University of Waterloo
CS240E, Winter 2022
Written Assignment 1

Due Date: Wednesday, January 19, 2022 at 5pm

Be sure to read the assignment guidelines (`http://www.student.cs.uwaterloo.ca/~cs240e/w22/guidelines.pdf`). Submit your solutions electronically as a PDF with file name a1sol.pdf using MarkUs. We will also accept individual question files named a1q1.pdf, a1q2.pdf, ... if you wish to submit questions as you complete them.

Question 0 Academic Integrity Declaration

Read, sign and submit A01-AID.txt now or as soon as possible. Failure to do so will result in 0 marks on the entire assignment.

Question 1 [5 marks]

There are two different definitions of 'little-omega' in the literature (to distinguish them, we will call them ω_0 and ω_1 here). Fix two functions $f(x), g(x)$. We say that

- $f(x) \in \omega_0(g(x))$ if for all $c > 0$ there exists $n_0 > 0$ such that $|f(x)| \geq c|g(x)|$ for all $x \geq n_0$, and
- $f(x) \in \omega_1(g(x))$ if $g(x) \in o(f(x))$.

Show that these two definitions are equivalent, i.e., $f(x) \in \omega_0(g(x))$ if and only if $f(x) \in \omega_1(g(x))$. Your proof must be from first principle, i.e., directly using the definitions (do not use the limit-rule). Note that $f(x), g(x)$ are not necessarily positive.

Question 2 [3+3+8=14 marks]

Consider the following (rather strange) code-fragment:

Algorithm 1: mystery (int n)

Input: $n \geq 2$
$L \leftarrow \lfloor \log(\log(n)) \rfloor$
print all subsets of $\{1, \ldots, 2^L\}$

For example, for $n = 17$, we have $\log 17 \approx 4.08$ and $\log(4.08) \approx 2.02$, so $\log\log(17) \approx 2.02$ and $L = 2$ (and we print the 16 subsets of $\{1, \ldots, 4\}$). This question is really asking about the run-time of `mystery`, but to avoid having to deal with constants, define $f(n)$ to be the number of subsets that we are printing when calling `mystery` with parameter n.

(a) Show that $f(n) \in O(n)$.

(b) Show that $f(n) \in \Omega(\sqrt{n})$.

(c) Prof. Conn Fused thinks that $f(n) \in \Theta(n^d)$ for some constant d. (By the previous two parts, necessarily $\frac{1}{2} \leq d \leq 1$.) Show that Prof. Fused is wrong, or in other words, for any $\frac{1}{2} \leq d \leq 1$ we have $f(n) \notin \Theta(n^d)$.

Question 3 [2+6+4=12 marks]

To reduce the height of the heap one could use a d-way heap. This is a tree where each node contains up to d children, all except the bottommost level are completely filled, and the bottommost level is filled from the left. It also satisfies that the key at a parent is no smaller than the keys at all its children.

a) Explain how to store a d-way heap in an array A of size $O(n)$ such that the root is at $A[0]$. Also state how you find parents and children of the node stored at $A[i]$. You need not justify your answer.

b) What is the height of a d-ary heap on n nodes? Give a tight asymptotic bound that depends on d and n. You may assume that n and d are sufficiently big (e.g. $d \geq 3$ and $n \geq 10$). Note that d is not necessarily a constant.

c) Assume that $n \geq 4$ is a perfect square. What is the height of a d-ary heap for $d = \sqrt{n}$? Give an exact bound (i.e., not asymptotic).

Question 4 [9 marks]

Consider a (max-oriented) meldable heap H that holds n integers. Describe an algorithm that is given H and an integer x, and that finds all items in H for which the priority is at least x. (Note that x may or may not be in H.) Your algorithm should have $O(1 + s)$ worst-case run-time, where s is the number of items that were found.

Question 5 [3+7(+5)=10(+5) marks]

How would you implement *increaseKey*(v, k) in a binomial heap? The method is given two parameters: a node v and the new value k that its key should have.

a) Prof. Dodo thinks he can implement this using *fix-up* as shown in Algorithm 2).

Show that Prof. Dodo is incorrect. Thus, give an example of a flagged tree that satisfies the binomial-heap-order property, indicate a node v and a key $k > v.key$, and show that calling *increaseKey*(v, k) with the code in Algorithm 2 would result in a flagged tree that does not satisfy the binomial-heap-order property. (Try to keep your tree small, no more than 16 nodes.)

Algorithm 2: *increaseKey*(v, k)

if $(k > v.key())$ **then**
    $v.key \leftarrow k$
    `// perform fix-up`
    while $p \leftarrow v.parent$ *is not NIL and* $p.key < v.key$ **do**
        swap key-value pairs of v and p
        $v \leftarrow p$

b) Give an implementation of *increaseKey* in a binomial heap that has worst-case run-time $O(\log n)$.

c) (Bonus) Give an implementation of *decreaseKey* in a binomial heap, and state a tight run-time bound. Make your implementation as efficient as you can (the amount of bonus-marks given will depend on the asymptotic bound that you achieve).