NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
Midterm (30%)
AY2019/20 Semester 2

CS2040 – Data Structures and Algorithms

7 March 2020 Time allowed: 90 minutes

INSTRUCTIONS TO CANDIDATES

1. Do **NOT** open the question paper until you are told to do so.
2. This question paper contains TWO (2) sections with sub-questions. Each section has a different length and different number of sub-questions. It comprises Ten (10) printed pages, including this page.
3. Answer all questions in this paper itself. You can use either pen or pencil. Write **legibly**!
4. This is an **Open Book Quiz**. You can check the lecture notes, tutorial files, problem set files, CP3 book, or any other books that you think will be useful. But remember that the more time that you spend flipping through your files implies that you have less time to actually answer the questions.
5. When this Quiz starts, **please immediately write your Matriculation Number** and **Tutorial Group (if you don't know your tutorial group, write your TA name and time slot)**.
6. The total marks for this paper is **100**.

TUTORIAL GROUP

STUDENT NUMBER: | A | | | | | | | | |

For examiners' use only		
Question	***Max***	***Marks***
Q1-4	12	
Q5	35	
Q6	35	
Q7	18	
Total	**100**	

Section A – Analysis (12 Marks)

Prove (the statement is correct) or disprove (the statement is wrong) the following statements below. If you want to prove it, provide the proof or at least a convincing argument. If you want to disprove it, provide at least one counter example. 3 marks per each statement below (1 mark for circling true or false, 2 marks for explanation):

1. The tightest time complexity of the following code fragment is $O(n)$. **[true/false]**

```
for (int i=1;i <= n; i=i*8) {
    System.out.println("outer loop");
    for (int j=1; j <= i; j=j*2) {
        System.out.println("Inner loop");
    }
}
```

2. Using a basic linked list we can only insert and remove from the front in O(1) time. However using a tailed linked list we can insert and remove from both the front and back in O(1) time. **[true/false]**

3. If separate chaining is used as the collision resolution technique for a hashtable, and load factor cut off is set at 10. This means that whenever the chain in any row exceeds length 10 we will rehash everything into a bigger hashtable. **[true/false]**

4. For insertion sort, we need to perform a linear scan from the back of the sorted portion to find the correct position to insert each value in the unsorted portion of the array. This causes insertion sort to be O(N^2). We can improve this to O($NlogN$) by performing a binary search to find the insertion point instead.
[true/false]

Section B – Applications (88 Marks)

Write in pseudo-code. Any algorithm/data structure/data structure operation not taught in CS2040 must be described, there must be no black boxes. Some **partial marks** will be awarded for correct answers not meeting the time complexity required.

5. **Genome analysis [35 marks]**

Given a gene represented as a **basic linked list** (as defined in the lecture notes) of characters from the set {'A', 'C', 'G', 'T'}, you are to implement 2 common operations in genome analysis as follows:

a.) *Gene Translocation* – given 2 genes K and L of length $|K|$ and $|L|$ respectively, where $|K|,|L| > 1$, swap the portion of K from index 0 to i with the portion of L from index j to $|L| - 1$, where $0 \leq i < |K| - 1$ and $1 \leq j < |L|$.

E.g K = ACCGTC , L = AGGTCCCT, i=2 and j=4, translocation will result in

K = CCCTGTC, L = AGGTACC

Implement gene translocation in time $O(|L| + |K|)$. **[18 marks]**

b.) *Longest Common Suffix Length* – given 2 genes K and L of length $|K|$ and $|L|$ respectively, where $|K|,|L| > 1$, return the length of the longest common suffix of the 2 genes (0 if there is no common suffix). The suffix of a gene refers to the part of a gene from

some index i ($0 <= i < length\ of\ gene$) to the end of the gene. The longest common suffix is the longest suffix of K and L that match each other.

E.g K = ACCGTC, L = AGGTCGTC, the longest common suffix is as underlined and of length 4.

Implement longest common suffix length in time $O(|L| + |K|)$. You cannot use any additional data structures (no array, no linked list, no stack, no queue etc …) to help you. You can modify the K and L if necessary. **[17 marks]**

6. **Mountain Peaks [35 marks]**

a) John is creating a game where a player has to climb a series of mountain peaks of different heights which can be from 0 to M meters above sea level, where $1{,}000 \leq M \leq 1{,}000{,}000$. In order to generate the peaks, John has created an array A of 50 randomly generated positive integers between 0 and M. John then starts going through A from the 1st to the last integer and accumulate the numbers by **multiplying** them. Consider each multiplication as a step, at step i the current accumulated total c_i will be the height of the i^{th} peak. If $c_i \geq M$, c_i will be set to $c_i \% M$. The height of $c_1$the first peak is set to $A[0]$.
Once he reaches the last integer he will start from the 1st integer again and repeat the process ad infinitum.
For e.g if $A = \{3,1,2\}$ and $M = 10$, the first 5 heights will be as follows:

$c_1 = 3$, $c_2 = 3 * 1 = 3$, $c_3 = 3 * 1 * 2 = 6$, $c_4 = 3 * 1 * 2 * 3 = 18\%10 = 8$,
$c_5 = 3 * 1 * 2 * 3 * 1 = 18\%10 = 8$

In fact, the first 10 peaks generated will be 3,3,6,8,8,6,8,8,6,8 ….
John soon realize that the sequence of heights would start repeating after a while. Let the repeated subsequence be R (6,8,8 in the example above). Your job is to come up with an algorithm to return one integer that is in R (any integer in R will do, so 6 or 8 in the example above) in $\leq O(M)$ time given A the array of 50 random integers. **[18 marks]**

i.) State the data structure(s) you will use to help you (N.A if no DS required)

ii.) Give the algorithm

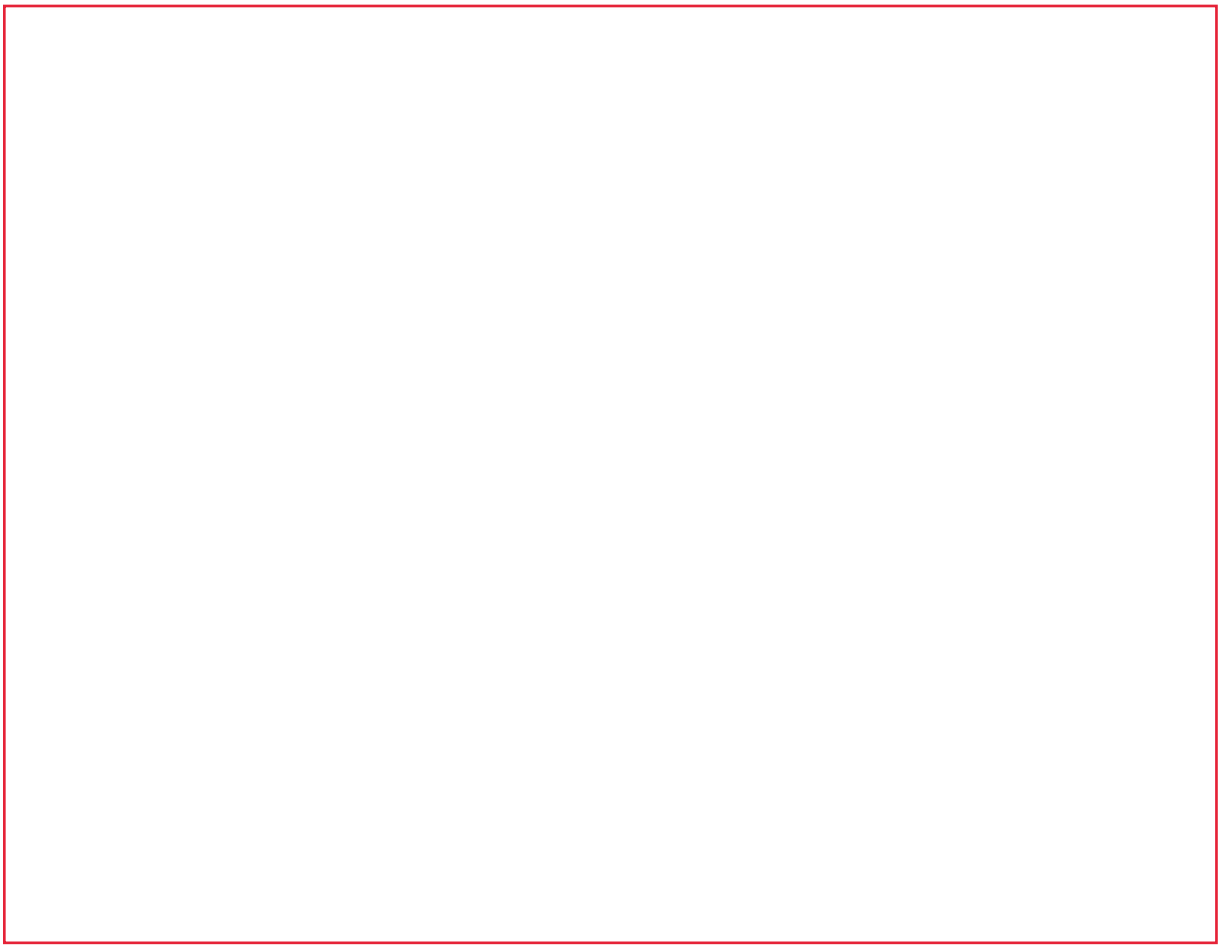
[**This may be a difficult sub-question*]

b) After helping John with his problem in a), he now returns to you with another problem. He has included the ability for the player to rappel from a peak X to another peak Y if height of $X \geq$ height of Y and all peaks in between X and Y are shorter than X and Y. John has now stored the 1st N integers ($N > 1$) out of the sequence R from a) in an array B. He wants you to find out what is the maximum number of peaks m between a pair of X and Y among all possible pairs of X and Y in B where the player can rappel from X to Y. Your algorithm should run in time $O(N)$ or better. **[17 marks]**

E.g if B = {5,11,10,3,12,7}, the pair X=12 and Y=11 has the maximum number of peaks in-between them, i.e 2, which should be output by your program. You will get the same result if 11 and 12 were swapped i.e B' = {5,12,10,3,11,7}.

i.) State the data structure(s) you will use to help you (N.A if no DS required)

ii.) Give the algorithm



7. **Scramble [18 marks]**

You have been invited to a game show "Scramble". In this game show, you will be presented with a sequence R_o of N $(N > 1)$ non-repeating integers sorted in increasing order. Then immediately, the game show host will press a "scramble" button which as the name suggest will scramble the sequence so that the integers will no longer be in sorted order. Let the scrambled sequence be R_s.

Now you will be given an integer A which is guaranteed to be in R_s and not the smallest value, and you are supposed to pick out (in any order) including A itself the 1st X largest integers $\leq A$, where $X > 0$. If there are less than X integers $\leq A$ then pick out all integers $\leq A$. Let the set of such integers be A'.

For e.g if R_o= {1,3,7,31,51} and A = 31, X = 3 then A' = {3,7,31}.

if we change X = 5, then A' = {1,3,7,31}

The game will proceed in rounds. For each round you will pick out 2 integers B and C from R_s. Without loss of generality assume $B < C$. Once they are picked, they will be placed in their position i_B and i_C in the original sorted sequence R_o, and all integers D where $B < D < C$ will also be placed in the positions between i_B and i_C although they will also be scrambled and each integer will also be at most $\leq \lfloor 0.3 * (i_c - i_b - 1) \rfloor$ positions away from its sorted position. You can now do two things in the order listed below before the start of the next round

1. Pick out integers to be included in A'.
   An incorrectly picked integer will result in failure.
   You can also choose not to pick any integers at all for the round.
2. Remove all integers from i_B to i_C **OR** keep all integers from i_B to i_C and remove all other integers. The remaining integers are sorted (forming a new R_o) then scrambled to form a new R_s for the next round.
   Removed integers will no longer be available. So if any required integer is removed before it is included in A', the game will end in failure.

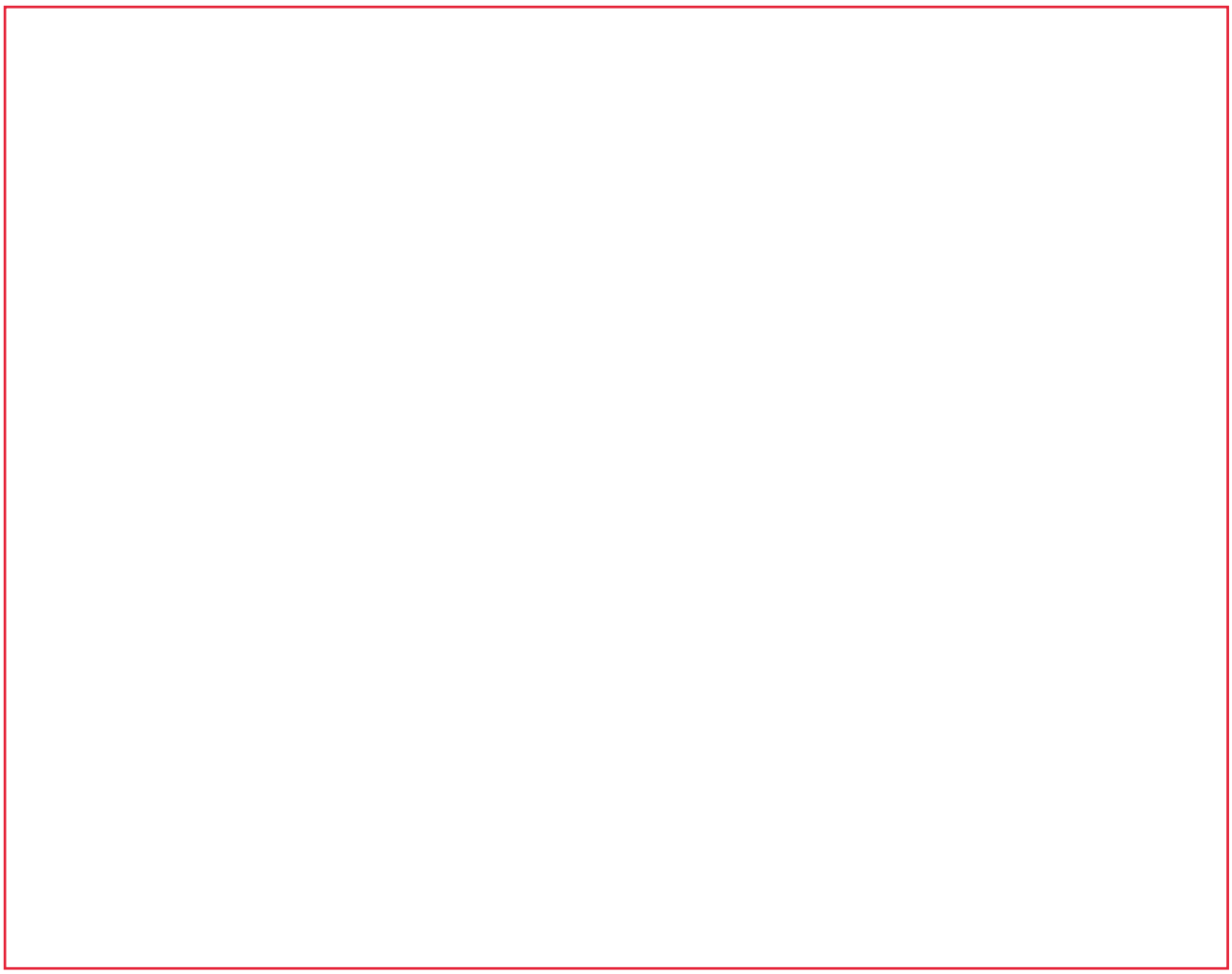
e.g of the game play:
If we start with R_s= {3,1,13,7,35,21,89,77,97,100,99} and A = 7, X = 2
for round 1 step 1, if we pick B=7, C=89 then we can have
R_s= {3,1,**7**,13,35,21,77,**89**,97,100,99} where integers between 7 and 89 is placed in the indices between 7 and 89 and they are again scrambled. Assume we don't pick any integer
to be in A' for this round.
For step 2, If we keep everything from 7 to 89 we will scramble those values and have R_s= {13,7,35,21,89,77} for round 2. If we throw everything from 7 to 89 away, we scramble the remaining integers and can have R_s= {3,1,99,97,100} for round 2.

If you can pick out all required integers in **at most $O(logN)$ rounds**, then you will win 1,000,000 dollars!

For each round there is a time limit and since you can only do computation mentally, an additional restriction is that you can only take $O(N)$ time in total to play the game and use at most $O(1)$ additional space.

Now given the afore mentioned restrictions, come up with an algorithm that will solve the problem in at most $O(logN)$ rounds and take in total $O(N)$ time and additional $O(1)$ space.

Non-viable solutions include memorizing the original sequence (it is too long), and using radix sort since it is too mentally challenging to perform (also requires too much space).



== END OF PAPER ==