National University of Singapore

School of Computing

CS3230 - Design and Analysis of Algorithms
Midterm Test

(Semester 2 AY2022/23)

Time Allowed: 90 minutes

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this assessment paper until you are told to do so.
2. This assessment paper contains TWO (2) sections.
   It comprises ELEVEN (11) printed pages, including this page.
3. This is an **Open Book** Assessment.
4. For Section A, use the OCR form provided (use 2B pencil).
   You will still need to hand over the entire paper as the MCQ section will not be archived.
5. For Section B, answer **ALL** questions within the **boxed space**.
   If you leave the boxed space blank, you will get automatic 1 mark (even for Bonus question).
   However, if you write at least a single character and it is totally wrong, you will get 0 mark.
   You can use either pen or pencil. Just make sure that you write **legibly**!
6. Important tips: Pace yourself! Do **not** spend too much time on one (hard) question.
   Read all the questions first! Some questions might be easier than they appear.
7. You can assume that all **logarithms are in base** 2.
8. Please write your Tutorial Group, '_', and Student Number only. Do **not** write your name.

T			_	A	0							

This portion is for examiner's use only

Section	Maximum Marks	Your Marks	Grading Remarks
A	39		
B	61		
Total	100		

A Multiple Choice Questions ($13 \times 3 = 39$ marks)

Select the **best unique** answer for each question. Each correct answer worth 3 marks.

The rest of this page 1 is REDACTED.

This page 2 is REDACTED.

This page 3 is REDACTED.

This page 4 is REDACTED.

This page 5 is REDACTED.

B Essay Questions (61 marks)

B.1 Prove that case 3 regularity condition is always satisfied in PA1-A (14 marks)

In Programming Assignment 1 (PA1), task A, we have to use master theorem to automatically solve recurrences in the form of:

$$T(n) = a \cdot T(\tfrac{n}{b}) + c \cdot n^d \log^k n$$

We are also given the following constraints $a > 0$, $b > 1$, $c > 0$, $d \geq 0$, and $k \geq 0$.

When case 3 of master theorem is applicable, many students do *not* also check if the required regularity condition is also satisfied, yet all of them still get the Accepted verdict (assuming there is no other bug other than skipping regularity condition check on case 3 situations). This is not because the test cases are weak. In fact, the regularity condition is *always* satisfied for case 3 of master theorem in this semester's PA1. Your job in this question is to formally prove it.

B.2 Exponentiation in Addition Machine (35 marks)

We have learned that we count the number of instructions the algorithm takes to measure its running time. If you recall our first lecture, we consider the Word-RAM as our computation model because this model resembles our modern computers. However, what about old computers with a more primitive computation model?

In 1990, Robert Floyd and Donald Knuth investigated a computation model called Addition Machine. This model only has the following limited arithmetic instructions:

- Addition $(+)$
- Subtraction $(-)$
- Comparisons $(=, \neq, <, \leq, >, \geq)$

Simply put, this model is equivalent to any modern language (e.g., C++, Java, or Python) but **WITHOUT** using multiplication, division, modulo, exponentiation, or even bit-wise operations. You can assume that this Addition Machine model is a restricted Word-RAM model.

So if we want to multiply $x \times y$ or divide (and round down) $\lfloor x/y \rfloor$, we can naively implement them as follows:

- For multiplication, we can repeatedly increment a temporary variable by x, for y many times, assuming $y \leq x$. If $x < y$, then we swap the x and y first, thus this $MULTI(x, y)$ runs in $\Theta(min(x, y))$ time.
- For division (with round down), we can repeatedly decrement x by y as long as x stays non-negative. The number of repetitions will be the answer, thus this $DIV(x, y)$ takes $\Theta(x/y)$ time.

B.2.1 Naive Exponentiation (5 Marks)

Suppose that we want to implement an exponentiation function a^n naively by multiplying a for n many times as the following:

```
int NAIVE_EXP(int a, int n) {
    if (n == 0) return 1;
    else return MULTI(a, NAIVE_EXP(a, n-1));
}
```

What is the time complexity, in $\Theta(\cdot)$ notation, of the above function?
For simplicity, assume that $a \leq n$ and the inputs are non-negative.

B.2.2 Fast Exponentiation? (14 Marks)

You have learned from our past lecture that there is a “faster” algorithm for exponentiation using a Divide and Conquer technique like the following:

```
int FAST_EXP(int a, int n) {
    if (n == 0) return 1;
    else if (n == 1) return a;
    else {
        int temp = FAST_EXP(a, DIV(n, 2));
        temp = MULTI(temp, temp)
        if (IS_ODD(n)) temp = MULTI(a, temp);
        return temp;
    }
}
```

Even by assuming that the *IS_ODD* function takes $O(n)$ time, you might think that this algorithm should run faster than the naive one, **but is it really true**? Prove (or disprove) by finding the time complexity, in $\Theta(\cdot)$ notation, of the $FAST\_EXP$ function! For simplicity, assume that $a \leq n$, n is a power of 2, and the inputs are non-negative.

B.2.3 Squaring a Number (14 Marks)

While it is not trivial, actually there is a faster division implementation such that $DIV(x, y)$ runs in only $\Theta(\log(x/y))$ time. We can then use it to implement $IS\_ODD(n)$ also in $\Theta(\log n)$ time. You don't need to prove them.

Instead, your job is to implement the algorithm of $SQUARE(n)$. This algorithm should return the value of n^2, and you may assume that n is always non-negative. You must also analyze the time complexity of your algorithm (using $\Theta(\cdot)$ notation). You may call the naive $MULTI$, the faster DIV, and the faster IS_ODD functions in your implementation.

To get a full mark, your algorithm should run in $O(\log^2 n)$ time, and you need to provide a correct $\Theta(\cdot)$ analysis. Partial 3 marks will be given for any algorithm which runs in $\Theta(n)$ time. **HINT**: Use Divide and Conquer technique!

B.2.4 Lesson Learned (2 Marks)

Lastly, please write anything that you learned from these small "experiments"! :)

B.3 Moderately Small Element (12 marks)

Given an unsorted array of n integers, we know how to find the smallest element in $O(n)$ time, and also we have learned in the lecture that $\Omega(n)$ time is necessary for this purpose. Now suppose we aim to find a *moderately small* element (instead of the smallest element). For an n-length array A, we call the element $A[i]$ *moderately small* if its rank is at most $n/10$. (Recall, if $A[i]$ is the j-th smallest element in the array A, then its rank is j.)

Given an unsorted array of length n, our objective is to find a moderately small element in $o(n)$ time with the help of randomization. For that purpose, try to solve the following questions.

B.3.1 What is the Probability (I)? (2 marks)

Let us pick an index i uniformly at random from the set $\{0, 1, \cdots, n-1\}$, and return $A[i]$. What is the probability that the returned output is a moderately small element?

B.3.2 What is the Probability (II)? (7 marks)

Let us now modify the procedure described in subsection B.3.1 as follows: Pick a *sequence* of indices $i_1, i_2, \cdots, i_s$ uniformly at random and independently from the set $\{0, 1, \cdots, n-1\}$ with replacement. Then output the *smallest* element among the set $\{A[i_1], A[i_2], \cdots, A[i_s]\}$. What is the probability that the returned output is a moderately small element?

B.3.3 What is the Running Time? (3 marks)

Given any n-length array, if you want to output a moderately small element with probability at least $1 - \frac{1}{n}$, what would be the tightest $O(\cdot)$ bound on the time complexity of the procedure described in subsection B.3.2?

B.4 Bonus Question (5 marks)

Suppose you are given as input a circular array $A[0 \cdots n-1]$ of length n containing all the distinct integers between $\{1, 2, \cdots, n\}$ in an arbitrary order. Your goal is to decide whether there exists three consecutive indices $i, i+1, i+2$ such that $A[i] + A[i+1] + A[i+2] > 1.5 \cdot n$. (Note, since the input array is circular, you should actually consider $i+1 (\mod n)$ and $i+2 (\mod n)$.) So if there exists such three consecutive indices, you should output "YES"; otherwise "NO". How many cells of the input array A you must read (in the worst-case) to output the correct answer? (Provide proper explanation in support to your answer.) [**No partial marks** will be given for this question.]

– END OF PAPER; All the Best –