Questions and Exercises to work out and turn in:

Grading Guidelines:

* A right answer will get full credit when:

1. It is right (worth 25%)
2. It is right **AND** neatly presented making it easy and pleasant to read. (worth an **extra** 15%)
3. There is an **obvious and clear link** between 1) the information provided in the exercise and in class and 2) the final answer. A clear link is built by properly writing, justifying, and documenting an answer (worth an **extra** 60%).
4. Calculation mistakes will be minimally penalized (2 to 5% of full credit) while errors on units will be more heavily penalized.

You are welcome/encouraged to discuss exercises with other students or the instructor. But, ultimately, **personal** writing is expected.

* USE THIS FILE AS THE STARTING DOCUMENT YOU WILL TURN IN. **DO NOT DELETE ANYTHING FROM THIS FILE:** JUST **INSERT** YOUR ANSWERS.
* IF USING HAND WRITING (STRONGLY DISCOURAGED), **USE THIS FILE** BY CREATING SUFFICIENT SPACE AND WRITE IN YOUR ANSWERS.
* FAILING TO FOLLOW TURN IN DIRECTIONS /GUIDELINES WILL COST **A 30% PENALTY.**

Objectives of this assignment:

* to use and manipulate the concepts presented in this module
* to propose and write algorithms in pseudocode
* to analyze the time complexity of algorithms
* to analyze the space complexity of algorithms
* to learn autonomously new concepts

What you need to do:

Answer the questions and/or solve the exercises described below.

Questions (20 points):

The objective is to compare two data structures: a min-heap and a binary search tree*.* Determine the time complexity for *Search, Minimum, Maximum, Successor, Predecessor, Insert, and Delete* on a min-heap and a binary search tree, respectively. Use a table to present and to compare their time complexities. You do not need to show how you get/determine the time complexities. Just refer your sources.

|  |  |  |
| --- | --- | --- |
|  | Min-heap | Binary Search Tree |
| **Search** | **O(n)** | **O(h)** |
| **Minimum** | **O(1)** | **O(h)** |
| **Maximum** | **O(n)** | **O(h)** |
| **Successor** | **O(n)** | **O(h)** |
| **Predecessor** | **O(n)** | **O(h)** |
| **Insert** | **O(h) or O(log n)** | **O(h)** |
| **Delete** | **O(h) or O(log n)** | **O(h)** |

**Binary Search Tree:**

**According to slide 30 from M2 – The time complexities of all methods are to be O(h).**

**Min Heap:**

***Insert - Delete:* Page 151 says that Max-Heap-Insert runs in O(log n) time. Since Max-Heap and Min-Heap have the same time complexities, the time complexities of these methods can be mutually applied. Since a heap is a complete binary tree, it’s height is .**

***Minimum:* In a min-heap, the minimum is always the first node to be checked.**

***Maximum – Search – Successor – Predecessor:*** [**https://en.wikipedia.org/wiki/Binary\_heap**](https://en.wikipedia.org/wiki/Binary_heap) **- Since all four of these processes are basically searching through the entire heap, they all run in O(n)**

Exercise 1 (10 points)

Suppose that we have numbers between 1 and 1000 in a binary search tree, and we want to search for the number 393. Which of the following sequences could not be the sequence of nodes examined?

a. 32, 282, 431, 428, 360, 374, 427, 393.

b. 954, 250, 941, 274, 928, 288, 392, 393.

c. 955, 232, 941, 270, 942, 275, 393.

d. 32, 429, 417, 249, 296, 412, 411, 308, 393.

e. 965, 308, 377, 651, 329, 422, 388, 393.

C. cannot be the sequence of nodes examined. This is because of the property of binary search trees that all nodes on the left of a given node have a key less than that node. In this sequence, however, after 941 is visited, 270 is visited, meaning the left child of 941 was chosen. The right child of 270 is 942, but 942 > 941, which breaks the rule that all nodes in the left subtree must be less than the given node. Thus, c. cannot be the sequence of nodes examined.

The sequence E also could not be the sequence of nodes examined when searching for 393. This is because the value 329 would not appear in the right subtree of value 377. Since 329 is less than 377, it would appear only in the left subtree.

Exercise 2 (35 points) A Recursive TREE-MAXIMUM(x)

1. Write a **recursive** version of TREE-MAXIMUM(s).

RECURSIVE-TREE-MAXIMUM(x)

1. if x.right == NIL
2. return x
3. else
4. RECURSIVE-TREE-MAXIMUM(x.right)
5. Analyze the **time** complexity of recursive version and compare it with the **iterative** version’s.
   1. The time complexity measured in comparisons is 1 comparison per call, and then a possible recursive call to the function. For every recursive call, there is again 1 comparison, and the recursion happens only until the rightmost node is visited. This results in a maximum of h calls, where h is the height of the tree. The recursive version has a time complexity of O(h), which is the same as the iterative version.

Time complexity analysis of recursive algorithms is discussed on page 89 of the textbook.

1. Analyze the **space** complexity of recursive version and compare it with the **iterative** version’s.
   1. The space complexity of RECURSIVE-TREE-MAXIMUM(x) is O(h). This is because the stack has to keep track of every call to RECURSIVE-TREE-MAXIMUM(x) from beginning to end. Since the function is called at most h times, the stack has a limit of h calls to keep track of, making the space complexity O(h), as opposed to the iterative version, which has a space complexity of O(1).

Space complexity analysis is discussed on slide 35 from M5 of Algorithms I.

Exercise 3 (35 points) TREE-PREDECESSOR

1. Write the TREE-PREDECESSOR procedure

TREE-PREDECESSOR(x)

* + 1. if x.left ≠ NIL
    2. return TREE-MAXIMUM(x.left)
    3. y = x.p
    4. while y ≠ NIL and x == y.left
    5. x = y
    6. y = y.p
    7. return y

1. Analyze its time complexity.
   1. In the worst case, TREE-PREDECESSOR(x) will either start at the top of a tree and have to work all the way down the tree to find a predecessor, or it will start at the bottom of a tree and have to work it’s way all the way to the top. In both cases, the maximum that the function will have to travel is the height of the tree which is O(h), so it’s time complexity is O(h).
2. Analyze its space complexity.
   1. If x.left is not NIL, the function calls TREE-MAXIMUM(x.left), which has a space complexity of O(1) as long as the iterative version is used. If x.left is NIL, the function only requires the space to store nodes in x and y, meaning it still has a space complexity of O(1).

What you need to turn in:

* Electronic copy of this file (including your answers) (standalone). Submit the file as a Microsoft Word or PDF file.
* Recall that answers must be well written, documented, justified, and presented to get full credit.
* How this assignment will be graded:
* A right answer will get full credit when:
* It is right (worth 25%)
* It is right AND neatly presented making it easy and pleasant to read. (worth 15%)
* There is an obvious and clear link between 1) the information provided in the exercise and in class and 2) the final answer. A clear link is built by properly writing, justifying, and documenting an answer (worth 60%).
* Calculation mistakes will be minimally penalized (2 to 5% of full credit) while errors on units will be more heavily penalized.
* You are welcome/encouraged to discuss exercises with other students or the instructor. But, ultimately, personal writing is expected.