Connor Jensen
CSCI3320
Written Assignment #3
April 11, 2024

Q1) (15 points)

Based on the definition of the classes 'Node' and 'SinglyLinkedList' shown below, write a function for a singly linked list. Reduce the complexity of your algorithm as much as possible.

```
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None


class SinglyLinkedList:
    def __init__(self):
        self.next_val = None
        self.size = 0
```

a) (5 pts) compareLength(list1, list2):

*Write **a recursive function** to compare the length of the lists. Your recursive function will return three possible values: 1, 0, or -1, which denote respectively that the length of L1 is greater than, equal to, or less than the length of L2. Note that the length of a linked list is defined as the total number of nodes in the list.*

```
 def compareLength(list1, list2):
     # The worst case complexity (Big-O) of this function is O(n)
     # This is because the only non O(1) processes are findLength calls
     # which each take O(len(list) == n) processes each
     """
     Compares the length of two SinglyLinkedList objects
     :param list1: first SinglyLinkedList object
     :param list2: second SinglyLinkedList object
     :return int: 1 if (list1 > list2), -1 if (list1 < list2), else 0
     """
     def findLength(list0):
         """
         Finds the length of a singly linked list
         :param list0: SinglyLinkedList object
         :return int: length of list0
         """
         list_length = 0
         next_val = list0.head
         while next_val:
             list_length += 1
             next_val = next_val.next
         return list_length

     list1_length = findLength(list1)
     list2_length = findLength(list2)

     if list1_length > list2_length:
         return 1
     elif list1_length < list2_length:
         return -1
     else:
         return 0
```

b) (10 pts) equal_list(list1, list2):

(Do NOT use a set for your solution)

i. *Write a function to verify if two lists, L1 and L2, are equivalent, considering that both must contain identical elements irrespective of their order. It is assumed that the lists exclusively consist of positive numbers. The function should return 'true' if the lists are dissimilar and 'false' if they are identical.*

ii. *What is the worst-case complexity of your algorithm? The complexity of your algorithm is an important grading criterion.*

```
def equal_list(list1, list2):
    # The worst-case complexity (Big-O) of this function is O(n) This is because
compareLength() function is O(n) and
    # dict_creator() takes O(len(list0) == n) processes
    # This complexity is due in part to the builtin dict.get()
    # function which is a constant time operation, which may have lowered the worst-case
complexity
    """
    Checks if two SinglyLinkedList objects are equal
    :param list1: first SinglyLinkedList object
    :param list2: second SinglyLinkedList object
    :return bool: False if equal, True otherwise:
    """
    def dict_creator(list0, list_dict):
        """
        Creates dictionary from list0
        :param list0: SinglyLinkedList object
        :param list_dict: empty dictionary object
        :return dict: completed list_dict object
        """
        next_val = list0.head
        while next_val:
            list_dict[next_val.data] = list_dict.get(next_val.data, 0) + 1
            next_val = next_val.next
        return list_dict

    if compareLength(list1, list2) != 0:
        return True
    else:
        return dict_creator(list1, {}) != dict_creator(list2, {})
```

Q2) (9 pts)

Convert the following Infix notations to Postfix notations. Show the elements stored in the stack after processing each input symbol in the table.

a)	A*B - (C*D) + E														
Input	A	*	B	-	(	C	*	D	)	+	E				
Stack		*	*	-	-(	-(	-(*	-(*	-	+					
Postfix	**AB*CD-+**														
b)	A*(B+C)-(D/E)														
Input	A	*	(	B	+	C	)	-	(	D	/	E	)		
Stack		*	*(	*(	*(+	*(+	*	-	-(	-(	-(/				
Postfix	**ABC+*DE/-**														
c)	A+(B-C)-D+E*F/G														
Input	A	+	(	B	-	C	)	-	D	+	E	*	F	/	G
Stack		+	+(	+(	+(-	+(-	+	-	-	+	+	+*	+*	+/	
Postfix	**ABC-+D-EF*G/+**														

Q3) (6 points)

Assume you need to implement a queue using two stacks. Write a Python (or pseudo) code for the dequeue method based on the 'front' and 'enqueue' methods given below:

```
class TwoStackQueue:
    """FILO (First In Last Out) methods to manipulate two Stack objects"""
    def __init__(self):
        self.stack_head = Stack()
        self.stack_body = Stack()
        self.count = 0
```

front( ): *O(1)*

```
def front(self):
    if self.stack_body.count >= 1:
        return self.stack_body.top()
    elif self.stack_head.count >= 1:
        return self.stack_head.top()
    else:
        return "Queue is empty"
```

enqueue(X): *O(1)*

```
def enqueue(self, data):
    self.stack_body.push(data)
    self.count += 1
```

dequeue( ): *complexity should be O(n)* where n is the number of elements in the queue.

```
    def dequeue(self):
        """
        DESCRIPTION:
            Removes and returns the first element in the queue given queue isn't empty
        Complexity:
            In the worst case scenario, this method will have to pop all elements
            from stack_body and push them into stack_head.
            This function has the worst case time complexity of O(n),
            where n is the number of elements in the queue.
        Returns:
             first element data after removing it from the queue
        """
        if self.count >= 1:
            # If stack_head is empty, pop all elements from
            # stack_body and push them into stack_head
            if self.stack_head.count == 0:
                while self.stack_body.count > 0:
                    self.stack_head.push(self.stack_body.pop())
            self.count -= 1
            return self.stack_head.pop()
        else:
            return "Queue is empty"
```

Q4) (5 pts):

(Proof by induction) Prove the following statement using induction.

A binary tree with depth d has at most 2^d leaves.

a. Assume: $m\text{Leaf}(d)$: 1 $if(d = 0) else\ return(\text{mLeaf(d-1)} + m\text{Leaf(d-1)})$

Prove: $max\text{Children(d)} = 2^d$

1. **Base Case (N = 0):** $maxChildren_0 = 2^0$
2. **Left Side:** $max\text{Children}_0 = 1$ **(d=0 is just root node or 1 leaf)**
3. **Right Side:** $2^0 = 1$

Base Case is True because both the left and right side are equal to 1 when d=0

a. Assumptions for Inductive Case:

   a. $m\text{Leaf}(d)$: 1 $if(d = 0) else\ return(\text{mLeaf(d-1)} + m\text{Leaf(d-1)})$

   b. $2^d = mLeaf(d)$ (when d=0)

   c. $computationComplexity(maxLeaf(d)) = maxLeaf(d)$

   d. $d = d + 1$

b. Inductive Form: $maxLeaf(d + 1) = 2^{d+1}$

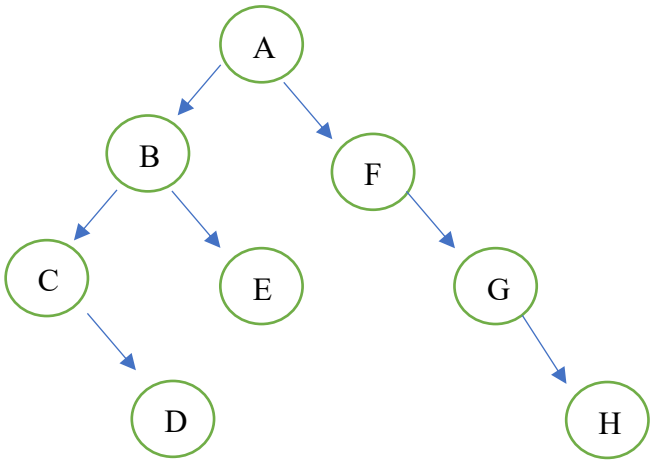
   a. $computationComplexity(max\text{Leaf}(d + 1)) = 2^{d+1}$

Inductive Case is True because both sides are equal to 2^{d+1} when $d = d + 1$

Q5) (5 pts)

Draw a binary tree with the eight nodes A, B, C, D, E, F, G and H, satisfying the following two conditions:

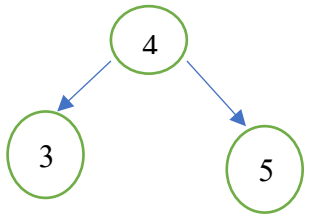
- if we print all the nodes in the *inorder* traversal, the output is **CDBEAHGF**;
- if we print all the nodes in the *preorder* traversal, the output is **ABCDEFGH**.



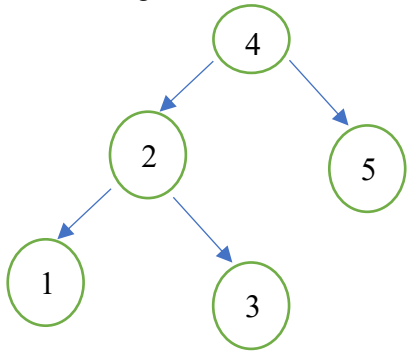
Q6) (8 pts)

Show the result of inserting 5, 4, 3, 1, 2, 9, 6, 8, into an initially empty AVL tree (insert one at a time in the given order).

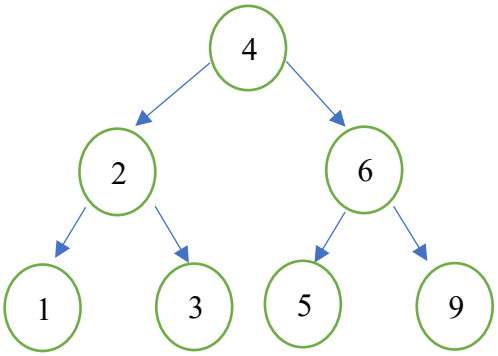
i) After inserting 5, 4, 3



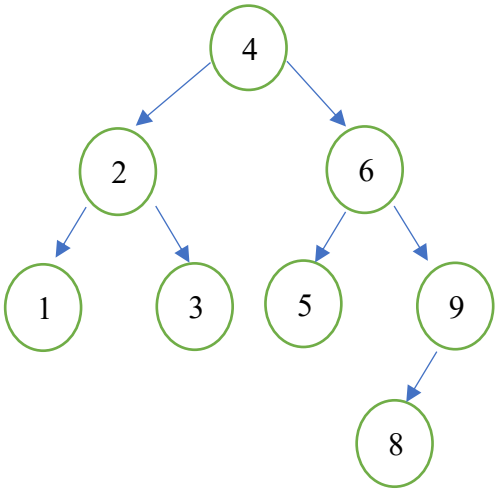
ii) After inserting 5, 4, 3, 1, 2



iii) After inserting 5, 4, 3, 1, 2, 9, 6



iv) After inserting 5, 4, 3, 1, 2, 9, 6, 8



Q7) (12 pts)

Use the B-tree shown below to answer the following questions:

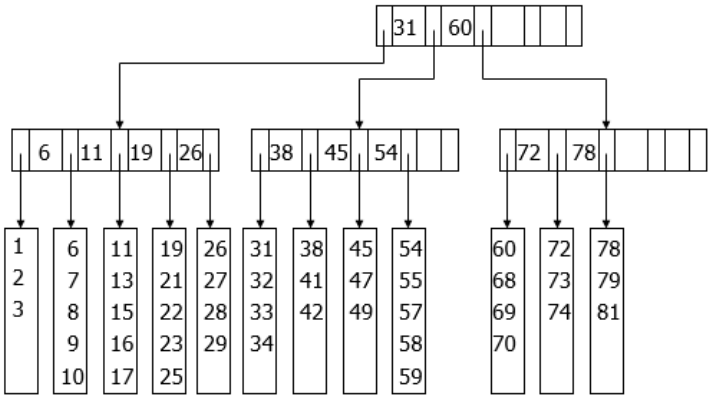
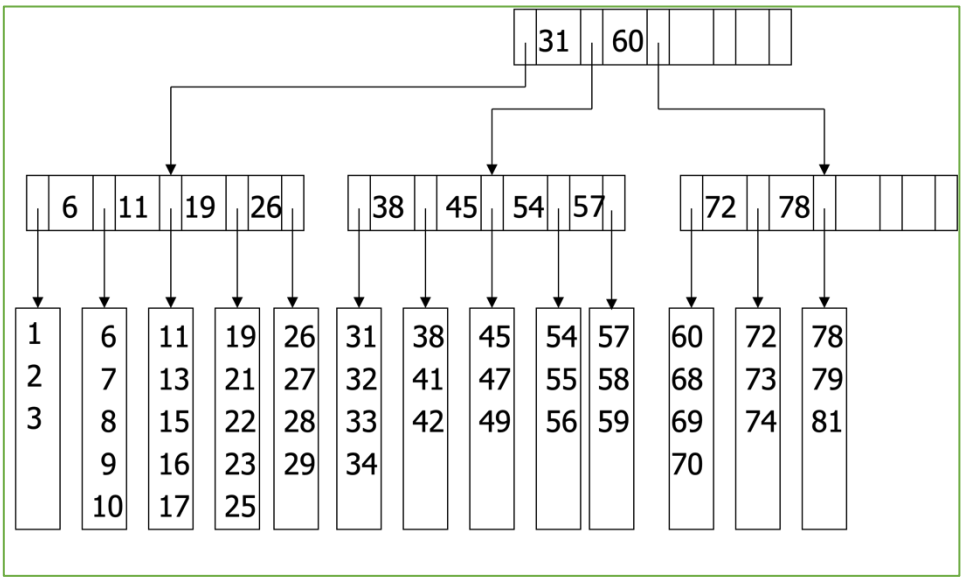


Figure 1. B-tree for Question 7

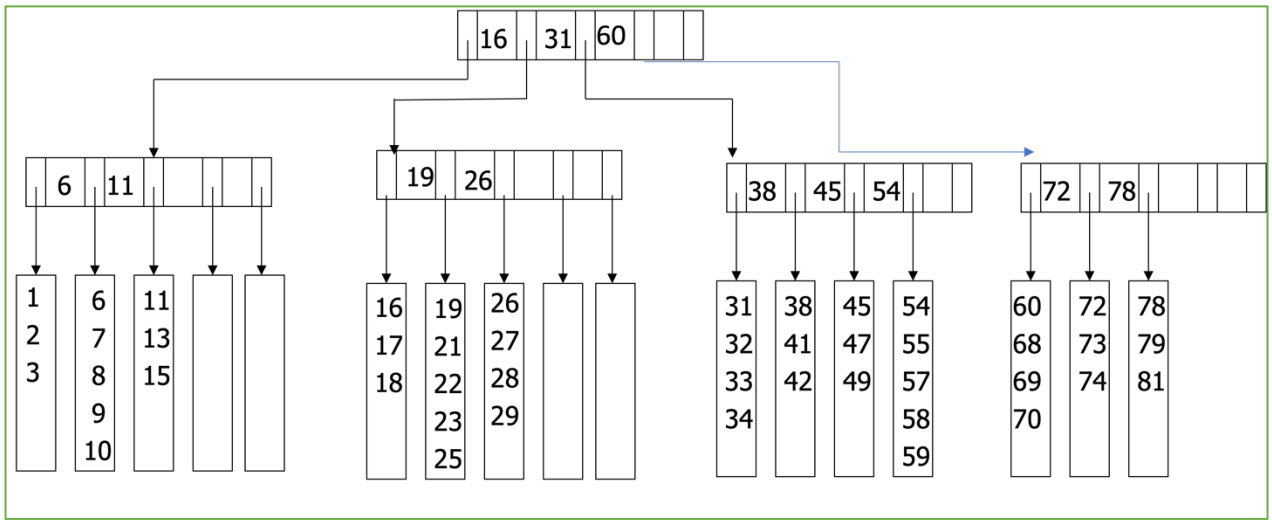
i) (2 pts) Find the parameters, M (branching factor), and L (number of recorders in a leaf) of the B-tree given in Figure 1.

M = 3, L=5

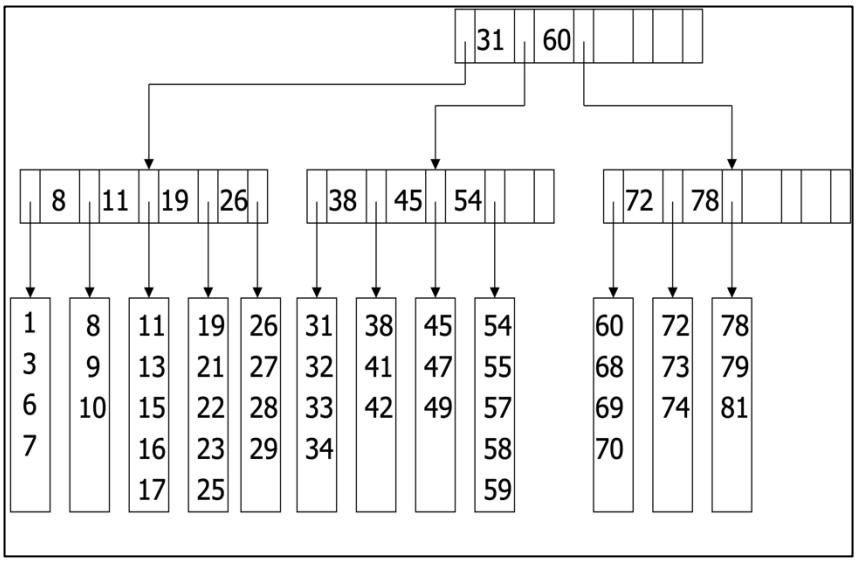
ii) (2 pts) Show the result of inserting a key '56' into the B-tree in Figure 1.



iii) (3 pts) Show the result of inserting a key '18' into the B-tree in Figure 1.



iv) (2 pts Show the B-trees after deleting ‘2’ from the B-tree in Figure 1.



v)(3 pts) Show **two potential states of the B-tree** after the removal of '72' from the B-tree in Figure 1.