University of Waterloo
CS240 Spring 2022
Assignment 2

Due Date: Wednesday, June 1 at 5:00pm

The integrity of the grade you receive in this course is very important to you and the University of Waterloo. As part of every assessment in this course you must read and sign an Academic Integrity Declaration before you start working on the assessment and submit it **before the deadline of June 1** along with your answers to the assignment; i.e. **read, sign and submit A02-AID.txt now or as soon as possible**. The agreement will indicate what you must do to ensure the integrity of your grade. If you are having difficulties with the assignment, course staff are there to help (provided it isn't last minute).

The Academic Integrity Declaration must be signed and submitted on time or the assessment will not be marked.

Please read `https://student.cs.uwaterloo.ca/~cs240/s22/assignments.phtml#guidelines` for guidelines on submission. **Each question must be submitted individually to MarkUs as a PDF** with the corresponding file names: a2q1.pdf, a2q2.pdf, ... , a2q4.pdf . It is a good idea to submit questions as you go so you aren't trying to create several PDF files at the last minute.

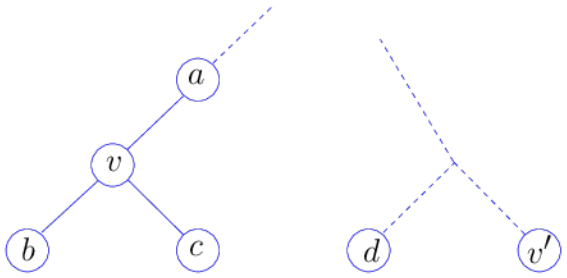
Late Policy: Assignments are due at 5:00pm on Wednesday. Students are allowed to submit **one** late assignment, 2 days after the due date on Friday by 5:00pm. Assignments submitted after Friday at 5:00pm or Wednesday at 5:00pm (if you have already used your one late submission) will not be accepted for grading but may be reviewed (by request) for feedback purposes only.

Question 1 [4 marks]

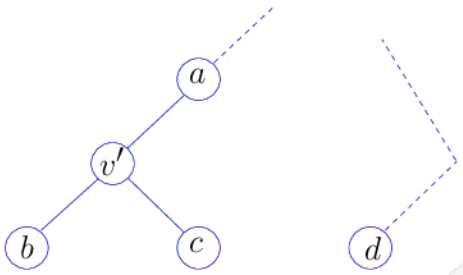
Suppose we are working with a heap, represented as an array, and we want to remove an element from it that is not necessarily the root. We are given the index of this element in the array. Describe an algorithm that performs this task and analyse its complexity (since we did not prove correctness of bubble up / bubble down in class, we do not require you to prove correctness).

Solution:

Swap the keys v (at node i) and v' (at the last node in the heap) and delete the last node. We go from



to



Then:

- if $a \geq v', v' \geq b, v' \geq c$ do nothing
- if $v' > a$, bubble up from i.
- if $a \geq v'$ but either $v' < c$ or $v' < b$, bubble down from i

The cost is $O(\log n)$.

Question 2 [8 marks]

We want to prove the following: there is no comparison-based algorithm that can merge m sorted arrays of length m into a unique sorted array of length m^2 doing $O(m^2)$ comparisons. We argue by contradiction, and we assume that it is possible, so that we have such an algorithm (which we call FastMerge).

Modify MergeSort in order to use FastMerge, and derive a contradiction. You may use the following property: if a function $T(n)$ satisfies $T(n) = \sqrt{n}T(\sqrt{n}) + O(n)$, then $T(n) = O(n \log(\log(n)))$. Do not worry about n being a perfect square or not.

Solution: Consider the algorithm FastMergeSort defined as follows: on input an array of length n, write $m = \sqrt{n}$, and call FastMergeSort recursively on $A[0..m-1], A[m..2m-1], \ldots A[n-m..n-1]$ (m arrays of length m). Then merge them using FastMerge.

The runtime $T(n)$ satisfies

$$T(n) = \sqrt{n}T(\sqrt{n}) + \Theta(n)$$

So $T(n) \in O(n \log \log n)$.

But $n \log \log n \in o(n \log n)$ (use the limit test to verify), a contradiction with the lower bound $\Omega(n \log n)$ for comparison-based sorting algorithms.

Question 3 [8 marks]

Given an array $A[0 \ldots n-1]$ of numbers, show that if $A[i] \geq A[i-j]$ for all $j \geq \log(n)$, the array can be sorted in $O(n \log(\log(n)))$ time.

Hint: Partition A into contiguous blocks of size $\log(n)$; i.e. the first $\log(n)$ elements are in the first block, the next $\log(n)$ elements are in the second block, and so on. Then, establish a connection between the elements within two blocks that are separated by another block.

Split the array into $m := \frac{n}{\log n}$ arrays of length $\log n$:

$$A = [A_1|A_2|A_3|\cdots|A_m]$$

By assumption, all entries of A_1 are less than or equal to all entries of A_3, which are less than or equal to A_5, ... and similarly with A_2, A_4, ...

Sort all $A_i's$. This takes time $O(\frac{n}{\log n} \log n \log\log n) = O(n \log\log n)$ using MergeSort. Then the two sub-arrays $[A_1|A_3|A_5|\cdots]$and $[A_2|A_4|A_6|\cdots]$ are both sorted. So it suffices to merge them, which we can do in time $O(n)$. The total time is $O(n \log\log n)$.

Note: There are many other valid solutions.

Question 4 [3+3+4 marks]

The *median* of a sequence $(a_1, \ldots, a_n)$ of integers is defined as follows: assume we sort these integers, and write them as $(a'_1, \ldots, a'_n)$ once sorted. Then their median is $a'_{\lceil n/2 \rceil}$. For instance:

- if $n = 1$, the median of (2) is 2
- if $n = 2$, the median of $(5, 1)$ is 1
- if $n = 3$, the median of $(2, 10, 1)$ is 2
- if $n = 4$, the median of $(5, 5, 2, 1)$ is 2, etc

Even though we sort the sequence to *define* the median, it is possible to avoid using any sorting algorithm to *compute* it: for instance, quickselect finds the median of a sequence of length n in average time $O(n)$.

In this problem, we study an *online* algorithm for finding the median of a sequence: we suppose that we receive the entries of the sequence one at a time, and we want to print the medians of all these partial sequences as we go. For instance:

- suppose we first receive **15**. We print **15**, which is the median of the sequence (15)
- next, we receive **10**. We print **10**, which is the median of the sequence $(15, 10)$
- next, we receive **1**. We print **10**, which is the median of the sequence $(15, 10, 1)$
- next, we receive **20**. We print **10**, which is the median of the sequence $(15, 10, 1, 20)$

- next, we receive **30**. We print **15**, which is the median of the sequence $(15, 10, 1, 20, 30)$

Re-computing the median from scratch every time would be too slow. Here is an idea for a better algorithm: use two heaps H_{lo} and H_{hi}, each of which will roughly contain half of the elements seen so far: if we have seen n elements, H_{lo} should contain the $\lceil n/2 \rceil$ smallest elements, H_{hi} should contain the $\lfloor n/2 \rfloor$ largest ones.

1. On the example $(15, 10, \dots)$ above, show us what these heaps would contain at each of the 5 steps (we don't know if these are min-heaps or max-heaps yet, so just tell us what elements they contain).

2. We would like to be able to read off the (current) median using just one access to H_{lo}. What kind of heap should it be, a min-heap or a max-heap? How long does finding the current median take?

3. Describe how to update the two heaps when inserting the next element. In particular, in which heap do you insert the element, and how do you ensure that H_{lo} and H_{hi} have the required size afterwards? Give the runtime of your update method, with a short justification; it should be $o(n)$. (At this stage, you will have to explain whether H_{hi} should be a min-heap or a max-heap.)

   1)

   $\{15\}, \emptyset$

   $\{10\}, \{15\}$

   $\{1, 10\}, \{15\}$

   $\{1, 10\}, \{15, 20\}$

   $\{1, 10, 15\}, \{20, 30\}$

   2) H_lo should be a max heap. Then, its max element is precisely the median, and we get it in time $O(1)$.
   3) We use a min-heap for H_{hi}. To insert x, we do a case discussion:

   - Suppose $|H_{\text{hi}}| = |H_{\text{lo}}|$. Compare x to the current median m.
     If $x \leq m$, insert x in H_{lo} (this takes time $O(\log n)$)
     Else, remove the min from H_{hi}, insert it in H_{lo}, and insert x in H_{hi} (time $O(\log n)$).
   - Suppose $|H_{\text{hi}}| < |H_{\text{lo}}|$. Compare x to the current median m.
     If $x > m$, insert x in H_{hi} (time $O(\log n)$)
     Else, remove m from H_{lo}, insert it in H_{hi} and insert x in H_{lo} (time $O(\log n)$)