National University of Singapore

School of Computing

CS2040C - Data Structures and Algorithms

Midterm Test

(Sat, 22 Sep 2017, S1 AY2017/18, 80m)

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this assessment paper until you are told to do so.
2. This assessment paper contains FOUR (4) sections.
   It comprises TEN (10) 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**.
   For full marks, you must use **C++ code** but beware of penalty marks for **compilation error**.
   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							

This portion is for examiner's use only

Section	Maximum Marks	Your Marks	Remarks
A	10		
B	15		
C	55		
D	20		
Total	100		

A Worst Case Time Complexity Analysis (10 marks)

Match the various data structure operations on the second column with the *tightest*[1] *worst case* time complexity(/ies) of those operations (the typical time complexities are given in the last column).

The operations (algorithms) referred below are the **unmodified version**, as per discussion in class, e.g. as currently discussed in VisuAlgo or as currently implemented in C++ STL.

Note that *it is possible* that there are some operations that have the same time complexity and thus there *may be* unused time complexity option(s).

No	Operations		Time Complexities
1	Insert a new item to the *front* of a C++ STL vector (resizeable array)		$O(1)$
2	Search for the k-th smallest item in a **sorted** *Array*		$O(\log n)$
3	Delete an item at index i of a C++ STL vector of size n and n-$i \leq 3$		$O(n)$
4	Search for an item in a **sorted** *Singly Linked List*		$O(n \log n)$
5	Pop **half** of the current contents of a C++ STL stack		$O(n^2)$
6	Pop the *most recently enqueued* item from a C++ STL queue		$O(n^2 \log n)$
7	Insert a new item *before* the tail item of a *Doubly Linked List*		$O(n^3)$
8	Insert a new item to the *front* of a C++ STL deque		$O(n^4)$
9	Extract *minimum* item of a Binary *Max* Heap using `ExtractMax()`		$O(2^n)$
10	Build a Binary *Max* Heap from any input (integer) array A of size n		$O(n!)$

Section A Marks = _____

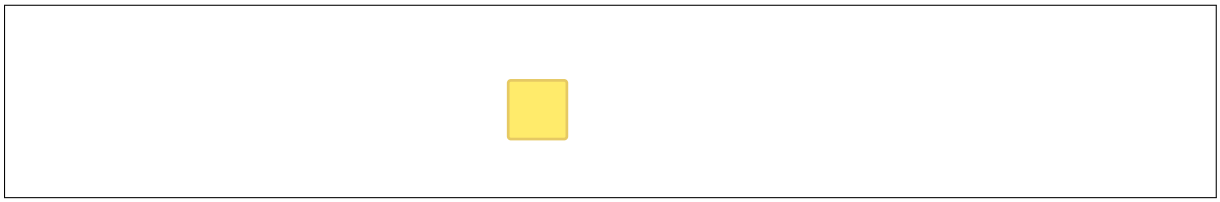
[1]What we meant by tightest worst case time complexity is as follows: If an algorithm/operation of the stipulated data structure needs at best $O(n^3)$ if given the worst possible input for that algorithm but you answer higher time complexities than that, e.g. $O(n^4)$ – which technically also upperbounds $O(n^3)$, you will get wrong answer for this question.

B Analysis (15 marks)

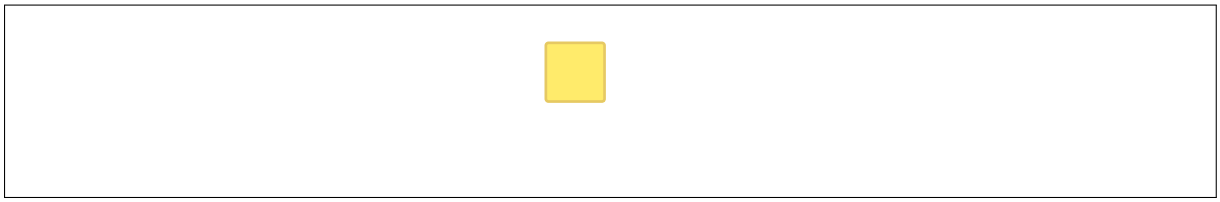
Prove (the statement is correct) or disprove (the statement is wrong) the statements below.

1. This C++ code runs in worst case time complexity of $O(n^2)$ and this analysis is tight.

```
int counter = 0;
for (int i = n; i >= 1; i--)
  for (int j = 1; j <= n/i; j++)
    counter++;
```



2. (Randomized) Quick sort is the *best* sorting algorithm to sort *any* set of n integers.



3. (Singly) Linked List is a *better* data structure to implement the Stack ADT `push(key)` and `last-key = pop()` than using C++ STL vector (i.e. a resizeable array).

4. (Singly) Linked List is a *better* data structure to implement the Queue ADT `enqueue(key)` and `earliest-key = dequeue()` than using C++ STL vector (i.e. a resizeable array).

5. The smallest element in a Binary *Max* Heap is always located at the bottommost, rightmost leaf, i.e. at index n of a compact array A that describes the Binary *Max* Heap.

Section B Marks = _____

C Technical/Algorithm Interview Questions (55 marks)

C.1 Anagram Checker (15 marks)

Given two strings A and B, both have the same length of n characters ($1 \leq n \leq 100\,000$), write the best algorithm that you can think of to determine whether A and B are anagram. Two equal length strings A and B are anagram if the characters of A can be formed by rearranging (permuting) the characters of the other string B. Analyze the worst case time complexity of your algorithm.

For example, A = "IVLE" and B = "EVIL" are anagram (from B, swap 'E' and 'I' then swap 'E' with 'L', then you get A), but A = "IVLE" and B = "LIFE" are not anagram[2]

A skeleton C++ code has been written for you below. Please complete it. If you are not sure of the required C++ syntax, you can write your answer in pseudo-code for slightly lesser marks.

```
#include <bits/stdc++.h>
using namespace std;
int main() {
  string A, B;
  cin >> A >> B;
  // sample test cases:
  // A = "IVLE"; B = "EVIL"; // should answer "Yes"
  // A = "IVLE"; B = "LIFE"; // should answer "No"

  return 0;
}
```

[2]If you have not seen the announcement at IVLE, here you go: "NUS has embarked on a year-long project to re-design and re-develop IVLE. In view of this, it is timely that a new and more relevant name be given to NUS Learning Management System...".

C.2 Linked List Partition (20 marks)

Given an implementation of unsorted *Singly Linked List* L of n distinct integers ($3 \leq n \leq 100\,000$), partition its content around a pivot value p, which is the value of the head element of L. You are **constrained to just use the given implementation below** to do this task. Analyze the worst case time complexity of your algorithm.

For example, if L contains this list: $5 \rightarrow 8 \rightarrow 6 \rightarrow 1 \rightarrow 7 \rightarrow 2$, you *may* produce *Lmod*(ified) that contains $2 \rightarrow 1 \rightarrow 5$ (the pivot p) $\rightarrow 8 \rightarrow 6 \rightarrow 7$ (any other valid partition will be accepted).

A skeleton C++ code has been written for you below. This is *an extension* of what has been shown in Lab Demo D03. Please complete and/or modify it as you see fit. If you are not sure of the required C++ syntax, you can write your answer in pseudo-code for slightly lesser marks.

```
#include <bits/stdc++.h>
using namespace std;
class SLL {
  struct Vertex {
    int item;
    Vertex *next;
  };
private:
  Vertex *head, *tail;
public:
  SLL() {
    head = tail = NULL;
  }
  void InsertAtHead(int value) {
    Vertex *vtx = new Vertex();
    vtx->item = value;
    vtx->next = head;
    if (head == NULL) tail = vtx; // first insertion
    head = vtx;
  }
  void InsertAfterTail(int value) {
    if (head == NULL)
      InsertAtHead(value); // special case
    else {
      Vertex *vtx = new Vertex();
      vtx->item = value;
      tail->next = vtx;
      tail = vtx;
    }
  }
```

```
  void PrintList() {
    for (Vertex *vtx = head; vtx != NULL; vtx = vtx->next)
      cout << vtx->item << " ";
    cout << endl;
  }
  SLL Partition() {
    SLL result; // do the required operations here









    return result;
  }
};
int main() {
  SLL L, Lmod;
  L.InsertAfterTail(5); L.InsertAfterTail(8); L.InsertAfterTail(6);
  L.InsertAfterTail(1); L.InsertAfterTail(7); L.InsertAfterTail(2);
  Lmod = L.Partition();
  L.PrintList();    // should remain 5->8->6->1->7->2
  Lmod.PrintList(); // [smaller than 5] -> 5 -> [bigger than 5]
  return 0;
}
```

C.3 Stack with 'findMax()' Operation (20 marks)

Assuming that we are dealing with a stack of (positive) integers, let's define one new operation of a standard Stack like the one that we discuss in class. This new operation is 'findMax()' that simply returns the current maximum integer currently inside the stack (or -1 if the stack is currently empty). Show how you are going to implement this new operation in $O(1)$ while maintaining that the standard push, top, pop operations of a stack all remains $O(1)$ too.

For example, if stack S contains : 5 (top) $\rightarrow$ 8 $\rightarrow$ 6 $\rightarrow$ 1 $\rightarrow$ 7 $\rightarrow$ 2, then a call of 'findMax()' should report 8. If we pop the topmost element 5, 'findMax()' should still report 8. However if we then pop the next topmost element 8, then 'findMax()' should now report 7.

A skeleton C++ code has been written for you below. Please complete and/or modify it as you see fit. If you are not sure of the required C++ syntax, you can write your answer in pseudo-code for slightly lesser marks.

```
#include <bits/stdc++.h>
using namespace std;
class StackRememberMax {
private:



public:
  StackRememberMax() {



  }
  void push(int value) { // put value at the top of Stack









  }
  int top() { // return the current top integer of Stack



  }
```

```
  int pop() { // pop the topmost integer from the stack









  }
  int findMax() { // report the max integer currently in this stack









  }
};
int main() {
  StackRememberMax S;
  cout << S.findMax() << endl; // should be -1
  S.push(2); S.push(7); S.push(1);
  S.push(6); S.push(8); S.push(5);
  cout << S.top() << endl; // should be 5
  cout << S.findMax() << endl; // should be 8
  S.pop(); // pop 5 out
  cout << S.top() << endl; // should be 8 now
  cout << S.findMax() << endl; // should still be 8
  S.pop(); // pop 8 out
  cout << S.top() << endl; // should be 6 now
  cout << S.findMax() << endl; // should now be 7
}
```

Section C Marks = _____ + _____ + _____ = _____

D Application (20 marks)

Given a list L containing n positive integers ($1 \leq n \leq 100\,000$), you are requested to execute a stream of two commands: 'R' (reverse the entire list L) and 'D' (drop the *first* integer currently in the list L). There can be up to 100 000 commands too. If any of the 'D' command is performed on a currently empty list, output a message "Error" and stop the program right away. If there is no error, print the final content of the list L.

For example, if the list L contains: $\{5, 8, 6, 1, 7, 2\}$, then commands "RDRDD" will first reverse L (we have $\{2, 7, 1, 6, 8, 5\}$), drop 2 (we have $\{7, 1, 6, 8, 5\}$), reverse the list again (we have $\{5, 8, 6, 1, 7\}$), drop 5 (we have ($\{8, 6, 1, 7\}$), and finally drop 8 (we have ($\{6, 1, 7\}$ and this is the final content of list L that we will print out).

This time, we do not give any skeleton C++ code. You are free to use whatever data structure(s) and/or algorithm(s) that you have learned in class. You are free to determine the Input/Output format that simplifies your program. If you are not sure of the required C++ syntax, you can write your answer in pseudo-code for slightly lesser marks. You can use page 10 for more writing space, if necessary. As usual, analyze the time complexity of your algorithm.

You can continue writing your answer here:

– End of this Paper, All the Best –