University of Waterloo
CS240E, Winter 2025
Written Assignment 1

Due Date: Tuesday, January 21, 2025 at 5pm

Be sure to read the assignment guideliness (`https://student.cs.uwaterloo.ca/~cs240e/w25/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 11:59PM on Jan. 21 will be accepted without penalty. Please note that submissions made after 11:59PM **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 $\omega_1, \dots, \omega_3$ here). Fix two functions $f(x), g(x)$ from $\mathbb{R}^+$ to $\mathbb{R}^+$; in particular they are never 0. We say that

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

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

(iii) $f(x) \in \omega_3(g(x))$ if the function $\frac{f(x)}{g(x)}$ tends to infinity.

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+6=9 marks]

Motvation: In class we often use a sloppy recursion, or assume that "n is divisible as needed". The following question illustrates that, with some limitations, this approach is justified.

a) Show that the following statement is **true**.

"Let $f(x)$ be a positive monotone function with domain $\mathbb{R}^+$. Assume that $f(x) \leq x$ whenever x is a power of 2, i.e., $x = 2^k$ for some integer $k \geq 0$. Then $f(x) \in O(x)$."

b) Show that the following statement is **false**.

"Let $f(x)$ be a positive monotone function with domain $\mathbb{R}^+$. Assume that $f(x) \le x$ for infinitely many integers, i.e., for any N there exists an integer $x \ge N$ with $f(x) \le x$. Then $f(x) \in O(x)$."

Reminder: To show that a statement is false, you need to give an example that satisfies all assumptions of the statement, but does not satisfy the conclusion.

Question 3 [5 marks]

Conside the following (strange) code-fragment. Let $f(n)$ be the number of times that *mystery* reaches the print-statement when called with parameter n.

Algorithm 1: *mystery*(int n)

```
for j ← ⌊(n−2)/2⌋ down to 0 do
    i ← j
    while 2i + 1 ≤ n − 1 do
        print "*"
        i ← 2i + 1
```

Give an asymptotically tight bound on $f(n)$. Justify your answer.

Question 4 [2+3+4=9 marks]

a) Let T be a meldable heap and let z be a node of T. Show how to implement a routine *remove-node*(z) which removes the node z from the meldable heap. The run-time should be $O(\log n)$ expected time. You may assume that the heap has parent-references.

b) Let T be a meldable heap and let z be a node of T. Let h be the height of the sub-heap rooted at z. Show how to implement *remove-node*(z) in $O(h)$ **worst-case** time.

c) Let T_1 and T_2 be two meldable heaps of height h_1 and h_2. If we merge T_1 and T_2 as explained in class, the resulting heap may well have height $h_1 + h_2$. (You need not show this.)

Give an algorithm that merges T_1 and T_2 into a meldable heap T that has height at most $\max\{h_1, h_2\} + 1$. Your algorithm should have **worst-case** run-time $O(\max\{h_1, h_2\})$.

Question 5 [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)

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

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 realization supports *size*, *find-max*, *delete-max*, *increase-key* and *insert* with $O(f(n))$ run-time, then you can also realize *decrease-key* with $O(f(n))$ run-time.