**Data Structures**

**Fall 2019**

**Practice Exam 2**

**Section 0 – Student Information**

s0\_name\_and\_id.py

**Open the file and make sure you type your last name, first name, and UTEP ID**

**DON’T FORGET THIS SECTION**

**DON’T FORGET THIS SECTION**

**DON’T FORGET THIS SECTION**

**DON’T FORGET THIS SECTION**

**DON’T FORGET THIS SECTION**

**DON’T FORGET THIS SECTION**

**Section 1 - Multiple Choice Questions**

s1\_multiple\_choice.py

**Problem 1 (4 points):** Which of the following trees is not a binary search tree?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Option 0: | Option 1: | Option 2: | Option 3: | Option 4:  All of them are binary search trees |

**Problem 2 (4 points):** Consider the following B-Tree.

A close up of a piece of paper

Description automatically generated

What would *print(self.root.children[2].keys[0])* print?

Option 0: 3

Option 1: 1

Option 2: 4

Option 3: 7

Option 4: 10

Option 5: None of the above

**Problem 3 (4 points):** Consider the following heap.

A picture containing object, watch, clock

Description automatically generated

What would *print(self.tree[5])* print?

Option 0: 8

Option 1: 5

Option 2: 6

Option 3: 2

Option 4: 3

Option 5: 1

Option 6: 4

Option 7: None of the above

**Problem 4 (4 points):**Show the configuration of an initially empty hash table of size 5 that solves collisions by chaining after performing the following sequence of operations.

ht = HashTable(5)

ht.insert(17)

ht.insert(18)

ht.insert(10)

ht.insert(8)

ht.insert(1)

|  |  |  |  |
| --- | --- | --- | --- |
| Option 0:    0:| |-->[10]  1:| |-->[1]  2:| |-->[17]  3:| |-->[8, 18]  4:| |-->[] | Option 1:  0:| |-->[10]  1:| |-->[1]  2:| |-->[17]  3:| |-->[18, 8]  4:| |-->[] | Option 2:  0:| |-->[]  1:| |-->[1]  2:| |-->[17]  3:| |-->[8, 18]  4:| |-->[10] | Option 3:  Answer is not listed |

**Problem 5 (4 points):** Consider an initially empty max-heap. Show the result of performing the following sequence of insertions: 1, 3, 5, 2, 7, 0

|  |  |  |  |
| --- | --- | --- | --- |
| Option 0:    [7, 3, 5, 0, 2, 1] | Option 1:  [7, 5, 3, 1, 2, 0] | Option 2:  [7, 2, 5, 1, 0, 3] | Option 3:  Answer is not listed |

**Problem 6 (4 points):** What is the running time of the *search* operation in a binary search tree?

|  |  |  |  |
| --- | --- | --- | --- |
| Option 0:    O(1) | Option 1:  O(log n) | Option 2:  O(n) | Option 3:  O(n log n) |

**Section 2 – Binary Search Trees**

s2\_binary\_tree\_section.py

**Problem 7 (6 points):** Complete the implementation of the \_*height* method. This method computes the height of a binary search tree (The parameter *node* represents the root of the [sub]tree)*.*  The height of an empty tree is -1. The height of a tree with only one node is 0.

**Problem 8 (6 points):** Complete the implementation of the *\_num\_nodes\_at\_depth* method. This method returns the number of nodes at depth *d* of a binary search tree (The parameter *node* represents the root of the [sub]tree)*.*

**Problem 9 (6 points):** Complete the implementation of the *\_min\_val* method. This method returns the smallest value in a binary search tree. (The parameter *node* represents the root of the [sub]tree)*.*

**Problem 10 (6 points):** Complete the implementation of the *\_max\_val\_at\_depth* method. This method returns the largest value in a binary search tree at depth *d*. (The parameter *node* represents the root of the [sub]tree)*.*

**Section 3 – B-Trees**

s3\_btrees\_section.py

**Problem 11 (6 points):** Complete the implementation of the \_*height* method. This method computes the height of a b-tree (The parameter *node* represents the root of the [sub]tree)*.*  The height of an empty tree is -1. The height of a tree with only one node is 0.

**Problem 12 (6 points):** Complete the implementation of the *\_num\_nodes\_at\_depth* method. This method returns the number of nodes at depth *d* of a b-tree (The parameter *node* represents the root of the [sub]tree)*.*

**Problem 13 (6 points):** Complete the implementation of the *\_max\_val\_at\_depth* method. This method returns the largest value in a b- tree at depth *d*. (The parameter *node* represents the root of the [sub]tree)*.*

**Problem 14 (6 points):** Complete the implementation of the *\_search* method. This method returns the node where *k* is, or None if *k* is not in the tree. (The parameter *node* represents the root of the [sub]tree)*.*

**Section 4 - Hash Tables**

s4\_hash\_tables\_section.py

**Problem 15 (6 points):** Complete the implementation of the *insert* method. This method inserts *k* into the table if and only if it is not already in it.

**Problem 16 (6 points):** Complete the implementation of the *contains* method. This method returns True if and only if *k* is in the table; False, otherwise.

**Problem 17 (6 points):** Complete the implementation of the *get\_longest\_list* method. This method should return the size of the longest list in the hash table.

**Problem 18 (6 points):** Complete the implementation of the *reset\_table* method. This method should remove all items in the table.

**Section 5 - Heaps**

s5\_heaps\_section.py

**Problem 19 (6 points):** Complete the implementation of the *get\_max\_sibling\_gap* method. The max sibling gap in a max-heap is defined as the maximum difference between the value of a right sibling and its left sibling.

**Problem 20 (6 points):** Complete the implementation of the *is\_valid* method. The purpose of this method is to verify that the contents of *self.tree* represent a valid max-heap.

**Problem 21 (6 points):** Complete the implementation of the *is\_a\_node\_equal\_to\_its\_parent* method. This method is to return True if and only if there is at least one element in the heap that is equal to its parent.

**Problem 22 (6 points):** Complete the implementation of the method *print\_path*. This method receives an integer *i* as input and prints all the elements in the path from index *i* to the root of the heap.

*Total numbers of points: 120 (graded out of 100)*