National University of Singapore

School of Computing

CS2040C - Data Structures and Algorithms

Midterm Quiz @ LT15+LT18

(Wed, 28 Sep 2022, S1 AY2022/23, 60m)

INSTRUCTIONS TO CANDIDATES:

1. You can start immediately after you are given the password to open this file.
2. This assessment paper contains FIVE (5) sections.
   It comprises EIGHT (8) printed pages, including this page.
   The last page is an empty page.
3. This is an **Open Book Assessment**.
   Additionally, you can also use your laptop too (but in airplane mode).
4. Answer **ALL** questions within the **boxed space** of the answer sheet (page 5-8).
   You will only need to hand over page 5-8 after this quiz.
   You can use either pen or pencil. Just make sure that you write **legibly**!

A Worst Case Time Complexity Analysis ($10 \times 4 = 40$ marks)

Write down the *tightest*[1] *worst case* time complexity of the various data structure operations or algorithms below. Each correct answer worth 4 marks and the grading scheme is very strict (4 vs 0, except if you also write an optional supplementary explanation).

The operations (algorithms) referred below are the **unmodified version**, as per discussion in class, e.g. as currently explained in VisuAlgo or as currently implemented in C++ STL. Unless otherwise mentioned, there are currently n (not necessarily distinct) 64-bit signed integers, i.e., `long long` ($100\,000 \leq n \leq 200\,000$) in the data structure: `std::vector v` (a resize-able array), `std::list l` (a DLL), `std::stack s` (LIFO), `std::queue q` (FIFO), `std::deque dq` (not really a DLL), or `std::priority_queue pq` (a Binary Max Heap).

[1] What we meant by tightest worst case time complexity is as follows: If an operation of the stipulated data structure/an algorithm needs at best $O(n^3)$ if given the worst possible input but you answer *higher* time complexities than that, e.g. $O(n^4)$ – which technically also upperbounds $O(n^3)$, you will get wrong answer for this question. Obviously, any other answer that is *'better'* than $O(n^3)$ for this specific example is also wrong.

No	Operations
1	Insert a new integer x at index $\frac{3*n}{4}$ of an unsorted vector `v` (important: you do **not** need to retain the order of integers in `v`)
2	Search the **last** index of a value `x` in a **sorted** vector `v` (important: `v` may contain duplicates)
3	Sort vector `v`, but this time v contains only 0 or 1
4	Check if there are at least 3 **different** integers in a list `l`
5	Finding the **smallest** integer which appears **the most** in a stack `s`
6	Reversing a queue `q`
7	Finding the median of a **sorted** deque `dq`
8	Extracting **only** the third largest element in priority queue `pq`
9	Call `pq.top()` of priority queue `pq` for $\frac{n}{5}$ times
10	Inserting an **initially empty** priority queue `pq` with n, n-1, n-2, ..., down to 1 one-by-one in that order

B A Sorting-Related Problem (30 marks)

In class, we have discussed various sorting problems and their performances on various input types. One particular input type is of interest to Steven recently: the 'nearly sorted' inputs. This time, he is interested with the following question:

Given an array A of n **distinct** 64-bit signed integers that is **not sorted**, what is the minimum number of swaps (*not necessarily swapping adjacent elements*) that you need to make A sorted in ascending order? For example, if Array B contains $n = 5$ integers: $\{2, 1, 7, 4, 3\}$, then the answer is at minimum 2 swaps: swap 2 and 1, then swap 7 and 3 (or the other way around). We cannot sort array B in just 1 swap so 2 is the optimal answer.

B.1 Manual Test Cases ($4 \times 2 = 8$ marks)

You are given four small test cases below. For each test case, write down the answer (1 mark) and the swap actions (the other 1 mark) needed.

1. Array C contains 7 integers: $\{77, 2, 3, 4, 5, 6, 1\}$.
2. Array D contains 4 integers: $\{2, 3, 7, 1\}$.
3. Array E contains 4 integers: $\{2, 7, 1, 3\}$.
4. Array F contains 7 integers: $\{2, 4, 5, 1, 3, 6, 7\}$.

B.2 Find the Upperbound (4 marks)

What is the largest possible answer (the largest minimum number of swaps given any input array). What input array causes the minimum number of swaps to be the largest possible? Explain!

B.3 Create Test Cases ($2 \times 3 = 6$ marks)

1. Create a test case (1 mark) (to simplify grading, use a permutation of n integers of 1 to n) so that it has the largest possible minimum swaps to sort it, yet the Optimized Bubble Sort algorithm can still terminate in $O(n)$. Explain! (2 marks)

2. Create a test case (1 mark) (to simplify grading, use a permutation of n integers of 1 to n too) so that it has the largest possible minimum swaps to sort it and also causes the Optimized Bubble Sort algorithm to still run in $O(n^2)$. Explain! (2 marks)

B.4 Propose an $O(n^2)$ Algorithm (5 marks)

There is a simple $O(n^2)$ algorithm that can be used to compute the required answer (it will work fast (1s) for medium range of n, e.g., $1 \leq n \leq 10\,000$). Explain what are the required modification(s) that you will need to do to achieve this. Hint: It is related to one of the sorting algorithm(s) that we have discussed in class. PS: If you can do the next section, i.e., the $O(n \log n)$ solution, you can leave this section blank and you will still get all the marks.

B.5 Propose an $O(n \log n)$ Algorithm (7 marks)

There is a faster $O(n \log n)$ algorithm that can be used to compute the required answer (it will work fast (1s) for large range of n, e.g., $1 \leq n \leq 200\,000$). Analyze your proposed algorithm carefully. Note that the grading scheme for this section is 0 or 7 marks, i.e., attempt the previous section first.

C PS3B - congaline, revisited (5 marks)

For a 'free' 5 marks, re-describe in pseudo-code on how to fully solve PS3B - congaline, i.e., describe how to store the congaline information, how to 'quickly identify the location of someone's partner', and how to implement those 5 operations F/B/R/C/P all in $O(1)$. Note that the grading scheme for this section has a special override. Your score will be 0 for this section if on random spot check, what you wrote in this section totally does not tally with what your code that gets 100/100 in PS3B actually does (and there will be additional manual follow-up discussion). This check will not be done to anyone else who did not score 100/100 in PS3B, i.e., you can answer the 'better version' than what your code did back in PS3.

D A New Priority Queue Operation (20 marks)

In class, we have learned Binary (Max) Heap data structure that can be used to efficiently (in $O(\log n)$) supports two crucial Priority Queue (PQ) operations: `insert(v)` and `extractMax()`. Let's assume

that our Binary (Max) Heap contains only 64-bit signed integers and there can be up to $0 \leq n \leq 200\,000$ integers inside our data structure.

This time, you have one interesting new operation: `setLimit(l)` that is supposed to modify the behavior of subsequent `insert(v)` and `extractMax()` operations as follows:

- `insert(v)` will reject the insertion of v if $v \geq l$ (greater than or equal to the limit) and,
- `extractMax()` will never report any integer that is currently already in the Binary (Max) Heap that happen to be *greater than or equal to* the limit l.

You can assume that initially (when the Binary (Max) Heap is instantiated), $l = \infty$, i.e., there is no limit initially. Another peculiar property is that each call `setLimit(l)` will make the limit **strictly lower** than the previous limit. An example run:

1. Starting from an empty Binary (Max) Heap, we `insert(100)`, `insert(60)`, `insert(65)`, `insert(70)`, `insert(77)`, `insert(70)` (duplicate is allowed) in that order.
2. If we call `extractMax()` now, we will extract and then return 100.
3. If we then `setLimit(70)`, then the limit is now set at $l = 70$.
4. Notice that the limit $l = 70$ now, so subsequent calls of `setLimit(l)`, if any, will set a strictly lower limit than this (never greater than or equal to), i.e., `setLimit(l)` where $l \geq 70$ will never be called from this point onwards.
5. If now we call `extractMax()` again, we need to report 65 as the two copies of 70s and 77 are all greater than or equal to the current limit $l = 70$.
6. If now we call `insert(70)`, it will be rejected as the limit is now set at $l = 70$.
7. If now we call `extractMax()` twice now, the first reports 60 and the second one reports none.

Propose the required modification(s) to Binary (Max) Heap data structure that you have learned in class in order to support this new PQ operation. To standardize, let's use `https://www.comp.nus.edu.sg/~stevenha/cs2040c/demos/BinaryHeapDemo.cpp` as the baseline. You will get partial marks of at most 12 out of 20 points if any of these three operations `insert(v)`, `extractMax()`, and/or `setLimit(l)` becomes $O(n)$ or worse (but they works). That's it, to get full 20 out of 20 points, all three operations must work and run in $O(\log n)$ or better.

E Easy Marks (5 marks)

To qualify for up to easy 5 marks, you need to **write both full names correctly**.
My CS2040C lecturer is _____ and Teaching Assistant (TA) is _____,

Write a **short** (maybe limit yourself to up to just 2 minutes to do this and about 3-4 sentences) **but honest (and not anonymous)** feedback on what you have experienced in the first 6 weeks of CS2040C in Semester 1 AY 2022/23 (including Week -02/-01 experience, if any). Feedback that are shared by *majority* (**not a one-off**) and can be easily incorporated to make the next 7 weeks of CS2040C better will be done. Grading scheme: 0-blank, 3-considered trivial feedback but not blank, 5-good and constructive feedback, thanks. (Penalty -1 mark for each wrong name above...).

This is the answer sheet that you will hand in later, write your Student Number in the box below:

A	0							

This portion is for examiner's use only

Section	Maximum Marks	Student Marks	Remarks
A	40		
B	30		
C	5		
D	20		
E	5		
Total	100		

My section A answers (note that the grading scheme is 4 vs 0 marks).
However, if you supply an explanation (use page 8), you will score non-zero mark even if it is wrong.

1 = O(_____)	2 = O(_____)	3 = O(_____)	4 = O(_____)	5 = O(_____)
6 = O(_____)	7 = O(_____)	8 = O(_____)	9 = O(_____)	10 = O(_____)

My section B.1.1 answer:

My section B.1.2 answer:

My section B.1.3 answer:

My section B.1.4 answer:

My section B.2 answer:

My section B.3.1 answer:

My section B.3.2 answer:

My section B.4 answer (can be skipped if your B.5 is correct):

My section B.5 answer (very strict grading, if you can't find $O(n \log n)$ solution, do B.4 first):

Please write your Student Number again in the box below (just in case this page is detached):

A	0							

My section C answers (you are allowed to review your own code on your laptop and even if you don't have it, i.e., you solved it on your home desktop, you still remember how to solve it, don't you?):

My section D answers:

My section E answers (two names and a short remarks, no need to be long-winded):

This page 8 is for extra writing space if you need any (e.g., to elaborate on your Section A answers). But if you ever need it, Steven thinks you are *probably* already digressing to wrong answers...

– End of this Paper, All the Best –