University of Waterloo
CS240E, Spring 2026
Written Assignment 1

Due Date: Tuesday, May 26, 2026 at 5:00pm

Be sure to read the assignment guideliness (`https://student.cs.uwaterloo.ca/~cs240e/s25/assignments.phtml#guidelines`). Submit your solutions electronically to Crowdmark. Ensure you have read, signed, and submitted the **Academic Integrity Declaration** AID01.

Grace period: submissions made before 7:59pm on May 26 will be accepted without penalty. Please note that submissions made after 7:59pm **will not be graded** and may only be reviewed for feedback.

Question 1 [9 marks]

There are many different definitions of "little-omega" in the literature (to distinguish them, we will call them $\omega_1, \ldots, \omega_3$ here). Fix two functions $f(x), g(x)$ from $\mathbb{R}^+$ to $\mathbb{R}^+$; in particular they are never 0. We will say that

(i) $f(x) \in \omega_2(g(x))$ if for all $c > 0$ there exists $n_0 > 0$ such that $f(x) \geq c \cdot g(x)$ for all $x \geq n_0$,

(ii) $f(x) \in \omega_1(g(x))$ if for all $c > 0$ there exists $n_0 > 0$ such that $f(x) > c \cdot g(x)$ for all $x \geq n_0$,

(iii) $f(x) \in \omega_3(g(x))$ if $g(x) \in o(f(x))$.

Show that these definitions are equivalent, i.e., $f(x) \in \omega_i(g(x))$ if and only if $f(x) \in \omega_j(g(x))$ for any i, j. Your proof may use the limit-rule (and related statements) only as far as their actual proofs are in the course notes; otherwise you need to re-prove the statement (but you may copy and modify proofs from the course notes).

Recall that the easiest way to prove that a number of statements are equivalent is to prove a circle of implications among them, e.g. (i)⇒(ii)⇒(iii)⇒(i). Picking the circle to prove is up to you, but state clearly what you are proving.

Question 2 [3+3+3=9 marks]

Consider the following (rather strange) code-fragment:

Algorithm 1: mystery (int n)

Input: $n \geq 2$

1 $L \leftarrow \lfloor \log(\log(n)) \rfloor$

2 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 [7 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 4 [3+7+3=13 marks]

How would you implement *increase-key*(z, k) in a binomial heap? The method is given as parameter a node z and a key k and it should increase the key of z to k if it was smaller before.

a) Prof. B. Fuddled thinks that they can implement this using *fix-up* as follows:

Algorithm 2: *increase-key*(z, k)

```
1 if (k > z.key()) then
2     z.key ← k
3     while p ← z.parent is not NULL and
        p.key < z.key do        // do fix-up
4         swap key-value pairs of z and p
5         z ← p
```

Show that Prof. Fuddled is incorrect. Thus, give an example of a flagged tree that satisfies the binomial-heap-order property, indicate a node z and a key $k > z.key$, and show that calling *increase-key(z,k)* results in a flagged tree that does not satisfy the binomial-heap-order property. (Try to keep your tree small, no more than 16 nodes.)

b) Give a method to implement *increase-key* in a flagged tree with the binomial-heap-order property, with worst-case run-time $O(\log n)$.

c) Recall that *decrease-key*(z, k) is given a node z and a key k and should decrease the key of z to k if it was bigger before. Show that this operation can be reduced to the

other operations. Specifically, show that if a priority queue implementation supports *size*, *find-max*, *delete-max*, *increase-key* and *insert* with $O(f(n))$ run-time, then you can also *decrease-key* with $O(f(n))$ run-time.