**CSCI-SHU 210 Data Structures**

**Final Exam**

**Tuesday 05/19/2020**

**Name:** \_\_Zhenming Wang\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**NetID:** \_\_\_\_\_zw1806\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* Do not open this test booklet until you are told to do so.
* Put your ID out, so proctors can check it during the exam. They may also ask you

to sign in.

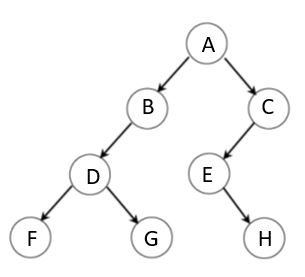
* The exam is closed book, closed notes, closed codes. No electronic devices may be used except scientific calculators. In your computer/laptop, only 4 programs may be running – Microsoft word, one python IDLE (not required), one browser with only one tab for https://newclasses.nyu.edu, Zoom.
* You have ONE HOUR and HALF to do this exam.
* There are 13 questions all together, with total 100 points.
* Write your answers neatly in the provided space. There is one extra page for overflow at the back. If you use this, leave a note so I'll know to look there.
* Read every question completely before answering it.
* For any questions related to runtime, consider an asymptotic analysis (tightest

bound).

* You do not have to do error checking. Assume all inputs to your functions are as described
* DO NOT COMMUNICATE with any other students during the exam!!
* Good luck!!!

**Question 1 (8 points):**

1. Fill the nodes of the following tree with the numbers 17, 1, 11, 3, 20, 7, 25, 9, so that the resulting tree would be a **binary search tree (BST)**. (3 points)



Fill the following table with your solutions:

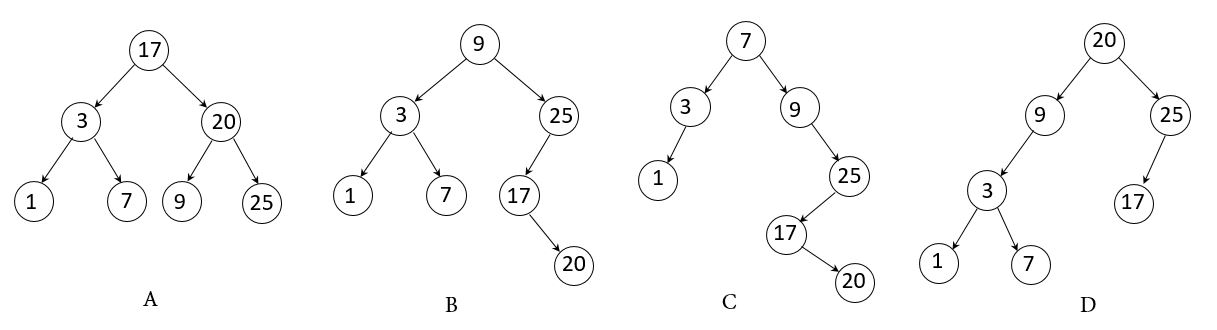
|  |  |  |  |
| --- | --- | --- | --- |
|  | Numbers |  | Numbers |
| A | 11 | E | 17 |
| B | 9 | F | 1 |
| C | 25 | G | 7 |
| D | 3 | H | 20 |

1. Let **BST** be an empty binary search tree. Give a sequence of keys to insert to **BST**, so that the resulting tree would contain 17, 1, 11, 3, 20, 7, 25, 9 and it would have the structure of the tree given in section (a). (2 points)

|  |  |
| --- | --- |
| 1st insert: \_\_\_\_\_\_\_11\_\_\_\_\_  2nd insert: \_\_\_\_\_\_\_\_9\_\_\_\_  3rd insert: \_\_\_\_\_25\_\_\_  4th insert: \_\_\_\_\_\_3\_\_\_\_\_\_ | 5th insert: \_\_\_\_17\_\_\_\_\_\_\_\_  6th insert: \_\_\_\_1\_\_\_\_\_\_\_\_  7th insert: \_\_\_\_7\_\_\_\_\_\_\_\_  8th insert: \_\_\_20\_\_\_\_ |

1. What would happen to the **BST in (a)** if you **deleted 11**. (No credit if your answer to (a) is wrong). Choose one of the following trees. (3 points)

**Your answer**: \_\_\_A\_\_\_\_



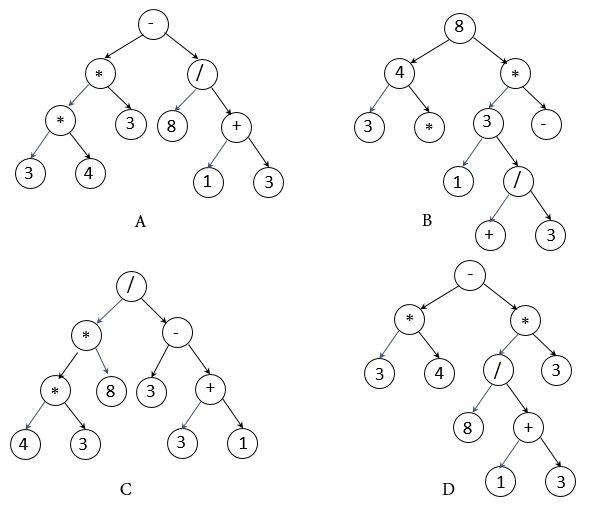
**Question 2 (3 points):**

Given a postfix **arithmetic expression :**

**3 4 \* 8 1 3 + / 3 \* -**

Which one of the following trees has the **postorder** traversal of the above expression?

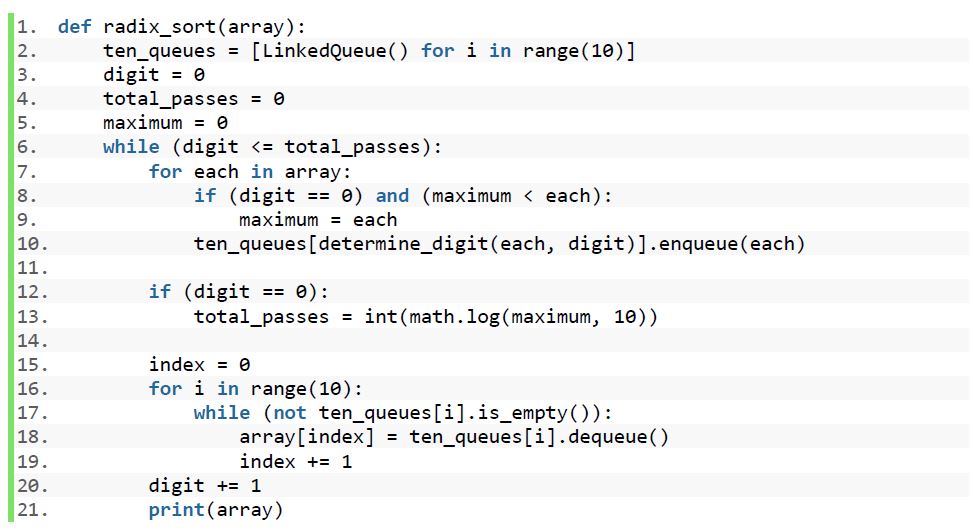
**Your answer: \_\_\_D\_\_\_\_**

****

**Question 3 (4 points):**

Suppose you run **decimal radix sort** on the following array:

[281, 363, 5825, 690, 587, 57, 933, 16, 927, 9].

****

Write down the outputs from this print (**at line 21**) after **3rd** pass? Fill the following table with your answer, each cell holds one number.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 9 | 16 | 57 | 281 | 363 | 387 | 690 | 5825 | 927 | 933 |

**Question 4 (10 points)**:

Give a **python** **implementation** of the following function:

**def** order\_switches(lissy):

Given lissy, a list of string values('OFF'/'ON'), this function orders the list lissy such that all the 'OFF' values come before all the 'ON' values.

For example, if lissy = ['ON','OFF','ON','OFF','ON','OFF','ON','OFF'], after the call order\_switches(lissy), lissy will be ['OFF', 'OFF', 'OFF', 'OFF', 'ON', 'ON', 'ON', 'ON'].

**Implementation Requirements**:

1. No points if your program simply count the number of 'ON' or 'OFF' values and then assign the correct number of 'OFF' and 'ON' values in the list.
2. **Your program must run in** *O****(n)* time.**
3. **Your program must be in place, and may use** *O****(1)* extra space.**

**def** order\_switches(lissy):

**”””**

**@lissy: a list of string values ('OFF'/'ON').**

**@return: nothing.**

**”””**

**# Your code**

prev = None ##by setting a prev, we could know where the latest OFF is.

curr = lissy[0]

for i in range(len(lissy)):

if lissy[i] == “OFF”:

if lissy[i-1] == “ON”:

##check if we have prev

if prev != None:

lissy[prev], lissy[i] = lissy[i], lissy[prev]

else:

lissy[i],lissy[i-1] =

lissy[i-1],lissy[i]

prev = i ###remembering last time we do swap

**Question 5 (10 points):**

Consider the following simple binary tree class:

1. **class** SimpleTree:
2. **def** \_\_init\_\_(self, element, left=None, right=None, parent=None):
3. self.\_element = element
4. self.\_left = left
5. self.\_right = right
6. self.\_parent = parent
8. **def** \_\_str\_\_(self):
9. **return** str(self.\_element)
10. **def** backlink(root):
11. **# Write your code in the next page (page #7).**

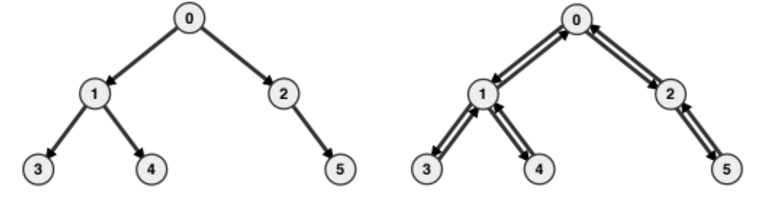
Given the root of a binary tree that has parent reference of all nodes not set up, your task is to implement a function **backlink(root) that will link all the parent reference of all nodes of this binary tree correctly**. Here parameter **root is the root node of a binary tree (class type: SimpleTree).**

After calling **backlink(root)** function, every node references to its parent node correctly. Note that root has no parent.

**Implementation requirements:**

1. You should assume that **parameter** **root** in the **backlink(root)** call, is the root node of a binary tree object **(Class Type: SimpleTree)**.
2. **Initially, all self.\_parent is set to None for the given Tree.**
3. The root node has no parent node, so its parent reference remains **None**.

**Example: before calling backlink(root) after calling backlink(root)**



**def backlink(root):**

**”””**

**@root: root of a SimpleTree, initially call will start with the root.**

**Given the root of a binary tree that has parent reference not set up, link all the parent reference of all nodes of this binary tree correctly.**

**@return: Nothing**

**”””**

**# Your code**

backlink\_helper(root)

###helper function

def backlink\_helper (node):

if node is None:

return

else:

#check if it has children

if node.\_left is not None:

node.\_left.\_parent = node

if node.\_right is not None:

node.\_right.\_parent = node

#recursion

backlink\_helper(node.\_left)

backlink\_helper(node.\_right)

**Filename: double\_linked\_list.py (will be used for Question No. 6)**

1. **class** DoubleLinkedList:
2. """A base class providing a doubly linked list representation."""
4. #-------------------------- nested \_Node class --------------------------
5. # nested \_Node class
6. **class** \_Node:
7. """Lightweight, nonpublic class for storing a doubly linked node."""
8. \_\_slots\_\_ = '\_element', '\_prev', '\_next'            # streamline memory
9. **def** \_\_init\_\_(self, element, prev, next):      # initialize node's fields
10. self.\_element = element                     # user's element
11. self.\_prev = prev                           # previous node reference
12. self.\_next = next                           # next node reference
14. #-------------------------- list constructor --------------------------
16. **def** \_\_init\_\_(self):
17. """Create an empty list."""
18. self.\_header = self.\_Node(None, None, None)
19. self.\_trailer = self.\_Node(None, None, None)
20. self.\_header.\_next = self.\_trailer            # trailer is after header
21. self.\_trailer.\_prev = self.\_header            # header is before trailer
22. self.\_size = 0                                # number of elements
24. #-------------------------- public accessors --------------------------
26. **def** \_\_len\_\_(self):
27. """Return the number of elements in the list."""
28. **return** self.\_size
30. **def** is\_empty(self):
31. """Return True if list is empty."""
32. **return** self.\_size == 0
34. **def** add\_second(self, e):
35. """Add the element after first non-sentinel node."""
36. if len(self) == 0:
37. print(“Can’t insert second in an empty list.”)
38. return
39. else:
40. # To do for question 6. Write your code in page #10

**Question 6 (10 points):**

In this question, we will implement **insert\_second(self, element)**, a new method in

the **DoubleLinkedList** class (source code is provided in the double\_linked\_list.py file page #8). When called, it adds **element** to the **linkedlist after it’s first non-sentinel node**, or prints an error message if the **linkedlist** doesn’t have any elements (therefore there is no second position).

**As for example,**

suppose we create a **DoubleLinkedList** object named **list1** which has the following 2 elements.

Header

**35**

**50**

Trailer

list1: Header <-->35<-->50<-->Trailer

After calling **list1.insert\_second(99)** function, **list1** will be as follows:

Trailer

Header

**35**

**99**

**50**

list1: Header<-->35<-->99<-->50<-->Trailer

**Implementation requirements:**

1. You are **not allowed to use any of the other methods** of the **DoubleLinkedList** class. You need to update the links of the nodes in the list to reflect the insertion.
2. This **DoubleLinkedList** class has two dummy sentinel nodes **header** and **trailer**.
3. **Your program must run in** *O****(1)* time.**
4. **Your program must be in place.**

**def insert\_second(**self**, element):**

**”””**

**@element: an element which will be added in self.**

**Add the element after first non-sentinel node of self.**

**@return: nothing**

**”””**

**if len(self) == 0:**

**print(“Can’t insert second in an empty list.”)**

**return**

**else:**

**# Your code**

#find currently first node

first = self.\_header.\_next

old\_second = first.\_next

#to insert, we must change old connection

new\_node = self.\_Node (element)

old\_second.\_prev = new\_node

first.\_next = new\_Node

new\_Node.\_prev = first

new\_Node.\_next = old\_second

**Filename: single\_linked\_list.py (will be used for Question No. 7)**

1. **class** Empty(Exception):
2. """Error attempting to access an element from an empty container."""
3. **pass**
5. **class** SingleLinkedList:
6. #-------------------------- nested \_Node class --------------------------
7. **class** \_Node:
9. **def** \_\_init\_\_(self, element, next = None):      # initialize node's fields
10. self.\_element = element               # reference to user's element
11. self.\_next = next                     # reference to next node
13. #-------------------------- Single Linked List methods ------------------
14. **def** \_\_init\_\_(self):
15. """Create an empty LinkedList."""
16. self.\_head = None                       # reference to the head node
17. self.\_size = 0                          # number of elements in the list
19. **def** \_\_len\_\_(self):
20. """Return the number of elements in the LinkedList."""
21. **return** self.\_size
23. **def** is\_empty(self):
24. """Return True if the LinkedList is empty."""
25. **return** self.\_size == 0
27. **def** insertAtFirst(self, e):
28. """Add element e to the start of the LinkedList."""
29. new\_Node = self.\_Node(e)
30. new\_Node.\_next = self.\_head
31. self.\_head = new\_Node
32. self.\_size += 1
34. **def** deleteFirst(self):
35. """Remove and return the first element from the LinkedList.
37. Raise Empty exception if the Linked list is empty.
38. """
39. **if** self.is\_empty():
40. **raise** Empty('LinkedList is empty')
41. answer = self.\_head.\_element
42. self.\_head = self.\_head.\_next           # bypass the former top node
43. self.\_size -= 1
44. **return** answer
46. **def** \_\_str\_\_(self):
47. result = []
48. curNode = self.\_head
49. **while** (curNode **is** **not** None):
50. result.append(str(curNode.\_element) + "-->")
51. curNode = curNode.\_next
52. result.append("None")
53. **return** "".join(result)
54. **def** delete\_every\_other\_node\_from\_SLL(self):
55. """delete every other node or alternating nodes from Single LinkedList"""
56. # To do for question 7. Write your code in page #13

**Question 7 (12 points):**

Give an implementation of the following method in the **SingleLinkedList** class (source code is provided in the single\_linked\_list.py file (Page #11)):

**def** delete\_every\_other\_node\_from\_SLL(self):

This method will delete every other nodes or alternating nodes starting from the 2nd node from the calling **SingleLinkedList** object. After calling this method, calling **SingleLinkedList** object will have only the alternating nodes.

**As for example:**

Suppose we create a **SingleLinkedList** object named **list1** which has the following 4 elements.

**\_head**

**3**

**2**

**1**

**9**

**∅**

list1: 3-->2-->1-->9-->None

After calling **list1.delete\_every\_other\_node\_from\_SLL ()** function, **list1** will be as follows:

**\_head**

**∅**

**1**

**3**

list1: 3-->1-->None

**Another Example:**

If **list1, a SingleLinkedList object,** contains the following 5 elements.

**\_head**

**∅**

**6**

**9**

**1**

**2**

**3**

list1: 3-->2-->1-->9-->6-->None

After calling **list1.delete\_every\_other\_node\_from\_SLL ()** function, **list1** will be as follows:

**\_head**

**1**

**6**

**∅**

**3**

list1: 3-->1-->6-->None

**Implementation requirements:**

1. Your implementation should use ***O(1)*** **additional memory.**
2. This **SingleLinkedList** Class has only \_head reference and \_size reference variable.
3. \_head reference will point to the first element in the single linked List.
4. Your implementation’s **runtime** should be ***O(n)*.**
5. Your implementation should be in place.

**def delete\_every\_other\_node\_from\_SLL(**self**):**

**”””**

**Delete every other node from self, that is, the calling SingleLinkedList object.**

**@return: Nothing, modify self in place.**

**”””**

**# Your code**

curr = self.\_head

prev = None

delete = False

#flip delete every other node

while curr is not None:

#check if delete is True

if delete == True:

next = curr.\_next ##save next reference

prev.\_next = curr.\_next

curr.\_next.\_prev = prev

curr = next

delete = False ##switch off delete mode

else:

curr = curr.\_next ##point to next element

prev = curr

delete = True

##delete is now True, to delete next round

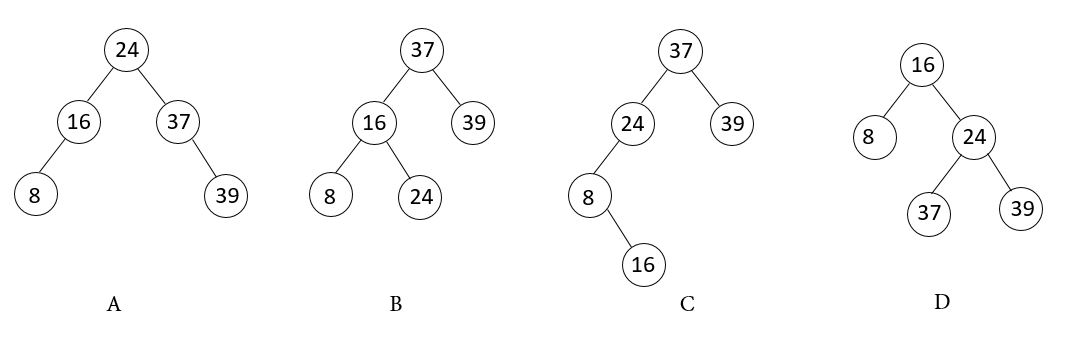
## Question 8 (6 pts):

Consider the following input **sequence** in order (first 39, then 37, then 24 and so on):

**39 37 24 8 16**

a. Which one of the following trees is the **AVL** tree of that sequence, **T1** that it produces.

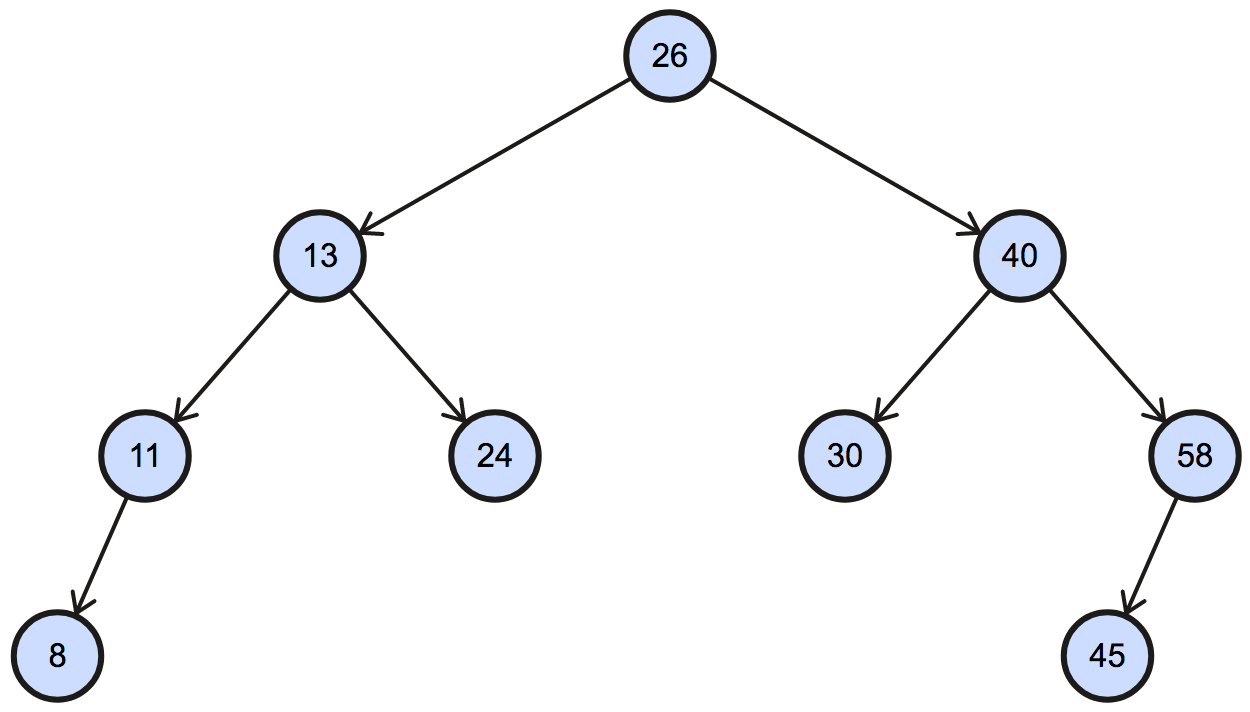
**Your answer: \_\_\_\_B\_\_\_\_\_**

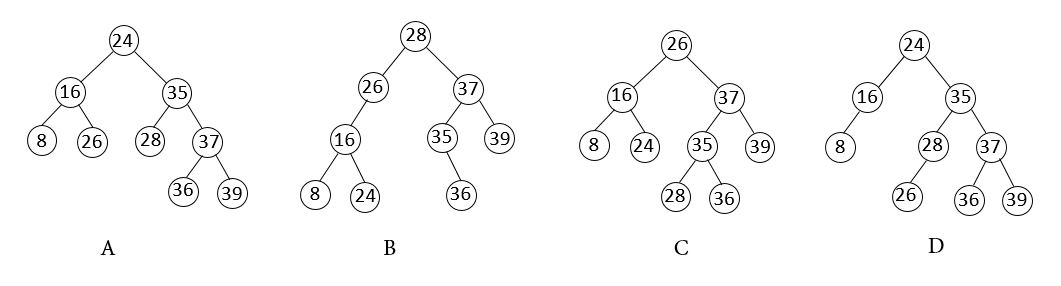


b. What would happen to the **AVL tree** after inserting the following **sequence** in order into **T1** :

**26 28 35 36**

**Your answer :\_\_\_D\_\_\_\_\_\_**





**Question 9 (6 points):**

In this question, you will insert and delete items to/from a *N=11* length open addressing hash table, **(hash function is: *h(k) = k mod 11*),** and **quadratic probing** will be used for resolving collisions.

1. We start with the following insertions sequence in order: **9, 1, 17, 8, 22, 3, 28, 19 and 18**
2. We then delete: **28 and 1**
3. Finally, we insert: **16**

Draw the table resulted after executing the operations above.

**Notes:**

1. **You do not need to rehash(resize) the table**.

2. You should mark **\_AVL** sentinel slots after deletions.

3. You should show three tables.

a.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 22 | 1 | 19 | 3 |  | 18 | 28 | 17 | 8 | 9 |  |

b.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 22 | \_AVL | 19 | 3 |  | 18 | \_AVL | 17 | 8 | 9 |  |

c.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 22 | \_AVL | 19 | 3 |  | 18 | 16 | 17 | 8 | 9 |  |

**Question 10 (10 points)**:

Magic Pair: If sum of two numbers from a given list is equal to a random target number, we will define that two numbers as Magic Pair for this question.

Give a **python** **implementation** of the following function:

**def** check\_magic\_pair\_exists(lissy, random\_target):

Given lissy, a list of distinct positive integers and a random\_target,a randomly generated positive integer, the above function returns True if sum of two numbers from lissy is equal to random\_target. If no such magic pair exists, the above function will return False.

For example,

>>> lissy = [15, 2, 3, 8, 1, 7, 10, 23], random\_target = 18 #randomly generated using abs(random.randint(1,2\*\*32)

>>> check\_magic\_pair\_exists (lissy, random\_target)

True

>>> lissy = [ 90, 100, 101, 70, 80, 103], random\_target = 140 #randomly generated using abs(random.randint(1, 2\*\*32)

>>> check\_magic\_pair\_exists (lissy, random\_target)

False

**Implementation Requirements**:

1. **If your program’s expected runtime is more than** O**(n), you will lose 8 points.**

import random

**def** check\_magic\_pair\_exists (lissy, random\_target):

**“““**

@lissy: list of distinct positive integers

@random\_target: a randomly generated positive integer

@return: True/False

”””

**# Your code**

record = {}

for i in lissy:

record[i] = ‘marked’ ##traverse list once to create a dictionary

for i in lissy:

if i < random\_target:

gap = random\_target-i #if we want to make to target using i

if record.get[gap] != None:

return True

return False

**Question 11 (4 points):**

Given the following maximum heap:



We are executing the following two operations:

• We start with calling remove\_max

• We then insert 63

For each one of the operations above, draw the resulting heap **in its array/list representation**.

After calling remove\_max:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 71 | 55 | 57 | 44 | 52 | 56 | 39 | 35 | 22 | 28 | 26 |  |

After inserting 63:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 71 | 55 | 63 | 44 | 52 | 57 | 39 | 35 | 22 | 28 | 26 | 56 |

**Provided just for reference.**

1. **class** MinHeapPriorityQueue:
2. """A min-oriented priority queue implemented with a binary heap."""
4. **class** \_Item:
5. """Lightweight composite to store priority queue items."""
6. \_\_slots\_\_ = '\_key', '\_value'
8. **def** \_\_init\_\_(self, k, v):
9. self.\_key = k
10. self.\_value = v
12. **def** \_\_lt\_\_(self, other):
13. **return** self.\_key < other.\_key    # compare items based on their keys
15. #------------------------------ nonpublic behaviors ------------------------
16. **def** \_parent(self, j):
17. **return** (j-1) // 2
19. **def** \_left(self, j):
20. **return** 2\*j + 1
22. **def** \_right(self, j):
23. **return** 2\*j + 2
25. **def** \_has\_left(self, j):
26. **return** self.\_left(j) < len(self.\_data)     # index beyond end of list?
28. **def** \_has\_right(self, j):
29. **return** self.\_right(j) < len(self.\_data)    # index beyond end of list?
31. **def** \_swap(self, i, j):
32. """Swap the elements at indices i and j of array."""
33. self.\_data[i], self.\_data[j] = self.\_data[j], self.\_data[i]
35. **def** \_upheap(self, j):
36. parent = self.\_parent(j)
37. **if** j > 0 **and** self.\_data[j] < self.\_data[parent]:
38. self.\_swap(j, parent)
39. self.\_upheap(parent)             # recur at position of parent
41. **def** \_downheap(self, j):
42. **if** self.\_has\_left(j):
43. left = self.\_left(j)
44. small\_child = left               # although right may be smaller
45. **if** self.\_has\_right(j):
46. right = self.\_right(j)
47. **if** self.\_data[right] < self.\_data[left]:
48. small\_child = right
49. **if** self.\_data[small\_child] < self.\_data[j]:
50. self.\_swap(j, small\_child)
51. self.\_downheap(small\_child)    # recur at position of small child
53. #------------------------------ public behaviors ---------------------------
54. **def** \_\_init\_\_(self):
55. """Create a new empty Priority Queue."""
56. self.\_data = []
58. **def** \_\_len\_\_(self):
59. """Return the number of items in the priority queue."""
60. **return** len(self.\_data)
62. **def** is\_empty(self):
63. **return** len(self) == 0
65. **def** add(self, key, value):
66. """Add a key-value pair to the priority queue."""
67. self.\_data.append(self.\_Item(key, value))
68. self.\_upheap(len(self.\_data) - 1)        # upheap newly added position
70. **def** min(self):
71. """Return but do not remove (k,v) tuple with minimum key.
73. Raise Empty exception if empty.
74. """
75. **if** self.is\_empty():
76. **raise** Empty('Priority queue is empty.')
77. item = self.\_data[0]
78. **return** (item.\_key, item.\_value)
80. **def** remove\_min(self):
81. """Remove and return (k,v) tuple with minimum key.
83. Raise Empty exception if empty.
84. """
85. **if** self.is\_empty():
86. **raise** Empty('Priority queue is empty.')
87. self.\_swap(0, len(self.\_data) - 1)        # put minimum item at the end
88. item = self.\_data.pop()                   # and remove it from the list;
89. self.\_downheap(0)                         # then fix new root
90. **return** (item.\_key, item.\_value)

**Question 12 (12 points):**

Consider the following simple binary tree class:

1. **class** SimpleTreeWithoutParent:
2. **def** \_\_init\_\_(self, element, left=None, right=None):
3. self.\_element = element
4. self.\_left = left
5. self.\_right = right
6. **def** \_\_str\_\_(self):
7. **return** str(self.\_element)
9. **def** PreOrderTraversal(root):
10. **if** root == None:
11. **return**
12. **yield** root.\_element
13. **if** root.\_left **is** **not** None:
14. **for** each **in** PreOrderTraversal(root.\_left):
15. **yield** each
17. **if** root.\_right **is** **not** None:
18. **for** each **in** PreOrderTraversal(root.\_right):
19. **yield** each
21. **def** convert\_preorder\_traversal\_to\_minheap(lissy):
22. # To Do for question number #12. Write your code in Page #21

Given the root of a binary tree where all of the nodes’ elements are integers. lissy is a python list which contains all the elements of that binary tree from Preorder traversal. Your task is to implement a python function **convert\_preorder\_traversal\_to\_minheap(**lissy**) that will convert** lissy **into the MinHeap**. Here parameter **root is the root node of a binary tree (class type: SimpleTreeWithoutParent) and** lissy **is a python list which contains all of the elements of that binary tree from Preorder traversal.**

After calling **convert\_preorder\_traversal\_to\_minheap(**lissy**)** function, lissy will represent a MinHeap structure.

**Implementation requirements:**

1. Your implementation’s run time should be O(n) and space complexity should be O(1).
2. Your implementation should be in place.
3. If you need, you may use additional functions (helper functions) with additional parameters.
4. You don’t need to use MinHeapPriorityQueue class for this question but if you want, you can use it or use concept from it.

For example,

1. leftSubTree = SimpleTreeWithoutParent(10, SimpleTreeWithoutParent(3), SimpleTreeWithoutParent(2))
2. rightSubTree = SimpleTreeWithoutParent(20,SimpleTreeWithoutParent(5), SimpleTreeWithoutParent(25))
3. root = SimpleTreeWithoutParent(30, leftSubTree, rightSubTree)
5. lissy = [each **for** each **in** PreOrderTraversal(root)]
6. **print**(lissy) # [30, 10, 3, 2, 20, 5, 25] - lissy contains elements from Preorder  traversal of the binary tree.
7. convert\_preorder\_traversal\_to\_minheap(lissy)
8. **print**(lissy) # [2, 10, 3, 30, 20, 5, 25] - now lissy represent a Minheap.

**def convert\_preorder\_traversal\_to\_minheap(lissy):**

**”””**

**@lissy: a list of integers, lissy contains all of the elements of a binary tree from PreOrder traversal.**

**Your task is convert this lissy into the minheap representation.**

**@return: Nothing**

**”””**

**# Your code**

**##starting from the tail, heapify**

ptr = len(lissy)-1

while ptr >= 0:

##check if node is bigger than its parent

if ptr%2 == 0:

parent = (ptr-2) / 2

if parent >= 0:

if lissy[parent] > lissy[ptr]:

##swap

lissy[parent],lissy[ptr] = lissy[ptr],lissy[parent]

if ptr % 2 == 1:

parent = (ptr-1) / 2

if parent >= 0:

if lissy[parent] > lissy[ptr]:

##swap to maintain min\_heap property

lissy[parent],lissy[ptr] = lissy[ptr],lissy[parent]

ptr -= 1

**Question 13(5 points):**

Given the following undirected simple graph, please draw the results of depth-first search traversal (DFS) and breadth-first search traversal (BFS) on the graph **starting at vertex D.**

A picture containing sitting, clock

Description automatically generated

**DFS:** \_D C B A E I M N K O J G F H L P\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

We go as deep as we can, once we found we could not go any deeper on current node, we return to the node that called current node. And we repeat same thing, checking if new node could be discovered

**BFS:** \_D C G H B J K L A F I N O P M \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Extra Page.** If you don't have enough space for answers. Please label which question(s) you are answering. Also put a note in the designated space for the question saying that the answer is on last page.