National University of Singapore

School of Computing

CS2040 - Data Structures and Algorithms
Midterm Test @ SR2

(Thu, 12 Jul 2018, S4 AY2017/18, 100m)

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this assessment paper until you are told to do so.
2. This assessment paper contains SIX (6) sections.
   It comprises ELEVEN (11) printed pages, including this page.
3. This is an **Open Book Assessment**.
4. Answer **ALL** questions within the **boxed space** in this booklet.
   You can use either pen or pencil. Just make sure that you write **legibly**!
5. 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.
6. You can use **pseudo-code** in your answer but beware of penalty marks for **ambiguous answer**.
   You can use **standard, non-modified** algorithm discussed in class by just mentioning its name.
7. Write your Student Number in the box below:

A	0	1						

Section	Maximum Marks	Student Marks	Remarks
A	10		
B	15		
C	15		
D	15		
E	5		
F	40		
Total	100		

A Worst Case Time Complexity Analysis (10 marks)

Write down the *tightest*[1] *worst case* time complexity of the various data structure operations or algorithms below.

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 Java API (version 8). Unless otherwise mentioned, there are currently n elements in the data structure. `A` is a Java ArrayList currently with n elements. `L` is a Java LinkedList currently with n elements, `PQ` is a Java PriorityQueue currently with n elements. `v` is a 32-bit Integer.

No	Operations		Time Complexities
1	`A.remove(0);`		$O(\_\_\_\_\_\_\_)$
2	`for (int i = 0; i < n; i++) A.remove(0);`		$O(\_\_\_\_\_\_\_)$
3	`for (int i = 0; i < n; i++) A.remove(n-1-i);`		$O(\_\_\_\_\_\_\_)$
4	`A.remove(v)` // but `v` does not exists in `A`		$O(\_\_\_\_\_\_\_)$
5	`A.add(v)` // and `v` does not exists before in `A`		$O(\_\_\_\_\_\_\_)$
6	`for (int i = 0; i < n; i++) L.removeFirst();`		$O(\_\_\_\_\_\_\_)$
7	`for (int i = 0; i < n; i++) L.removeLast();`		$O(\_\_\_\_\_\_\_)$
8	`for (int i = 1; i <= n; i *= 2) PQ.insert(i);`		$O(\_\_\_\_\_\_\_)$
9	`for (int i = 0; i < n; i++) PQ.poll();`		$O(\_\_\_\_\_\_\_)$
10	`PQ.contains(v)` // `v` exists in `PQ`		$O(\_\_\_\_\_\_\_)$

[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.

Section A Marks = _____

B Analysis (15 marks)

Prove (the statement is correct) or disprove (the statement is wrong) the statements below.

1. Java `Collections.sort` *can* be made to run *slower* than $O(n \log n)$ (and more than $O(n^2)$) by using it to sort a collection of n specific items.

2. We *can* compute the number of inversions (number of Bubble Sort swaps) of an array A with n integers faster than $O(n^2)$.

3. Suppose that you have two Linked Lists SLLa and SLLb. SLLa/SLLb contains n/m unsorted alphabets [‘A’..‘Z’], respectively ($100 < n, m < 10\,000; n \neq m$). As we cannot get an element at index i in $O(1)$ if we use Linked List, to check whether an alphabet is inside *both* SLLa and SLLb, we need an $O(n \times m)$ algorithm that is roughly like this:
```
for (Character a :  SLLa) for (Character b :  SLLb) if (a == b) return true;
return false;
```

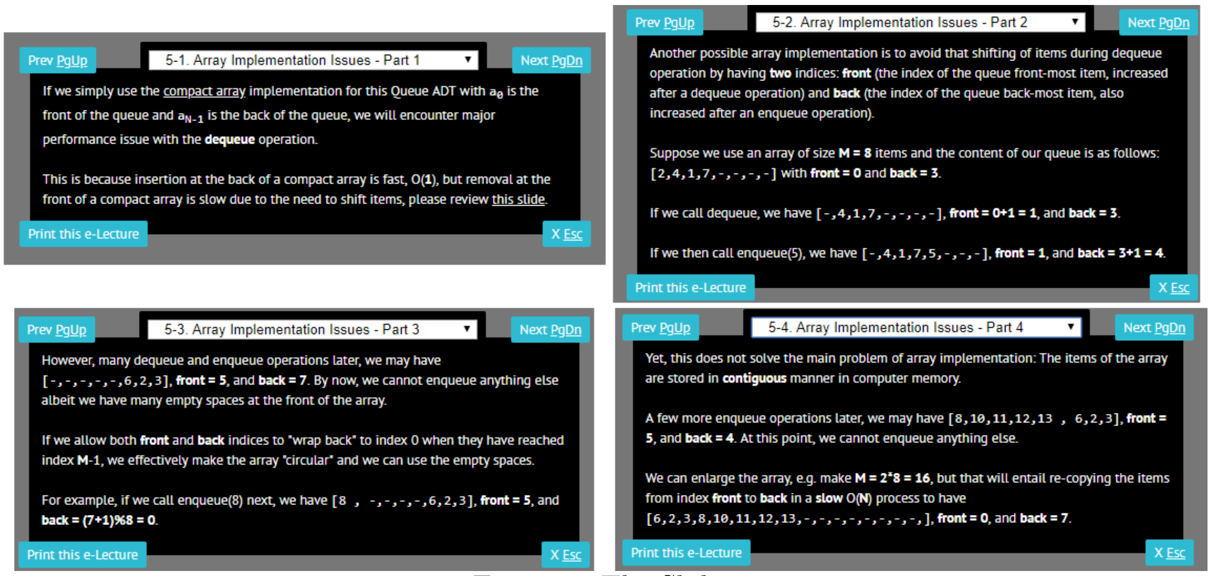
4. A Stack/Queue is a Last-In-First-Out/First-In-First-Out data structure, respectively. Thus, there is *no way* we can insert a sequence of n ($n > 1$) integers into a Stack and (the same sequence of n integers into) a Queue and when we peek the top/front of the Stack/Queue, respectively, we see the same integer.

5. Suppose that we need to use a *special kind* of ADT Priority Queue where all enqueue (Insert(v)) operations will be performed first *before* all subsequent dequeue (ExtractMax()) operations (from highest priority to lowest priority). For this kind of ADT PQ, we *can* use another data structure and/or algorithm to achieve similar time complexities as if we use Binary (Max) Heap.

Section B Marks = _____

C Alternative Implementation (15 marks)

In class, we have discussed that (Single) Linked List is likely one of the best possible implementation of a general ADT Queue. However, for some ADT Queue applications, we actually know what is the **maximum size** of the queue so we can use an array of that maximum size with two pointers (front and back) that can wrap-around that has been discussed in VisuAlgo e-Lecture slides (see Figure 1).



5-1. Array Implementation Issues - Part 1

If we simply use the compact array implementation for this Queue ADT with a_0 is the front of the queue and a_{N-1} is the back of the queue, we will encounter major performance issue with the **dequeue** operation.

This is because insertion at the back of a compact array is fast, O(**1**), but removal at the front of a compact array is slow due to the need to shift items, please review this slide.

5-2. Array Implementation Issues - Part 2

Another possible array implementation is to avoid that shifting of items during dequeue operation by having **two** indices: **front** (the index of the queue front-most item, increased after a dequeue operation) and **back** (the index of the queue back-most item, also increased after an enqueue operation).

Suppose we use an array of size **M = 8** items and the content of our queue is as follows: `[2,4,1,7,-,-,-,-]` with **front = 0** and **back = 3**.

If we call dequeue, we have `[-,4,1,7,-,-,-,-]`, **front = 0+1 = 1**, and **back = 3**.

If we then call enqueue(5), we have `[-,4,1,7,5,-,-,-]`, **front = 1**, and **back = 3+1 = 4**.

5-3. Array Implementation Issues - Part 3

However, many dequeue and enqueue operations later, we may have `[-,-,-,-,-,6,2,3]`, **front = 5**, and **back = 7**. By now, we cannot enqueue anything else albeit we have many empty spaces at the front of the array.

If we allow both **front** and **back** indices to "wrap back" to index 0 when they have reached index **M**-1, we effectively make the array "circular" and we can use the empty spaces.

For example, if we call enqueue(8) next, we have `[8 , -,-,-,-,6,2,3]`, **front = 5**, and **back = (7+1)%8 = 0**.

5-4. Array Implementation Issues - Part 4

Yet, this does not solve the main problem of array implementation: The items of the array are stored in **contiguous** manner in computer memory.

A few more enqueue operations later, we may have `[8,10,11,12,13 , 6,2,3]`, **front = 5**, and **back = 4**. At this point, we cannot enqueue anything else.

We can enlarge the array, e.g. make **M = 2*8 = 16**, but that will entail re-copying the items from index **front** to **back** in a **slow** O(**N**) process to have `[6,2,3,8,10,11,12,13,-,-,-,-,-,-,-,-,]`, **front = 0**, and **back = 7**.

Figure 1: The Slides.

Now, **what are the (positive, neutral, negative) implications of such implementation for an ADT Queue with known maximum size**? One such implication has been listed below. Your job is to list down *at least 5 (can be more...)* other logical statements of this implementation. Your answer will be graded based on the soundness and the quality of the statements (3 marks for each valid statement; -1 mark for random/irrelevant statement; min 0 mark and max 15 marks).

1. The enqueue operation remains $O(1)$.
2.
3.
4.
5.
6.

Section C Marks = ____

D Create Test Cases (15 marks)

Create Test Cases for each scenario below. Each valid test case worth 5 marks.

1. Supply an input array `X` of `N = 7` **distinct** integers $\in$ [1..7] so that the Optimized Bubble Sort (as currently implemented in VisuAlgo sorting page by 03 July 2018, during the discussion in Lecture 2a) requires **exactly 10 (Bubble Sort) swaps** to make `X` sorted in increasing order.

2. Supply an input array `X` of `N = 15` integers $\in$ [1..15] so that Randomized Quick Sort (as currently implemented in VisuAlgo sorting page by 03 July 2018, during the discussion in Lecture 2a) requires **at least 105 comparisons** to make `X` sorted in increasing order.

3. Given a Java Linked List that already contains 101 integers from 0 (head) $\leftrightarrow$ 1 $\leftrightarrow$ 2 $\leftrightarrow$ 3 $\leftrightarrow$ ... $\leftrightarrow$ 100 (tail). Show where to insert 4 more integers so that Java has to perform **at least 200 pointer advancements** to be able to insert those 4 more integers in their correct positions.

Section D Marks = _____

E Easy Marks (5 marks)

Write a **short** (maybe limit yourself to up to 5 minutes to do this) **but honest (and not anonymous)** feedback on what you have experienced in the first 6 sessions (3 weeks) of CS2040 in Special Semester 4. Suggestions that are shared by *majority* (**not a one-off feedback**) and can be easily incorporated to make the next 6 sessions (3 weeks) of CS2040 better will be done. Grading scheme: 0-blank, 1/2-considered trivial feedback but not blank, 4/5-good and constructive feedback, thanks.

Section E Marks = _____

F Applications (40 marks)

F.1 FIFA World Cup Group Stage (25 marks)

In the recent FIFA World Cup 2018, there is an interesting tie-breaking situation for the first time in history. Colombia, Japan, Senegal, and Poland were drawn together in group H. After Matchday 1, Japan beat Colombia 2-1 and Senegal *also* beat Poland 2-1 too. During Matchday 2, Japan and Senegal both had a draw 2-2. Finally after Matchday 3, Japan lost to Poland 0-1 and Senegal *also* lost to Colombia 0-1.

Because a win (scoring more goal(s) than opponent) worth 3 points, a draw (scoring equal number of goal(s) with opponent) worth 1 point, and a loss worth 0 point, then both Japan and Senegal were actually tied with 3+1 = 4 points each. To rank the teams, FIFA uses these rules, in decreasing order.

1. Highest number of points (using the +3/+1/0 for win/draw/loss above),
2. Goal difference (number of goals scored minus number of goals conceded, can be +ve/0/-ve),
3. Goals scored (**we stop here for this question**),
4. Points obtained in group games between teams concerned,
5. Goal difference from games involving teams concerned,
6. Number of goals scored in games between teams concerned,
7. Fair play points (*For the first time in history, a World Cup group has to be decided by rule so far down the list: Rule no 7, the fair play rule. Japan went through as they only had 4 yellow cards whereas Senegal had 6 yellow cards*),
8. Drawing of lots by FIFA (imagine that...)

For this question, let's simplify it a bit. You are given n football teams ($2 \le n \le 1\,000$) numbered from 1 to n in the first line and then a list of $n \times (n-1)/2$ results of matches between them each in one line with format: Team id a, a space, team id b ($1 \le a, b \le n$), a space, goal(s) x scored by team id a, a dash '-', and finally goal(s) y scored by team id b during their match ($0 \le x, y \le 31$).

Your task is to sort and then print n team results in correct order by using the **first three** tie-breaking three rules above **plus a rule that simplifies rules 4-8**: Highest number of points, if tie, by goal difference, if still tie, by goals scored. **If still tie at this point, prefer team with lower team id in this question**. The format of the n output lines is: Team id, a space, points of this team id, a space, goal difference of this team id, a space, goal scored by this team id.

A sample I/O is shown below. Team id 1/2/3/4 are actually Colombia/Japan/Senegal/Poland, respectively.

Sample Input	$\rightarrow$	Sample Output
4		1 6 3 5
1 2 1-2		2 4 0 4
3 4 2-1		3 4 0 4
2 3 2-2		4 3 -3 2
1 4 3-0		
2 4 0-1		
1 3 1-0		

A skeleton Java code has been written for you. Please complete it and **analyze its time complexity**.

```
import java.io.*;
import java.util.*;
class E1_WC {
  public static void main(String[] args) throws Exception {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    PrintWriter pw = new PrintWriter(System.out);
    // Elaborate how you will read the input (n + n*(n-1)/2 lines) and store them




















    // Elaborate how you will sort the teams here
    // Elaborate how you will print out the output (n lines) here









    pw.close();
  }
  // you can create a new (helper) class(es) if necessary









} // the overall time complexity of my Java code above is O(_____________)
```

F.2 Stacking Integers (15 marks)

You are given three stacks: $s1$, $s2$, $s3$ where $s2$ and $s3$ are initially empty and $s1$ contains n ($1 \le n \le 100\,000$) distinct positive integers $\in [1..n]$ but **in arbitrary order**.

Your job is simply to determine if it is possible to make $s3$ contains n positive integers in decreasing (sorted) order from top of stack to bottom, but you are constrained as follows:

1. You can only transfer integers from a **lower numbered** stack to a **higher numbered** stack.
2. You can use $s2$ as a temporary data structure (if necessary).
3. You are **NOT** allowed to use *any other* data structure in your solution.
4. You are allowed to use one (or two) more temporary integer variable(s) (if necessary).

For example, if $s1$ contains {5 (bottom), 2, 1, 3, 4 (top)} initially, then the answer is 'possible':

```
   (top) 4                                           5 (top)
         3                               4           4
         1           1                   3           3
         2           2  3        3  2    2           2
(bottom) 5           5  4     5  4  1    5     1           1 (bottom)
         -------- => -------- => -------- => -------- => --------
         s1 s2 s3    s1 s2 s3    s1 s2 s3    s1 s2 s3    s1 s2 s3
```

Two sample I/Os are shown below. Convince yourself that the answer for second sample is 'impossible':

Sample Input 1	→	Sample Output 1
5 2 1 3 4		true

Sample Input 2	→	Sample Output 2
1 5 2 4 3		false

A skeleton Java code has been written for you. Please complete it and **analyze its time complexity**.

```
import java.io.*;
import java.util.*;
class E2_Stack {
  public static void main(String[] args) throws Exception {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    PrintWriter pw = new PrintWriter(System.out);
    String[] token = br.readLine().split(" "); // there are n tokens
    Stack<Integer> s1 = new Stack<>();
    for (int i = 0; i < token.length; i++) // O(n)
      s1.push(Integer.parseInt(token[i])); // permutation of [1..n]
    pw.println(isPossible(s1));
    pw.close();
  }
  // is it possible to reorder content of s1 (a permutation of [1..n])
  // into its sorted permutation [1 (bottom), 2, ..., n-1, n (top)] in s3?
  private static Boolean isPossible(Stack<Integer> s1) {
    Stack<Integer> s2 = new Stack<>(); // the ONLY (temp) data structure allowed
    Stack<Integer> s3 = new Stack<>(); // contain [1, 2, ..., n (top)] at the end
    // you can use at most two more integer variables (if necessary)

  }
} // the overall time complexity of my Java code above is O(_____________)
```

– End of this Paper, All the Best –

Section F Marks = _____ + _____ = _____