University of Waterloo
CS240 - Fall 2025
Assignment 2

Due Date: Tuesday September 30 at 5:00pm

Please follow the guidelines for submission on the course webpage.

`https://student.cs.uwaterloo.ca/~cs240/f25/assignments.phtml#guidelines`

Each question must be submitted individually to Crowdmark. Submit early and often.

Grace period: submissions made before 7:59PM on September 30 will be accepted without penalty. Your last submission will be graded. Please note that submissions made after 7:59PM **will not be graded** and may only be reviewed for feedback.

There are 51 possible marks available. The assignment will be marked out of 49.

Problem 1 [5+5+5=15 marks]

For this question, it is sufficient to provide the drawings requested. For all parts, assume a max-oriented version of binary heap.

a) Starting with an empty heap, construct the heap resulting from insertion of 31, 40, 58, 34, 25, 43, 10. Show the heap, drawn as a binary tree, after the insertion of 40, after insertion of 34, and after insertion of 10.

b) Let $A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix}$. Draw the heap, in binary tree form, after A is heapified in-place using fix-downs according to the recipe in Module 2.

c) Consider a heap of size $n > 15$ that is stored as an array A and contains the priorities $A = \begin{bmatrix} -1 & -2 & -3 & \dots & -n \end{bmatrix}$. Perform two `deleteMax` operations, and show the first three levels of the resulting heap, drawn as a binary tree, after each operation.

Problem 2 [10 marks]

Let $A[0 \dots n-1]$ be a random permutation of the the first n non-negative integers. Let $P(n)$ be the probability that A is a max-heap, that is, that the heap-order property is satisfied. Derive the value of $P(n)$ for each of the first eight sizes $n = 1, 2, 3, \dots, 8$.

Problem 3 [10 marks]

A sorting algorithm is said to sort *in-place* if only a constant number of elements of the input are ever stored outside the array. Suppose you are given an array $A[0 \dots n-1]$, of (key, element) pairs, with each integer key in the range $0 \ldots n-1$ occurring exactly once, and with no way to know the value of each element. Allowing non-comparison-based algorithms, give an $O(n)$ in-place algorithm to sort A in ascending order of its keys. Analyze the running time of your method.

Problem 4 [4+6=10 marks]

A *deterministic* algorithm is one whose execution depends only on the input. By contrast, the execution of a *randomized* algorithm depends also on some randomly-chosen numbers. A *Las Vegas* randomized algorithm always produces the correct answer, but has a running time which depends on the random numbers chosen (randomized quick-select and quick-sort are of this type). Informally, such algorithms are always correct, and probably fast. A *Monte Carlo* randomized algorithm has running time independent of the random numbers chosen, but may produce an incorrect answer. Informally, such algorithms are always fast, and probably correct.
Given an array A of length n, an element x is said to be *dominant* in A if x occurs at least $\lfloor n/2 \rfloor + 1$ times in A. That is, copies of x occupy more than half of the array.

a) Given an array A that contains a dominant element, describe an in-place Monte Carlo randomized algorithm to find the dominant element. Show that your algorithm has running time in $O(1)$ and returns the correct answer with probability at least 1/2.

b) Given an array A that contains a dominant element, describe an in-place Las Vegas randomized algorithm to find the dominant element. Show that your algorithm always returns the correct answer, and has expected running time in $O(n)$.

Problem 5 [6 marks]

You are given an array $A[0 \ldots n-1]$ of integers (not necessarily distinct) that forms a max-heap of size n. Describe an algorithm that takes as input an integer c, not necessarily in the heap, and reports all integers in the heap that are greater than or equal to c. The running time of your algorithm should be in $O(1+k)$, where k is the number of integers reported. Provide a brief explanation for why the running time of your algorithm is in $O(1+k)$.