Midterm Examination Problem Sheet
TIME: 04/20/2021, 13:20–16:10

This is a open-book exam. You can use any printed materials as your reference during the exam. Any electronic devices are not allowed.
Any form of cheating, lying or plagiarism will not be tolerated. Students can get zero scores and/or get negative scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.
Both English and Chinese (if suited) are allowed for answering the questions. We do not accept any other languages.
*Unless explicitly asked, you do **not** need to justify why your algorithm runs in the desired time/space complexity, but you are welcomed to briefly explain the idea to help the TAs grade your solution.*

There are 16 problems. 8 of them are marked with * and are supposedly simple(r); 4 of them are marked with ** and are supposedly regular; 4 of them are marked with *** and are supposedly difficult. The simpler ones are worth 10 points, and others are worth 15 points. The full credit is 200 points. The 16 problems are organized to 8 sections, each with two problem. You can use one and only one A4 paper per section. For each section, please write down the section ID (top-left), your name and school ID (top-right) on the A4 paper. Then, write down the problem ID and your answer to each problem as instructed. You should keep your answers as *concise* as possible to facilitate grading. In particular, please use proper *pseudo code* to describe your algorithm!

1 Queue with Circular Array

1. (*) Consider a queue implemented with a circular array of size 8. The items are being enqueued into the array with strictly increasing values $v_1 < v_2 < \ldots < v_M$. Assuming that the number of items that are in the queue is always no more than 8. Show the contents of the circular array after 11 enqueue and 3 dequeue operations.

2. (***) Student Salmon was not very careful and lost the note that stores the `front` and `back` indices to the circular array in the question above. Given any circular array of size N used to store enqueued items with strictly increasing values $v_1 < v_2 < \ldots < v_M$, such that the circular array is full, i.e., all N locations are non-empty. Design an $O(\log N)$-time and $O(1)$-extra-space algorithm that helps Salmon find the `front` index of the circular array.

2 Linked List

3. (*) Consider a linked list like this

   `head->1->2->3->...->(2K)->NIL`

   with $K > 1$. Design an algorithm that "un"-shuffles the linked list, which means that the new linked list should have the same set of nodes as the original one, while looking like

   `head->1->3->5->...->(2K-1)->2->4->6->...->(2K)->NIL`

   Each number (the data field) within the original node cannot be changed. You can only change the `next` link in each node in your algorithm.

4. (**) Assume that you have a linked list with K elements but you do not know K in advance. You only know that $K > 1$ is a multiple of 5 and have the `head` pointer to the first node of the list on hand. Design an $O(K)$-time and $O(1)$-space algorithm, *without counting K explicitly*, that returns the data stored within the $(\frac{2}{5}K)$-th node of the linked list, assuming that nodes are numbered $1, 2, \ldots, K$.

3 Complexity!

For the next two sections, please prove the results using the following definition of the asymptotic notations (which is exactly the same as the notation you used in Homework 1). You can use any tool that you have learned in your calculus or discrete math class, but you cannot use anything other than the following definition (and the hints in the problems). Suppose that f and g are real-valued functions defined on $\mathbb{R}^+$, and $g(x)$ is strictly positive for all sufficiently large x (formally, there is some x_1 such that $g(x) > 0$ for all $x > x_1$). We call $f(x) = O(g(x))$ if and only if there exist positive numbers c and x_0 such that

$$|f(x)| \leq cg(x) \text{ for all } x > x_0.$$

5. (*) Using the inequality $\ln x \leq x - 1$ for all $x > 0$, prove that $\ln n = O(n)$. Please note the inequality comes from computing the tangent line of $f(x) = \ln x$ at $x = 1$.

6. (**) Prove that $\ln\left((g(n))^k\right) = O(g(n))$ for any real number $k > 0$ and any monotonically increasing $g(n)$.

4 Oh No, More Complexity!!

7. (**) Prove that $n^k = O(\exp(n))$ for any real number $k > 0$. (*Hint: the concept of "tangent line" for $f(x) = \ln x$ may help.*)

8. (***) Prove or disprove that for $f(n) = O(g(n))$, $\exp(f(n)) = O(\exp(g(n)))$. Note that the previous question is a special case of this claim using $f(n) = \ln(n^k)$ and $g(n) = n$.

5 Cut-In Queue?!

9. (*) Number the locations in the size-N queue as $1, 2, 3, \ldots, N$, with location 1 being the front. One add-on operation to the queue is to `enmiddle`, which inserts an element to the ceiling($\frac{N}{2} + 1$)-th location of the queue of N elements. We will call this new data structure the cut-in queue. Design a cut-in queue implementation using *two regular deques*. Your implementation needs to support the following three cut-in queue operations: `enqueue`, `dequeue`, and `enmiddle`. Assume in the regular queue (implemented naively with one deque) the first two operations both can be achieved with $O(1)$ running time. Your implementation of cut-in-queue can only use $O(1)$ extra space and each of the three operations need to be completed in $O(1)$ time.

10. (*) Student Tuna loves the cut-in queue and decides to write some code to test it. Somehow the code has some bugs, and a for-loop that originally implements

    ```
    for(i=0;i<2*N;i++) enqueue(i);
    ```

    was mistyped as

    ```
    for(i=0;i<2*N;i++) enmiddle(i);
    ```

    The original implementation intends to have a queue that stores an ordered sequence from 0 to $2N - 1$. Now the cut-in queue does not have an ordered sequence. Luckily, Tuna found that the cut-in queue library he used is actually a cut-in deque (given that two underlying deques were used to implement it). That is, it supports the `pop_front` and `pop_back` operations. Design an algorithm that use those two operations to "correct" Tuna's bug. That is, your algorithm should pop and print 0 to $2N - 1$ in order from the cut-in deque that Tuna's code constructed.

6 Stack

11. (**) Consider a binary search tree T (with its `root` as the entry point) of size N, and one empty stack S. Design a non-recursive algorithm (with $O(N)$ time and $O(1)$ extra space) that uses the stack, with only `push`, `pop`, `peep` (without popping) operations, to print the sorted keys (i.e., the in-order traversal) of T.

12. (***) Consider a stack with the usual `push`, `pop`, `peep` (without popping) operations on integers. Design a data structure that supports an additional operation: `peep_max`, that peeps the maximum element within the stack directly (when the stack is non-empty), without popping. All operations needs to be done in $O(1)$ time. Note that this task can actually be done in $O(1)$ extra space, but you will be allowed $O(N)$ extra space (where N is the size of the stack) if needed.

7 Tree and Heap

13. (*) Consider a tree where each node links to a parent (except the root, of course). Design an $O(h)$-time $O(1)$-extra-space algorithm that returns the closest common ancestor of two different nodes, where h is the height of the tree. For instance, in the following director tree,

    ```
    dsa/instructor
    dsa/ta/ta1
    dsa/ta/ta2
    dsa/ta/ta3
    dsa/hw/hw1/write
    dsa/hw/hw1/prog
    dsa/hw/hw2/write
    dsa/hw/hw2/prog
    dsa/hw/hw2/data
    dsa/midterm/goodluck
    ```

    The closest common ancestor of two nodes `hw1` (within `dsa/hw/`) and `data` (within `dsa/hw/hw2/`) is `hw` (within `dsa/`).

14. (*) Student Urchin has a conjecture that every size-N array sorted in a descending order is actually a binary max-heap. Assume the elements in this array are indexed by $1, 2, \ldots, N$ (element indexed by i is called element i thereafter). Then, as taught in class, a complete binary tree can be represented by this array. The root of the tree is represented by element 1. Each tree node, represented by element i, has a left child represented by element $2i$, and a right child represented by element $2i + 1$. Prove Urchin's conjecture: If the array is sorted in a descending order, the corresponding complete binary tree is a max-tree (which in term makes the array a max-heap).

8 Sorted Heap?!

15. (*) Student Wagyu designs a new data structure called a "sorted heap." In addition to the usual heap properties, for any sub-tree within the sorted heap, she requires that any node in the left sub-sub-tree holds a smaller key than the one of any node in the right sub-sub-tree. Show a 7-node (or more) example that demonstrate the existence of the "sorted heap" data structure.

16. (***) Prove that the post-order traversal on the sorted heap above visit the node keys in ascending order.