The University of Saskatchewan

Saskatoon, Canada
Department of Computer Science

CMPT 280– Intermediate Data Structures and Algoirthms

Assignment 2

Date Due: Friday, July 14, 2017, 11:45pm

Total Marks: 76

NOTE: Due to the nature of this assignment, you may NOT WORK IN PAIRS. Unfortunately, there can be no extreme programming (pilot/copilot) on this assignment.

1 Submission Instructions

Assignments must be submitted using Moodle.

Responses to written (non-programming) questions must be submitted in a PDF file, plain text file (.txt), Rich Text file (.rtf), or MS Word's .doc or .docx files. Digital images of handwritten pages are also acceptable, provided that they are **clearly** legible, and in a common format such as JPEG or

PNG.

Programs must be written in Java.

If you are using Eclipse (or similar development environment), do not submit the workspace (project). Hand in only those files identified in Section 5. Export your .java source files from the workspace and submit only the .java files.

No late assignments will be accepted. See the course syllabus for the full late assignment policy for this class.

2 Background

2.1 Timing Analysis

Questions 1 through 10 in Section 3 concern timing analysis. These questions are not programming questions and should be submitted in one of the acceptable document formats listed above.

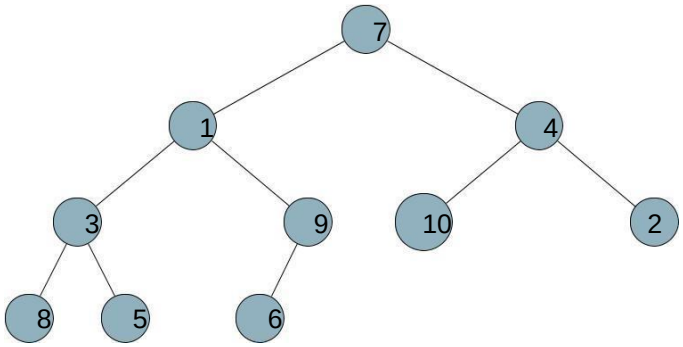
2.2 Arrayed Trees

Question 11 is about a bounded binary tree implementation. You should remember binary trees from CMPT 115 (or similar course) – they are trees in which each node has at most two children. What you probably didn't know is that binary trees can be stored using an array, rather than a linked structure. In such an array, the contents of the root node are stored in offset 1 of the array (offset 0 is unused). The contents of the children of the node whose contents are stored at offset i are stored at offset 2i and 2i + 1, respectively. Thus, the left child of the root is at offset 2 1 = 2, the right child of the root is at offset 2 1 + 1 = 3, the left child of the left child of the root is at offset 2 2 = 4, and so on. The parent of the node whose contents are at offset i, is at offset i/2 (integer division). Thus, the parent of node at offset 7 is at offset 3.

Example 1
Here is the array representation of a tree storing the elements 1 through 10, in no particular order. The array

-	7	1	4	3	9	10	2	8	5	6

is a representation of the tree:



Note that we do not allow any unused cells in the array between used ones. All the nodes in the array are stored contiguously. This means that we can represent only a particular subset of binary trees with this representation. Namely, it is the set of trees where all levels except possibly the last level are complete (full) and the nodes in the last level are all as far to the left as possible. You might be thinking that this is too restrictive and not very useful because we can't represent **all** binary trees with this data structure. However, as we will see on future assignments, this array-based tree data structure is highly useful and efficient for implementing certain other important data structures.

Also note that if we read off the items from left to right in each level of the tree, starting from the top level, we get the items in the same order as they appear in the array (we will visit this again when studying breadth first search).

3 Your Tasks

Question 1 ():

Suppose the exact time required for an algorithm A is given by:

$$T_A(n) = 1234n + 19n^3 + 99 + 27n^{2.5}(log_2 n) + 42\sqrt{n}$$

(a) (2 points) Which of the following statements are true?

1. Algorithm A is $O(\log n)$
2. Algoirthm A is $O(n)$
3. Algoirthm A is $O(n^3)$
4. Algoirthm A is $O(2^n)$

(b) (1 point) Give the correct time complexity for A in big-Θ notation.

Question 2 ():

For each of the following functions, give the tightest upper bound chosen from among the usual simple functions listed in Section 4.5 of the course readings. Answers should be expressed in big-O notation.

(a) (1 point) $f_1(n) = 12n \log_2 n + n^2 \log_2 n + \frac{2^n}{4200}$

(b) (1 point) $f_3(n) = n^2 \log_2 n^2 + 8n^3 + \log_2(2^n)$

(c) (1 point) $f_2(n) = 4n^{0.5} + \frac{\log_2 n}{n} + 1234$

Question 3 ():

If possible, simplify the following expressions. Hint: See slide 11 of topic 4 of the lecture slides!

(a) (1 point) $O(n^2)$ + O (log n) + O (n log n)

(b) (1 point) $O(2^n)$ $O(n^2)$

(c) (1 point) 42O (n log n) + 18$O(n^3)$

(d) (1 point) O ($n^2 \log_2 n^2$) + O (m) (yes, that's an 'm', not a typo; note that m is independent of n)

Question 4 (5 points):

Consider the function f (n) = 2n3 + 5n2 + 42. Use the definition of big-O to prove that $f(n) \in O(n^3)$.

Question 5 (5 points):

Consider the function $g(n) = 12n^2 \log n^2 + 6n + 42$. Use the definition of big-O to prove that $g(n) \in O(n^2 \log n^2)$.

Question 6 (3 points):

Consider again the function $g(n) = 12n^2 \log n^2 + 6n + 42$. Use the definition of big-O to prove that $g(n)$ is **not** in $O(n)$.

Question 7 ():

Consider the following Java code fragment:

```
// Print out all ordered pairs of numbers between 1 and
    n for (i = 1; i <= n; i++) { for (j = 1; j <= n; j++)
    {
                  System .out. println ( i + ", " + j) ; }
    }
```

(a) (3 points) Determine the exact number of statements (i.e. the statement counting approach) that are executed when we run this code fragment as a function of n. Show all of your calculations.

(b) (1 point) Express the function you obtained in part a) in big-Θ notation.

Question 8 ():

Consider the following pseudocode:

```
Algorithm roundRobinTournament (a)
This algorithm generates the list of matches that must be
played in a round - robin pirate - dueling tournament (a tournament where each
pirate duels each other pirate exactly once).

a is an array of strings containing names of entrants into a tournament
       n = a.length for i = 0 to n-1 for j = i+1 to n-1
print a[i] + " duels " + a[j] + ", Yarrr!"
```

(a) (5 points) Determine the exact number of statements (i.e. use the statement counting approach that are executed by the above algorithm. Express your answer as a function of n. Show all your calculations.

(b) (1 point) Express the function you obtained in part a) in big-Θ notation.

Question 9 ():

Consider the following pseudocode:

```
Algorithm moveDown (a) a is an array of numbers
int i = 1
n = a.length
while (a[i] > a[2*i] || a[i] > a[2*i+1]) && 2*i+1 < n)
       if a[2*i] >= a[2*i +1]
              largest = 2*i
       else
              largest = 2*i + 1
       temp = a[i]
       a[i] = a[largest]
       a[largest] = temp
       i = largest
```

(a) (2 points) Determine the exact number of statements (i.e. use the statement counting approach) that are executed during one iteration of the while loop in the worst case. Your answer should be expressed in terms of *n* (the length of the array) Show all calculations.

(b) (5 points) Determine the exact number of times the while loop executes in the worst case.

(c) (3 points) Determine the exact number of statements executed in the worst case by the whole algorithm.

(d) (1 point) Identify an Active Operation

(e) (2 points) Determine the exact number of times the active operation is executed.

(f) (2 points) Express the answers to parts c) and e) in big-O notation.

Question 1 0 (28 points):

Your task is to write a Java class called `ArrayedBinaryTreeWithCursors280<I>` which extends and implements the abstract class `ArrayedBinaryTree280<I>` (provided in the `lib280-asn2.tree` package as

part of `lib280-asn2`). This week's lab will also talk more about array-based trees.

Some of the work of implementing `ArrayedBinaryTreeWithCursors280<I>` has already been done.

There are several methods in defined in `ArrayedBinaryTreeWithCursors280<I>` which are defined but not implemented; these are marked with `//TODO` comments. Note that `ArrayedBinaryTreeWithCursors280<I>` also implements the interfaces `Dict280<I>` and `Cursor280<I>`. There are several missing methods required by these interfaces that also needed to be implemented. The headers for these are not yet present in `ArrayedBinaryTreeWithCursors280<I>` – you need to add them. Until you do, the com-piler will complain on line 15 that there are unimplemented abstract methods inherited from the interfaces. The interfaces `Dict280<I>` and `Cursor280<I>` and their ancestors (yes, they have ancestor interfaces!) document what these methods are supposed to do.

You may not modify any of the existing code in the provided `ArrayedBinaryTreeWithCursors280.java` **file**, but you can add to it. **You may also not modify any other files within** `lib280-asn2`.

There is already a regression test included in `ArrayedBinaryTreeWithCursors280<I>`. Your completed implementation of the arrayed binary tree should pass the given regression test. If all the regression tests are successful, the only output should be: Regression test complete.

Hint: one of your first major decisions after adding the appropriate method headers for the inherited abstract methods will be to start implementing the insert method and decide where the new element

should be inserted. If you think about it, there's really only one place it can go...

Hint: The algorithm for deleting an element is to replace the element to be deleted by the right-most element in the bottom level of the tree, then delete the right-most element in the bottom level of the tree.

*Reminder: the elements of the items array (defined in the abstract class `ArrayedBinaryTree280<I>`) represent the nodes of the tree. You are storing the **contents** of nodes in the array. There is no node class. It is very important that the contents of the root are stored in offset 1 and we don't use cell 0 of the array, otherwise, the given formulae for finding the child or parent of a node at offset i will not work correctly.*

4 Files Provided

lib280-asn2: A copy of `lib280` which includes solutions to assignment 1, the `ArrayedBinaryTree280<I>` abstract class, and the partially completed `ArrayedBinaryTreeWithCursors280<I>` class for Question 11. The `ArrayedTree280<I>` interface can be found in the `lib280-asn2.tree` package.

5 What to Hand In

You must submit the following files:

Q1-10.doc/docx/rtf/pdf/txt - your answers to questions 1 through 10. Digital images of handwritten pages are also acceptable, provided that they are **clearly** legible.

ArrayedBinaryTreeWithCursors280.java - your arrayed binary tree implementation and regression test.