Simón Fraser University  
Computing Science 225  
Fall 2019 
Midterm Examination 2

Time: 50 minutes

Circle the campus you attend most often: Burnaby  Surrey  Vancouver

Last/Family Name (please, print): ____________________________

First Name (please, print): _________________________________

Student Number: _________________________________________

Signature: ________________________________________________

Instructor: Anne Lavergne

This examination has 7 pages.

- No books, cheat sheets, calculators, computers, cell phones, or other materials may be used.
- Read each question carefully before answering it.
- List any assumptions you make when answering a question.
- ADT means Abstract Data Type.
- BST means Binary Search Tree.
- Always comment your code.
- The marks for each question are given in [ ]. Use this to manage your time:
  ➔ 1 mark corresponds to 1 minute of work.
  ➔ Do not spend more time on a question than the number of marks assigned to it.

<table>
<thead>
<tr>
<th>#</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part 1</td>
<td></td>
</tr>
<tr>
<td>Q. 1 to 12</td>
<td>/ 24</td>
</tr>
<tr>
<td>Part 2</td>
<td></td>
</tr>
<tr>
<td>Q. 1</td>
<td>/ 4</td>
</tr>
<tr>
<td>Part 2</td>
<td></td>
</tr>
<tr>
<td>Q. 2</td>
<td>/ 8</td>
</tr>
<tr>
<td>Part 2</td>
<td></td>
</tr>
<tr>
<td>Q. 3</td>
<td>/ 8</td>
</tr>
<tr>
<td>Part 2</td>
<td></td>
</tr>
<tr>
<td>Q. 4</td>
<td>/ 6</td>
</tr>
<tr>
<td>Total</td>
<td>/ 50</td>
</tr>
</tbody>
</table>

Good luck!
Part 1 – [2 marks each – No part marks!] Answer each question in its box.

1. In our Assignment 3, we were introduced to the exception throwing mechanism of C++. List all the files used in this assignment in which we would be expecting to find try/catch block statements?

   BankSimApp.cpp
   PriorityQueue.h
   (Queue/BinaryHeap/PriorityQueue Test Drivers – but these files are not officially part of Assignment 3 – Bank Simulation as they as testing tools)

2. Imagine a List ADT class implemented as follows:

   ```cpp
   class List {
       private:
           unsigned int arraySize;
           Patient * elements;
       public:
           /* Public method declarations here. */
           /* There are no public attributes. */
   }
   
   Using the Big O notation, express the time efficiency of determining the number of elements stored into an object of this List ADT class.

   O(n)

   Since there is no “elementCount” data member, in order to determining the number of elements stored into an object of this List ADT class, one would have to traverse the List object, counting each element.

3. Consider a Binary Search Tree ADT class. Using the Big O notation, express the space efficiency of the worst case scenario of its insertion if this operation is implemented recursively.

   O(n)

   The worst case scenario of a BST is a degenerate BST (long linked list).

   Therefore, the worst case scenario of its insertion operation is O(n) in which one would have to go all the way down to the leaf level in order to insert an element.

   If this insertion is recursively implemented (called at each level of the BST), there will be a stack frame (stack memory) associated with each call made to this recursive insertion method (given to each execution of this method) -> O(n) stack frames.
So, the space efficiency of the worst case scenario of the BST’s insertion, if this operation is implemented recursively, is O(n).

4. How many nodes does a full binary tree contain if it has M leaves? Express your answer using M. You cannot use H (height) nor L (level).

\[ 2M - 1 = (M - 1) + M \quad \text{where } M \text{ is the number of leaves (nodes at level } H \text{) in a full tree of height } H \text{ and } (M - 1) \text{ is the number of nodes in a full tree up of height } H, \text{ from the root (level 1) to level } H - 1 \]

5. What is the time efficiency (expressed using the Big O notation) of the copy constructor of the Stack ADT class?

O(n) since each element in the original Stack object is copied onto the second Stack object and there are n element in the original Stack.

6. In our Assignment 3, what did an object of our Queue ADT class simulate?

A bank line-up (or bank queue)

7. Using the description of complete tree given in our lectures, draw all possible complete binary trees of height H = 3. You do not have to populate your binary trees.

Total: 4 complete binary trees of height H = 3

8. Circle all the nodes that are out of balance in this binary search tree:
9. List the result of traversing the above binary search tree A in pre-order, printing the search key value of each element as you are traversing the tree.

10  8  5  3  4  6  15  11  20  16

10. Consider the binary search tree A depicted in Question 8 above, list two of its elements that are adjacent to each other and that are located in the right subtree of the root (not including the root).

15 and 20  or  15 and 11  or  20 and 16

11. Consider the binary search tree A depicted in Question 8 above, draw the resulting binary search tree once you have deleted the element with the search key value 5 using its predecessor.

![Binary Search Tree Diagram]

12. Imagine you are logged into one of the CSIL (Ubuntu) computers and you have opened a Terminal window. At the command line, you cd (change directory) into your Assignment 3/Queue subdirectory (folder) in which you have created your Queue ADT class (template). Using the makefile you have downloaded from the Assignment 3 Part 1 and saved in this folder, you compile your Queue along with its test driver QueueTestDriver.cpp and execute your test driver to confirm that your Queue ADT class works just fine. Then, you execute the command make clean at the command line.

At this point, you execute the command ls > listing.txt followed by cat < listing.txt. What has the last command printed on the computer monitor screen? Write it all out below:

Makefile
Queue.h
QueueTestDriver.cpp

(EmptyDataCollectionException.cpp and EmptyDataCollectionException.h
Event.cpp and Event.h
listing.txt)
Part 2 – Answer the following questions in the space provided.

1. [4 marks – No part marks!] Insert the following elements **L U X E M B O R G** (in the given order) in an binary search tree and draw the resulting binary search tree.

![Binary Search Tree Diagram]

2. a) [4 marks - No part marks!] Insert the element with the search key value **8** into the binary heap **A** below (represented as an array) and draw the resulting binary heap (represented as an array as well) to its right-hand side.

<table>
<thead>
<tr>
<th>Binary Heap A</th>
<th>Resulting Binary Heap (array)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 5 6 4 3 5</td>
<td>8 5 7 4 3 5 6</td>
</tr>
</tbody>
</table>

b) [4 marks - No part marks!] Remove twice from the binary heap **B** below (represented as an array) and draw the resulting binary heap (represented as an array as well) to its right-hand side.

<table>
<thead>
<tr>
<th>Binary Heap B</th>
<th>Resulting Binary Heap (array)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 3 2 4 5 9 6 11 15</td>
<td>3 4 6 11 5 9 15</td>
</tr>
</tbody>
</table>
3. [8 marks] Write a recursive algorithm (in pseudocode) that finds the maximum element (the element with the largest search key value) in a binary search tree and returns a pointer to it. State any assumptions you are making while answering this question.

Assumption:

```cpp
class Node {
private:
    ElementType element;
    Node * left;
    Node * right;

    ...}
```

Assumption:

```cpp
class BST {
private:
    Node * root;
    int elementCount;

    ...}
```

```cpp
max( Node * someNode ) // the first call to max -> max(root)

    // 2 marks
    if someNode == NULL i.e., the tree is empty
      return NULL

    // 3 marks
    if someNode has no right subtree -> someNode -> right == NULL
      return someNode->element

    // 3 marks
    else
      max( someNode->right )
```
4. [6 marks] A **deque** is a data collection that allows for efficient insertion (inserting 1 element in \(O(1)\)) and removal of elements (removing one element in \(O(1)\)) on either of its ends. Hence you can enqueue elements at the front or at the end of such data collection and you can dequeue elements from its front or from its end.

For example, consider the following deque (depicted here as a sequence of elements’ search key values):

**Example - Deque A:** front -> B R T W Q <- end

I obtain the search key value **Q** if I dequeue an element from its end and I obtain the following modified deque:

**Example - Deque B:** front -> F B R T W <- end

if I enqueue an element with search key value **F** at its front.

Design the underlying data structure you would use in order to implement a Deque ADT class while satisfying the time efficiency of the four operations discussed above. To represent your design, draw your underlying data structure and populate it using the data in the last example above, i.e., Example - Deque B. Make sure you label all parts of your underlying data structure well and include all necessary details in order for a reader to understand how the four Deque operations discussed above can be performed in \(O(1)\).

As a reader, I can see that the four Deque operations discussed above can be performed in \(O(1)\):

- the enqueueAtFront(…) and dequeueAtFront(…) done at the green arrow and
- the enqueueAtBack(…) and dequeueAtBack(…) done at the blue arrow