks] A *web* is a data structure similar to a heap that can be used to implement the priority queue ADT. As with a heap, a web is defined by two properties:

   - **Structural property:** A web W consists of l levels, $l \geq 0$. The ith level, for $0 \leq i < l$, contains at most $i + 1$ entries, indicated as $W_{i,j}$ for $0 \leq j \leq i$. All levels but the last are completely filled, and the last level is left-justified.
   - **Ordering property:** Any node $W_{i,j}$ has at most two *children*: $W_{i+1,j}$ and $W_{i+1,j+1}$, if those nodes exits. The priority of a node is always greater than or equal to the priority of either child node.

   For example, the following diagram shows a web with 9 nodes and 4 levels. The arrows indicate "$\geq$" relationships.



(a) [1+2+2+2 = 7 Marks] A web W with n nodes can be stored in an array A of size n, similarly to an array-based heap. So, for example, if $n \geq 1$, the top of the web $W_{0,0}$ will be stored at $A[0]$. Give formulas for the array index that the following web entires will have, justifying how you arrived at these conclusions. Assume the n, i and j are such that all indicated web nodes actually exist.

i [1 Mark] $W_{i,j}$

ii [2 Marks] Left and right children of $W_{i,j}$

iii [2 Marks] Left and right parents of $W_{i,j}$

iv [2 Marks] Left and right children of the node corresponding to $A[k]$

(b) [2 Marks] Give upper and lower bounds for the number of nodes n in a web W with l levels, where $l \geq 1$. For example, a web with 2 levels has at least 2 and at most 3 nodes. Make your bounds as tight as possible. Briefly justify your answer.

(c) [3 marks] Briefly describe an efficient algorithm to eject the maximal element from the web W that is stored in an array A of size n. Find the complexity of this algorithm and justify your conclusion Note: you do not have to write this algorithm in pseudocode. We are expecting that you write a short paragraph or a short list of bullet points describing the important steps of the algorithm and explaining the time complexity.

3. [14 Marks] Researchers from the School of BioSciences have requested our help with one of their experiments. They are performing behavioural experiments with zebrafish. At any one instance in time there are a large number of zebrafish in the aquarium. For their particular experiment, the biologist take a snapshot of the aquarium and then need to find the longest series of zebrafish such the length of each fish along the horizontal direction in the aquarium is increasing. They also need to know the number of zebra fish in this series.

   For example, the snapshot of the aquarium resulted in fish lengths of $[2, 5, 3, 7, 11, 1, 12, 4, 15, 14, 6, 16]$. One possible longest series of increasing lengths in this case is $[2, 3, 7, 11, 12, 14, 16]$ with 7 zebrafish. We say one possible longest series of increasing lengths here because it is not necessarily unique. For example, the length 14 in the output could be replaced with 15: $[2, 3, 7, 11, 12, 15, 16]$ and also be valid.

   In this question you will consider algorithms for finding the longest series of increasing lengths via the function LongestIncreasingLengths($A[0, \cdots, n-1]$), as well as the size of this output array.

(a) [1+2+1 = 4 Marks] Consider a recursive algorithm:

i [1 Mark] Write down a recurrence relation for the function LongestIncreasingLengths.

ii [2 Marks] Using this recurrence relation, write a recursive algorithm in pseudocode for LongestIncreasingLengths that only calculates the array size of the longest series of increasing lengths. You do not need to output the actual array containing the longest series of increasing lengths in this part of the question. For the example above with input $A = [2, 5, 3, 7, 11, 1, 12, 4, 15, 14, 6, 16]$, the output should just be 7. The pseudocode should be about 10 lines of code.

iii [1 Mark] What is the time complexity of this recursive algorithm? Justify your answer.

(b) [5+1+1 = 7 Marks]

i [5 Marks] Building on from your recursive algorithm in part (a), write down a dynamic programming implementation in pseudocode for the function LongestIncreasingLengths($A[0, \cdots, n-1]$) to find the longest series of increasing lengths. This should also output the size of the longest series of increasing lengths. The pseudocode should be about 20 lines of code.

ii [1 Mark] Explain how the recurrence relation used for your dynamic programming implementation involves overlapping instances.

iii [1 Mark] What is the time complexity of your algorithm and how much auxiliary space was required. Justify your answer.

(c) [1+2 = 3 Marks] The time complexity of the recursive algorithm for LongestIncreasingLengths was exponential, while the dynamic programming algorithm lead to a polynomial time complexity (note, you need to determine that polynomial above). Here we will investigate an algorithm for the function LongestIncreasingLengths that has a time complexity of $\mathcal{O}(n \log n)$.

Consider building a set of arrays for the input array $A[0, \cdots, n-1]$. As we scan along A, we will compare $A[i]$ with the final element in each array in this set. This comparison will satisfy the following conditions:

(1) If $A[i]$ is smaller than the final element in each array, start a new array of size 1 with $A[i]$.

(2) If $A[i]$ is larger than the final element in each array, copy the longest array and append $A[i]$ to this new array.

(3) If $A[i]$ is in between, find the array with the final element that is greater than $A[i]$ and replace that element with $A[i]$.

i [1 Mark] Write down the set of arrays that satisfy these rules for the input array $A = [0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]$.

ii [2 Marks] Building from these conditions, explain how an algorithm for the function LongestIncreasingLengths could run with time complexity $\mathcal{O}(n \log n)$. You may make use any algorithm introduced in the lectures to help you with your explanation. Note: you do not have to write this algorithm in pseudocode. We are expecting that you write a short paragraph or a short list of bullet points describing the important steps of the algorithm to explain the time complexity.

Hint: what if you only consider the final elements of this set of arrays as a single array?