CS 101 Fall 2024 - Quiz 1B

October, 8, 2024 - 20 Minutes **Name:** **Student ID:**

1. **(2 points) Honor Code**

I promise that I will complete this quiz independently and will not use any electronic products or paper-based materials during the quiz, nor will I communicate with other students during this quiz.

I will not violate the Honor Code during this quiz. √ True ○ False

2. **(10 points) True or False**

Determine whether the following statements are true or false.

(a) (1’) In any circular doubly linked list, you are able to traverse the entire list starting from any node. √ True ○ False

Solution: Obviously.

(b) (1’) Linked list is more efficient than array when we only want to find some element with specific value. ○ True √ False

Solution: Finding by value is $O(n)$ for both linked list and array. And array is more efficient in actual performance because of smaller constant factor.

(c) (1’) In any singly linked list, removing the last element requires $O(1)$ time. ○ True √ False

Solution: Singly linked list is not guaranteed to maintain the tail pointer, in which case removing the last element requires $\Theta(n)$ time.

(d) (1’) We want to maintain a database which stores students’ names and ids and we only import all the data when students get admitted. After that, dropout seldom happens and transferring student never appears. We’d better use array instead of linked list. √ True ○ False

Solution: If inserting/removing elements in the middle seldom happens, array is more efficient in actual performance because of smaller constant factor.

(e) (1’) In any stack, you are able to access elements in the middle of the stack without popping the top elements. ○ True √ False

Solution: Unlike array, random access is not guaranteed for stack. For example, a stack implemented with linked list.

(f) (1’) In a stack implemented using an array, it is possible that the push operation result in a stack overflow. √ True ○ False

Solution: Obviously.

(g) (1') If we implement a queue using a circular array, the minimal memory we need is related to the maximal possible numbers of elements in the queue.

✓ True ○ False

Solution: Obviously.

(h) (1') If $f(n) = n \log n$ then for all $\alpha \geq 1$, we have $f(n) = o(n^{\alpha})$. ○ True ✓ False

Solution: When $\alpha = 1$, we have $f(n) = \omega(n)$.

(i) (1') For any two functions $f(n)$ and $g(n)$, if $f(n)$ is $O(g(n))$, then $g(n)$ is $\Omega(f(n))$.

✓ True ○ False

Solution: Obviously.

(j) (1') For an algorithm, it is possible that the worst-case running time is $O(n)$ and the best-case running time is $\Omega(n)$.

✓ True ○ False

Solution: It is possible when the running time is $\Theta(n)$ in all cases.

3. (4 points) Possible Order Popped from Stack

Suppose there is an initially empty stack of capacity 7, and then we do a sequence of 14 operations, which is a permutation of 7 `push(x)` and 7 `pop()` operations. If the order of the elements pushed to the stack is `1 2 3 4 5 6 7`, then for each sequence of elements listed below, determine whether it is a possible order of the popped elements. If possible, write down the 14 operations in order.

(a) (2') `3 2 4 6 7 5 1`

Solution: Possible: `push(1), push(2), push(3), pop(), pop(), push(4), pop(), push(5),push(6), pop(), push(7), pop(), pop(), pop()`

(b) (2') `2 4 5 6 1 3 7`

Solution: Impossible.

4. (7 points) Order the functions

Order the following functions so that for all i, j , if f_i comes before f_j in the order then $f_i = O(f_j)$. Do NOT justify your answers.

$$
\begin{aligned}
f_1(n) &= \sqrt{n} \\
f_2(n) &= n^{\frac{1}{4}} \\
f_3(n) &= 2^{\log_2 n} \\
f_4(n) &= \left(\tfrac{1}{2}\right)^n \\
f_5(n) &= 3^n \\
f_6(n) &= \log_2 n \\
f_7(n) &= 2^{\sqrt{n}} \\
f_8(n) &= n!
\end{aligned}
$$

As an answer you may just write the functions as a list, e.g. $f_8, f_4, f_1, \ldots$.

Solution:

$$f_4, f_6, f_2, f_1, f_3, f_7, f_5, f_8$$

$$\left(\frac{1}{2}\right)^n, \log_2 n, n^{\frac{1}{4}}, \sqrt{n}, 2^{\log_2 n}, 2^{\sqrt{n}}, 3^n, n!$$

5. (4 points) Analysing the Time Complexity of a Function

We are going to analyze the average-case time complexity of function FOO. Assume that all basic operations take constant time.

```
function FOO($a_1, a_2, \cdots, a_{n-1}, a_n$)    ▷ $a$ is an array with $n$ elements
    $max \leftarrow a_1$    ▷ $max$ is the maximal value among the first $i$ elements
    for $i = 2$ to $n$ do
        if $max < a_i$ then
            $max \leftarrow a_i$
            $(a_1, a_2, \cdots, a_{i-1}, a_i) \leftarrow (a_i, a_{i-1}, \cdots, a_2, a_1)$    ▷ Reverse the first $i$ elements
        end if
    end for
end function
```

The probability of entering the `if` body in the i-th `for` iteration is $\underline{\quad\quad 1/i \quad\quad}$, because it is the probability that a_i has the maximal value among the first i elements. (Assuming all elements in array a is independent and evenly distributed.)

And the time complexity of the `if` body in the i-th `for` iteration is $\Theta(i)$ because we need to reverse the first i elements.

Therefore the average-case time complexity of the `if` statement is $\Theta(\underline{\quad\quad 1 \quad\quad})$.

Solution: $\frac{1}{i} \times \Theta(i) = \Theta(1)$

And the `for` loop iterates $\Theta(n)$ times , so the average-case complexity of `for` loop is $\Theta(\underline{\quad\quad n \quad\quad})$.

Solution: $n \times \Theta(1) = \Theta(n)$

Therefore the average-case time complexity of FOO is $\Theta(\underline{\hspace{4em} n \hspace{4em}})$.