Assignment 8

Name: DONGWOOK LEE

Problem 8.1 *Stacks & Queues*

(a)

Data Structure Implemented using Python:

StackUsingLinkedList.py | BinarySearch2LinkedList.py | LinkedList2BinarySearch.py

```python
class StackNode:
    def __init__(self, data):
        self.value = data
        self.address = None


class Stack:
    def __init__(self):
        self.size = -1
        self.currentSize = -1
        self.firstNode = None

    def set_size(self, size):
        self.size = size
        self.currentSize = 0

    def push(self, x):
        if self.size == -1:
            print("The Stack is not set")
            return False

        if self.currentSize >= self.size:
            print("Stack Overflow")
            return False

        new_node = StackNode(x)
        new_node.address = self.firstNode
        self.firstNode = new_node

        self.currentSize += 1

    def pop(self):
        if self.size == -1:
            print("The Stack is not set")
            return False

        if self.currentSize < 1:
            print("Stack Underflow")
            return False

        temp = self.firstNode
        self.firstNode = self.firstNode.address
        self.currentSize -= 1

        return temp

    def is_empty(self):
        if self.size == -1:
            print("The Stack is not set")
            return False

        if self.currentSize == 0:
            return True
        else:
            return False
```

(b)

```python
# Queue Behavior
class Queue:
    def __init__(self):
        self.stack1 = Stack()
        self.stack2 = Stack()

    def set_size(self, size):
        self.stack1.set_size(size)
        self.stack2.set_size(size)

    def push(self, x):
        self.stack1.push(x)

    def de_queue(self):
        for k in range(0, self.stack2.currentSize):
            self.stack2.pop()

    def set_queue(self):
        self.de_queue()
        for k in range(0, self.stack1.currentSize):
            self.stack2.push(self.stack1.pop().value)

    def pop(self):
        temp = self.stack2.pop()
        if temp:
            print(temp.value)
        return temp
```

Input & Result 1:

```python
Q = Queue()
Q.set_size(3)
Q.push('a')
Q.push('b')
Q.push('c')
Q.set_queue()
Q.pop()
Q.pop()
Q.pop()
Q.pop()
```

Run: StackUsingLinkedList

```
a
b
c
Stack Underflow

Process finished with exit code 0
```

Input & Result 2:

```python
Q = Queue()
Q.set_size(5)
Q.push(1)
Q.push(2)
Q.push(3)
Q.push(4)
Q.push(5)
Q.push(6)
Q.set_queue()
Q.pop()
Q.pop()
Q.pop()
Q.pop()
```

Run: StackUsingLinkedList

```
"C:\Users\danie\PythonProgramming\2. 실습프로젝트\venv\Scripts\python.exe" "C:/Users/danie/PythonProgramming/2. 실습프로젝
Stack Overflow
1
2
3
4

Process finished with exit code 0
```

Problem 8.2 *Linked Lists & Rooted Trees*

(a)

```
# n elements in Linked List

A = original Linked List
n = number of elements in A

for i in range(1, n):
    temp = A.head
    A.head = A.head.next
    A.head.next = temp

    if i == 1:
        A.head.next.next = None
```

In this Algorithm, we do not need to use any auxiliary storage (except using 'temp' as auxiliary variable)

We can reverse the Linked List only by changing the direction of pointers in between Stack Nodes.

(b)

Class Definition:

StackUsingLinkedList.py BinarySearch2LinkedList.py LinkedList2BinarySearch.py

```
# Suppose we have a BST named 'A'
# and make a Linked List


class Tree:
    def __init__(self, key):
        self.key = key
        self.parent = None
        self.left = None
        self.right = None


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


def tree_minimum(x):
    while x.left is not None:
        x = x.left
    return x
# θ(h)


def tree_successor(x):
    if x.right is not None:
        return tree_minimum(x.right)
    y = x.parent
    while y is not None and x == y.right:
        x = y
        y = y.parent
    return y
# θ(h)
```

Construct a Tree:

```
root = Tree(15)

a = Tree(6)
root.left = a
a.parent = root

b = Tree(18)
root.right = b
b.parent = root

c = Tree(3)
a.left = c
c.parent = a

d = Tree(7)
a.right = d
d.parent = a

e = Tree(17)
b.left = e
e.parent = b

f = Tree(20)
b.right = f
f.parent = b

g = Tree(2)
c.left = g
g.parent = c
```

```

h = Tree(4)
c.right = h
h.parent = c

i = Tree(13)
d.right = i
i.parent = d

j = Tree(9)
i.left = j
j.parent = i
```

Changing into a Linked List:

```
# Making into Linked List
currentTree = tree_minimum(root) # θ(h)
head = Node(currentTree.key)
tail = head
for k in range(1, 11): # θ(n-1)
    currentTree = tree_successor(currentTree) # θ(h)
    tail.next = Node(currentTree.key)
    tail = tail.next
```

Time Complexity: O((n-1)*h + h) = O(h*((n-1)+1)) = O(h*n)

Print out:

```python
# Print out to check validity
while head is not None:
    print(head.key)
    head = head.next
```

Run: BinarySearch2LinkedList

```
2
3
4
6
7
9
13
15
17
18
20
```