CMSC 221: Data Structures
Fall 2017
Exam 1

Name: ______________ Key ______________ Section: ______________

Instructions:

1. There are test questions on the front and the back of each sheet.

2. This is a closed book exam. Do not use any notes, books, or neighbors except your one page, two side, handwritten, cheat sheet which MUST be turned in with your exam.

3. Show your work. Partial credit will be given. Grading will be based on correctness and clarity.

4. You have 75 minutes to complete the exam. Watch your time appropriately. You should take about 15 minutes per question section.

Integrity: The University of Richmond’s Honor Code is “We, the students of the University of Richmond, shall promote and uphold a community of integrity and trust.” Upon accepting admission to University of Richmond, a student immediately assumes a commitment to uphold the Honor Code, to accept responsibility for learning, and to follow the philosophy and rules of the Honor System. Students will be required to state their commitment on examinations, research papers, and other academic work. Ignorance of the rules does not exclude any member of the Richmond community from the requirements or the processes of the Honor System.

I agree to uphold this commitment and produce original work in this exam, i.e., I will not cheat nor will I consciously let anyone cheat.

Signature: ______________________________

DO NOT BEGIN THE EXAM UNTIL INSTRUCTED TO DO SO. GOOD LUCK!

<table>
<thead>
<tr>
<th>Question</th>
<th>Points received</th>
<th>Points possible</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Bonus</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>
1. (20 points) **True or False.**

**Circle** the correct answer for each question.

(a) **True** or **False**: Assuming equivalent theoretical complexities, array-based implementations of data structures and algorithms often outperform linked structure-based implementations in practice because of the memory hierarchy in computer organization.

(b) **True** or **False**: The functions of the stack ADT all run in $O(1)$ time.

(c) **True** or **False**: The functions of an array-based implementation of the stack ADT can all be implemented to run in amortized $O(1)$ time.

(d) **True** or **False**: The functions of a doubly-linked-list-based implementation of the queue ADT can all be implemented to run in $O(n)$ time.

(e) **True** or **False**: The functions of an singly-linked-list-based implementation of the deque ADT can all be implemented to run in $O(1)$ time.

(f) **True** or **False**: The Positional List ADT provides access to elements using indices, a mechanism for describing the location of an element in a sequence.

(g) **True** or **False**: The tree shown above has size 9 and height 4, and has 1 root, 5 internal nodes, and 4 leaves (external nodes).

(h) **True** or **False**: A pre-order traversal of the tree shown above could visit the nodes in the order: I, J, K, L, M, N, O, P, and Q.

(i) **True** or **False**: A pre-order traversal of the tree shown above could visit the nodes in the order: I, K, Q, O, J, M, N, L, and P.

(j) **True** or **False**: A post-order traversal of the tree shown above could visit the nodes in the order: P, Q, M, J, O, L, N, K and I.
2. (20 points) **Short Answer.**

Provide the best answer you can for all the questions below.

(a) One fundamental storage technique, a linked structure, encodes adjacency using pointers. If only next adjacency is stored, the structure is a(n) **Singly linked list**. A special case of this, **Circularly linked lists**, occurs when the first and last elements are marked adjacent.

(b) Express the function \( f(n) = 100,000 \log n + 0.000032 \cdot 2^n \) as concisely as possible in terms of Big-Oh (\( O() \)) notation \( O(2^n) \).

Express the function \( f(n) = 80 + \log^2 n + 23n\sqrt{n} + 45n \log n \) as concisely as possible in terms of Big-Oh (\( O() \)) notation \( O(n\sqrt{n}) \).

(c) If a Deque (double-ended queue) ADT is implemented using a doubly-linked list, then in the worst case an `removeLast()` operation takes time \( O(1) \). If the Deque is implemented using a singly-linked list, then in the best case an `removeLast()` operation takes time \( O(n) \) for a Deque of size \( n \).

(d) If a Positional List ADT is implemented using a expandable circular array, then an `addFirst(e)` operation takes in the best case \( O(1) \) time, in the average case \( O(n) \) time, and in the worst case \( O(n) \) time, assuming implementation uses an incremental-strategy for resizing an array.

(e) If the List ADT is implemented using a doubly linked-list, then in the worst case a `get(i)` operation takes time \( O(n) \) and in the average case an `add(i, e)` operation takes time \( O(n) \).
3. (10 points) **Tree Properties.**

(a) (4 points) Draw a general tree of size 5 with the minimum number of levels. Draw a general tree of size 4 with the maximum number of levels.

(b) (2 points) What is the minimum number of levels $l$ of a proper binary tree in terms of its size $n$ and $l$, i.e., write $l$ as a function of $n$? *Hint: It is an inequality.*

$$l \geq \log(n + 1)$$

(c) (2 points) What is the maximum number of levels $l$ of an improper binary tree in terms of its size $n$, i.e., write $l$ as a function of $n$? *Hint: It is an inequality.*

$$l \leq n$$

(d) (2 points) A complete binary tree has the maximum number of nodes in each level. What is the size $n$ of a complete binary tree with $l$ levels, i.e., write $n$ as a function of $l$?

$$n = 2^l - 1$$
4. (20 points) **Lists.** Show how to implement portions of the List ADT using a single Queue named \( Q \).

(a) (10 points) **Add.** Describe your algorithm for add in pseudocode. What is the time complexity of this operation if the queue is implemented with a non-circular expandable array using the incremental strategy of resizing? Please provide a short high-level intuition as justification.

**Algorithm** \( \text{ADD}(i, e) \)

**Input:** Index \( i \), element \( e \)

1: \textbf{for} \( j \leftarrow 1 \) \textbf{to} \( i \) \textbf{do}
2: \( \text{Q.\textsc{enqueue}}(\text{Q.\textsc{dequeue}}()) \)
3: \( \text{Q.\textsc{enqueue}}(e) \)
4: \textbf{for} \( j \leftarrow i + 1 \) \textbf{to} \( \text{Q.\textsc{size}}() - 1 \) \textbf{do}
5: \( \text{Q.\textsc{enqueue}}(\text{Q.\textsc{dequeue}}()) \)

\( O(n^2) \)

Either \textsc{enqueue}() or \textsc{dequeue}() will need to take \( O(n) \) time in a non-circular queue, thus \( n \) of these operations will take \( O(n^2) \) time.

(b) (10 points) **Remove.** Describe your algorithm for remove in pseudocode. What is the time complexity of this operation if the queue is implemented with a singly-linked list? Please provide a short high-level intuition as justification.

**Algorithm** \( \text{REMOVE}(i) \)

**Input:** Index \( i \)

1: \textbf{for} \( j \leftarrow 1 \) \textbf{to} \( i \) \textbf{do}
2: \( \text{Q.\textsc{enqueue}}(\text{Q.\textsc{dequeue}}()) \)
3: \( \text{Q.\textsc{dequeue}}(e) \)
4: \textbf{for} \( j \leftarrow i + 1 \) \textbf{to} \( \text{Q.\textsc{size}}() \) \textbf{do}
5: \( \text{Q.\textsc{enqueue}}(\text{Q.\textsc{dequeue}}()) \)

\( O(n) \),

All queue operations can be implemented in \( O(1) \) time, and \( O(n) \) operations occur.
5. (30 points) **Tree Traversal.**

Describe an efficient algorithm for determining the successor position following a given input position of a binary tree. A successor will be defined as the next node in an inorder traversal ordering. For example, in the tree below: the successor of $A$ is $C$, $D$ is $B$, and $E$ is $A$. If there is no successor, return null.

```
          A
         / \  
        B   C
       / \  /  
      D   E
```

(a) (10 points) **Pseudocode.** Write your algorithm. Name the algorithm `INORDERNEXT` and assume this will become a new function of the binary tree ADT, i.e., it has access to all other ADT functions without having a specific tree.

**Algorithm** `INORDERNEXT(p)`

**Input:** Position $p$

**Output:** Successor position

1. if `RIGHT(p) ≠ ∅` then
2.   Position $s ← RIGHT(p)$
3.   while `LEFT(s) ≠ ∅` do
4.     $s ← LEFT(s)$
5.   return $s$
6. else
7.   Position $n ← p$
8.   Position $m ← PARENT(n)$
9.   while $m ≠ ∅ ∧ n = RIGHT(m)$ do
10.    $n ← m$
11.   $m ← PARENT(n)$
12. return $m$

(b) (5 points) **Correctness.** Explain your pseudocode above and describe why it computes the successor correctly.

Essentially, the inorder successor is defined as the leftmost node of the right child or the first parent that has an unexplored right child. The algorithm is broken in these two cases, one that follows left children and the other that follows ancestors.
(c) (5 points) **Time Complexity: Best case.** Quantify the best case time complexity for your algorithm. Be sure to justify your complexity.

\( O(1) \). Let the tree be implemented with a linked structure. One best case is such that the right child of the node exists and has no children. In this case, only a constant number of operations are performed, thus the algorithm is best case \( O(1) \) time.

(d) (5 points) **Time Complexity: Worst case.** Quantify the worst case time complexity for your algorithm. Be sure to justify your complexity.

\( O(h) \), where \( h \) is the height of the tree. Let the tree be implemented with a linked structure. In the worst case, the full height of the tree must be traversed to find the next node, either leftmost or all ancestors.

(e) (5 points) **Time Complexity: Average case.** Quantify the average case time complexity for your algorithm. Be sure to justify your complexity.

\( O(1) \). Let the tree be implemented with a linked structure. If we sum up the length of each possible step of \( n \) iterations of inorder next, we will yield \( 2n \). This is because each edge of the tree is traversed twice, once down to the leftmost node, and once up the ancestors. Thus, on average each call to inorder next will take \( O(1) \) time.
6. (10 points) **Bonus.** Let $T$ be a binary tree with $n$ positions, and, for any position $p$ in $T$, let $d_p$ denote the depth of $p$ in $T$. The *distance* between two positions $p$ and $q$ in $T$ is $d_p + d_q - 2 \cdot d_a$, where $a$ is the lowest common ancestor (LCA) of $p$ and $q$. The *diameter* of $T$ is the maximum distance between two positions in $T$. Describe an efficient algorithm for finding the diameter of $T$. What is the running time of your algorithm? Justify your complexity.

**Algorithm** DIA(METER($T$))

**Input:** Binary tree $T$

**Output:** Diameter of the tree

1: return DIA(METER($T$, $T$.root))

**Algorithm** DIA(METER($T$, $n$))

**Input:** Binary tree $T$ and position $n$

**Output:** Diameter of the subtree rooted at $n$

1: if $T$.isExternal($n$) then
2:   $n$.height ← 0
3:   return 0
4: else
5:   Position $l$ ← $T$.left($n$)
6:   Position $r$ ← $T$.right($n$)
7:   $d_l$ ← DIA(METER($T$, $l$))
8:   $d_r$ ← DIA(METER($T$, $r$))
9:   $n$.height ← $1 + \text{MAX}(l$.height $+$ $r$.height$)$
10:  return MAX($d_l$, $d_r$, $l$.height $+$ $r$.height $+$ 1)

This algorithm runs in $O(n)$ time, as it is a post-order traversal that does $O(1)$ work for each visit.