Data Structures - CSCI-SHU 210

Solution

Final Practice Exam

First Name	
Last Name	
NetID	
Section	

Details

- **Duration:** 75 Minutes
- Computers, calculators, phones, textbooks, or notebooks are not allowed

Instructions

- Fill the table with your information as it appears on Brightspace and Albert.
- Show your NYU ID card to the proctor before signing the attendance sheet.
- You can use the blank space to make notes for yourself. Your notes are not evaluated.
- Turn off and place your mobile devices with your belongings at the front of the room.
- Please do not damage the exam paper.
- Do not discuss this quiz with anyone until the test is handed back.
- If you need to ask questions, please raise your hand and wait for the instructor.
- Stay seated until the instructor has collected all exams.

Please sign the statement below:

I, ___________________, confirm that I am a student in this class and I will complete this exam without accessing any unauthorized information during the exam. I have read and understood the exam instructions above.

Multiple Choices Questions

After answering all knowledge questions, transfer your solution letter to the table below. Only one solution is correct for each answer option. Please mark your solution clearly:

Questions 1.1 to 1.10

	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	1.10
Answer	C	A	B	A	D	C	C	B	A	C
Points										

Question 1.1. - Tree Terminology

If a binary tree has *n* nodes, how many edges does it have?

- ☐ A: $log_2 n$
- ☐ B: n
- ☐ C: $n - 1$
- ☐ D: none of the listed

Question 1.2. - Tree Terminology

Suppose a binary search tree has *n* nodes. The minimum height of the binary search tree is $Ceil(log_2(n + 1) - 1)$

- ☐ A: The above statement is True
- ☐ B: The above statement is False

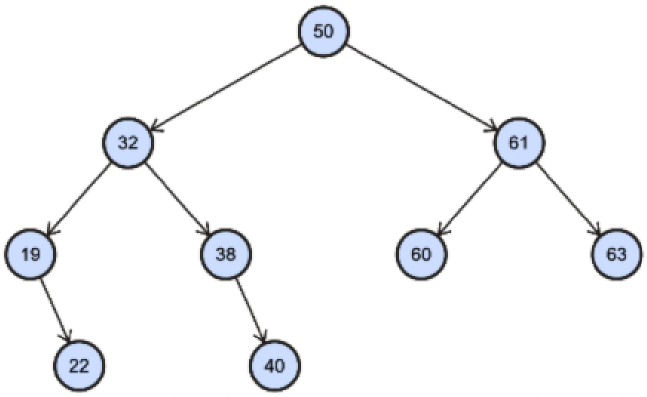
Question 1.3. - Hashtable

The worst-case complexity for getting the length of a hashtable is $O(n)$, based on the Hashtable data structure from our lecture and recitation.

- ☐ A: The above statement is True
- ☐ B: The above statement is False

Question 1.4. - Tree Traversal

Consider the following tree and select the correct statement below:



- ☐ A: The element before 32 in an in-order traversal is: 22
- ☐ B: The element before 50 in an in-order traversal is: 61
- ☐ C: The element after 19 in an in-order traversal is: 32
- ☐ D: The element after 40 in an in-order traversal is: 60

Question 1.5. - Function Mystery

Consider the following code snippet:

```
def mystery(lis):
  S = [0] * len(lis)
  for i in range(len(lis)-1):
    T = lis[0 : i + 1]
    S[i] = sum(T) * (i + 2)
  return S
```

Suppose `len(lis) = n`. The worst-case big-O analysis of this code is:

- ☐ A: $O(log_2n)$
- ☐ B: $O(log_2n * log_2n)$
- ☐ C: $O(n)$
- ☐ D: $O(n^2)$

Question 1.6. - Time Complexity

What is the time complexity of the following function, assuming n = len(string):

```
def is_special(string):
    return special_helper(string, 0)


def special_helper(string, index):
    if index >= len(string) // 2:
        return True
    if string[index] != string[-1 - index]:
        return False
    return special_helper(string, index + 1)
```

- ☐ A: $O(n^2)$
- ☐ B: $O(log_2 n)$
- ☐ C: $O(n)$
- ☐ D: $O(1)$

Question 1.7. - Tree Insert

The following items are inserted into a binary search tree in this order:

`12, 8, 5, 3, 9, 13, 14, 52, 1, 2, 55, 4`

Which node is the deepest?

- ☐ A: 9
- ☐ B: 55
- ☐ C: 2
- ☐ D: 1

Question 1.8. - Heap Remove

What is the worst-case time complexity of removing the minimum item in a min-heap of size *n* implemented using a linked binary tree:

☐ A: $O(n * log_2 n)$

☐ B: $O(log_2 n)$

☐ C: $O(n)$

☐ D: $O(1)$

Question 1.9. - Heap Correct Statement

Select the correct statement:

☐ A: The last node in a heap is the deepest rightmost leaf node in a complete tree

☐ B: The last node in a heap is the deepest leftmost leaf node in a complete tree

☐ C: The last node in a heap is the deepest rightmost non-leaf node in a full tree

☐ D: The last node in a heap is the deepest leftmost non-leaf node in a full tree

Question 1.10. - Hashtable

For a given hashtable using the hash function h(x) = x mod 17 and linear probing, as seen in the picture below:

	18	19	88	1	22	90	41	59	77	44				14	32
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

How many probes will be required to insert 8?

☐ A: 4

☐ B: 5

☐ C: 3

☐ D: 2

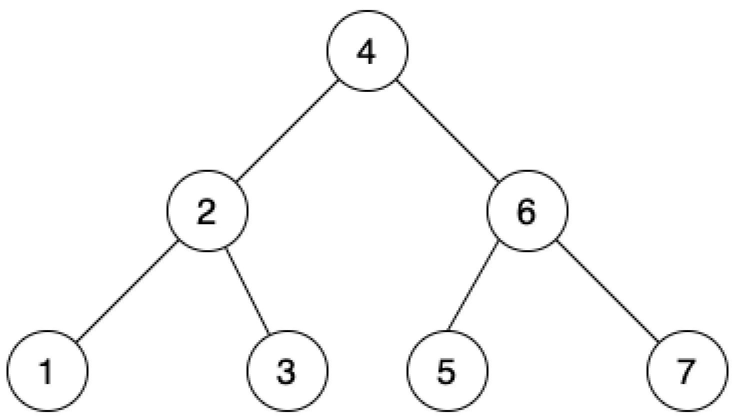
2. Programming Question

Please try to write your programming solution as clearly as possible.

Question 2.1. - Search Tree from Array

Given a sorted list of numbers, convert the list to a perfect Binary Search Tree in array representation, as seen in the figure. Remember: A perfect tree is a complete tree with the maximum number of leaf nodes.

Binary Search Tree tree representation for the sorted array [1, 2, 3, 4, 5, 6, 7]:



Example 1:

```
def main():
    ls = [1, 2, 3, 4, 5, 6, 7]
    res = convert(ls)
    print(res)  # Should print: [4, 2, 6, 1, 3, 5, 7]
```

Requirements:

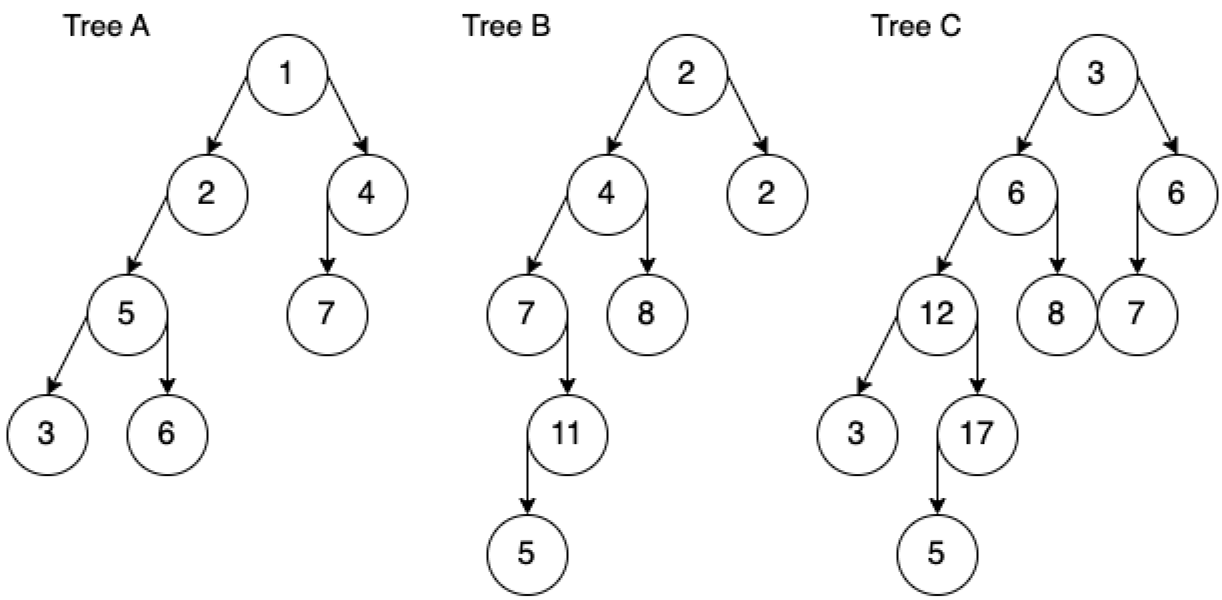
- You are not allowed to use global variables.
- You can not use any library or third-party tools.
- You are allowed to use the Math libraries log function.
- Your solution has to be O(n) time complexity.
- Your solution has to be O(n) space complexity.
- You are not allowed to create a linked binary search tree.

```
def convert(ls):
```

```
def convert(ls):
    res = [None] * len(ls)
    convert_helper(0, ls, res, 0, len(ls) - 1)
    return res

def convert_helper(result_idx, input, output, left, right):
    if left > right:
        return

    mid = (left + right) // 2
    output[result_idx] = input[mid]

    convert_helper(result_idx*2+1, input, output, left, mid - 1)
    convert_helper(result_idx*2+2, input, output, mid + 1, right)
```

Question 2.2. - Merging Trees

Given two binary trees A and B, of size *n* and *m,* respectively, write a function to merge them into a tree C. The function should perform the sum of the elements in two nodes if the corresponding node exists in both trees (e.g., the root node in tree A and tree B in the figure). Your function should merge all nodes in the output tree C. Your function has to create new nodes and return the root node of a new tree C. You cannot modify the values or structures in the input trees.



The above picture shows the merge of trees A and B, with the resulting tree C.

Requirements:

- You are not allowed to use global variables.
- You cannot use any library or third-party tools.
- Your solution has to be in O(n+m) time complexity
- Your solution has to be O(n+m) space complex
- You are not allowed to modify the input trees.
- Your function has to return the root node of a new tree.

Example 1:

```
treeA = TreeWithoutParent(1)
treeA._left = TreeWithoutParent(2)
treeA._right = TreeWithoutParent(4)
treeA._left._left = TreeWithoutParent(5)
treeA._right._left = TreeWithoutParent(7)
treeA._left._left._left = TreeWithoutParent(3)
treeA._left._left._right = TreeWithoutParent(6)

treeB = TreeWithoutParent(2)
treeB._left = TreeWithoutParent(4)
treeB._right = TreeWithoutParent(2)
treeB._left._left = TreeWithoutParent(7)
treeB._left._right = TreeWithoutParent(8)
treeB._left._left._right = TreeWithoutParent(11)
treeB._left._left._right._left = TreeWithoutParent(5)

treeC = treeA + treeB
```

```python
def __add__(self, other):
```

```python
def _add_helper(self, t1, t2):

    # case 1: the nodes exist in tree A and tree B
    if t1 and t2:
        t3 = TreeWithoutParent(t1._element + t2._element)
        t3._left = self._add_helper(t1._left, t2._left)
        t3._right = self._add_helper(t1._right, t2._right)

    # case 2: the node only exists in tree A
    elif t1:
        t3 = TreeWithoutParent(t1._element)
        t3._left = self._add_helper(t1._left, None)
        t3._right = self._add_helper(t1._right, None)

    # case 3: the node only exists in tree B
    elif t2:
        t3 = TreeWithoutParent(t2._element)
        t3._left = self._add_helper(None, t2._left)
        t3._right = self._add_helper(None, t2._right)

    # base case: there is no node
    else:
        return None

    return t3

def __add__(self, other):
    return self._add_helper(self, other)
```

Question 2.3.- Binary Tree Layers

Given an array representation of a binary tree, return a list of all node values at a given depth. For example, if depth d is equal to 0, you should return a list containing only the element of the root node. If the depth is equal to 1, you should return a list containing the elements of the direct children of the root node, and so on. Skip any non-existent nodes. Notice that the value d can be any non-negative integer.

Example:

```
def main():
      lissy = [4, 2, 6, None, 1, 3, 5, 7]

      print(get_layer(lissy, 0)) # Expect: [4]
      print(get_layer(lissy, 1)) # Expect: [2, 6]
      print(get_layer(lissy, 2)) # Expect: [1, 3, 5]
      print(get_layer(lissy, 3)) # Expect: [7]
      print(get_layer(lissy, 4)) # Expect: []
```

Requirements:

- You are not allowed to use global variables.
- You are allowed to use helper functions.
- Your solution has to be recursive. Loops are not allowed.
- Do not use any library or third-party tools.
- Your solution must have a running time of at most *O(2^d)*.
- You can not create a tree of nodes to solve this problem.
- Your output list can not contain *None* values.

```
def get_layer(lissy, depth):
        return get_layer_helper(lissy, depth, 0, 0)

        def get_layer_helper(lissy, d, cur_root_idx, i):
          if cur_root_idx >= len(lissy):
            return []
          if i == d:
            value = lissy[cur_root_idx]
            return [] if value is None else [value]
          return get_layer_helper(lissy, d, cur_root_idx*2+1, i+1) +
        get_layer_helper(lissy, d, cur_root_idx*2+2, i+1)
```

Appendix 1 - TreeWithoutParent Implementation

```
class TreeWithoutParent:
  def __init__(self, element, left=None, right=None):
    self._element = element
    self._left = left
    self._right = right

  def __add__(self, other):
    # You need to implement this function.
    # Here, this function is to merge self & other into a new tree
    # Return type: an instance of TreeWithoutParent
    pass
```

You can tear off this page and use it as scratch paper.