HW6

36-650 – Statistical Computing

Deadline: November 14th, 11:59PM EST

- You must submit your HW solution as series of python files and screenshots showing the results for running your code.
- Your GitHub repository should contain a folder named HW-6. Inside this folder, there should be a folder for each question.
- On GitHub, check-in the answer for each question in a separate python file in the corresponding folder.
- Each python file may include one or more python functions. Each function SHOULD NOT exceed 15 lines of code (excluding comments).
- For each question, submit a screenshot for the result of running your code.
- Make sure your GitHub repository is publicly viewable.
- On Canvas, only submit a URL to your GitHub repository.
- Plan to get the needed assistance from instructor or TA by Friday. Expect no assistance over the weekend.
- Not following the submission guidelines may result in grading penalties.

Question 1 (10 points)

You have list of Agile story statuses (“Ready”, “In Progress”, “Blocked”, “Completed”). You would like to store these values in your application and they will be accessed and searched several times but they won’t be changed. Which built-in data structure can be used? List or Tuple? And Why?

Question 2 (10 points)

In the Queue implementation that was shown in the class, write delete() python function to delete an item with specific value. Your function should access elements in the queue using enqueue() and dequeue() functions only. Refer to the enqueue() and dequeue() implementations that were demoed in the class.

```
obj = Queue()
obj.enqueue(5)
obj.enqueue(7)
obj.enqueue(13)
obj.enqueue(4)
obj.enqueue(7)

obj.delete(7)
print(obj.dequeue()) # Should return 5
```

```
5
```

Question 3 (10 points)

Develop python function that can be called to sort unsorted linked list in ascending order of the data values in the Nodes.

- Use the Linked List implementation shown in the lecture.
- Your function should be named sort(self) and should be added to the Linked List class
- Don't add any additional pointers to the Linked List class (other than the head variable).
- Don't use any built-in sort() functions provided by Python.

```
first_node = Node()
first_node.set_data(11)
linked_list = LinkedList(first_node)
linked_list.insert(3)
linked_list.insert(6)
print("The Linked List is:")
linked_list.print_list()

linked_list.sortList()
print("After sorting, the Linked List is:")
linked_list.print_list()
```

```
The Linked List is:
11  3  6

After sorting, the Linked List is:
3  6  11
```

Question 4 (10 points)

Write Python function to delete set of keys from a dictionary. Your function should accept list of the keys to be deleted and the dictionary

delete_keys([“key1”,”key2”], dictionary)

```
dict = {"firstName": "Mohamed", "lastName": "Farag", "birthYear": 1990, "nationality": "Egyptian"}
print(delete_keys(["lastName","birthYear"], dict))
```

```
{'firstName': 'Mohamed', 'nationality': 'Egyptian'}
```

Question 5 (10 points)

Develop Python function to remove duplicates from Singly Linked List (unsorted). Use the Linked List implementation introduced in the class materials.

```
first_node = Node()
first_node.set_data(11)
linked_list = LinkedList(first_node)
linked_list.insert(3)
linked_list.insert(6)
linked_list.insert(3)
linked_list.insert(11)
linked_list.insert(6)
linked_list.insert(5)
linked_list.insert(7)
linked_list.insert(5)

print("The Linked List is:")
linked_list.print_list()

linked_list.remove_dups()
print("After removing the duplications, the Linked List is:")
linked_list.print_list()
```

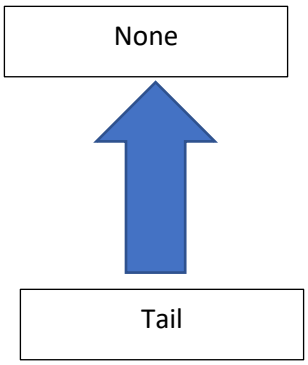
```
The Linked List is:
11  3  6  3  11  6  5  7  5

After removing the duplications, the Linked List is:
11  3  6  5  7
```

Question 6 (20 points)

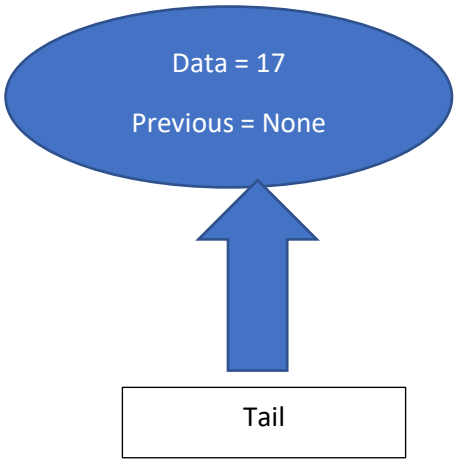
Build a data structure named “Reverse Linked List”. This data structure should have only two attributes: data and previous.

Create a class named ReversedLinkedList that keeps track of the linked list using one attribute, named “tail”.

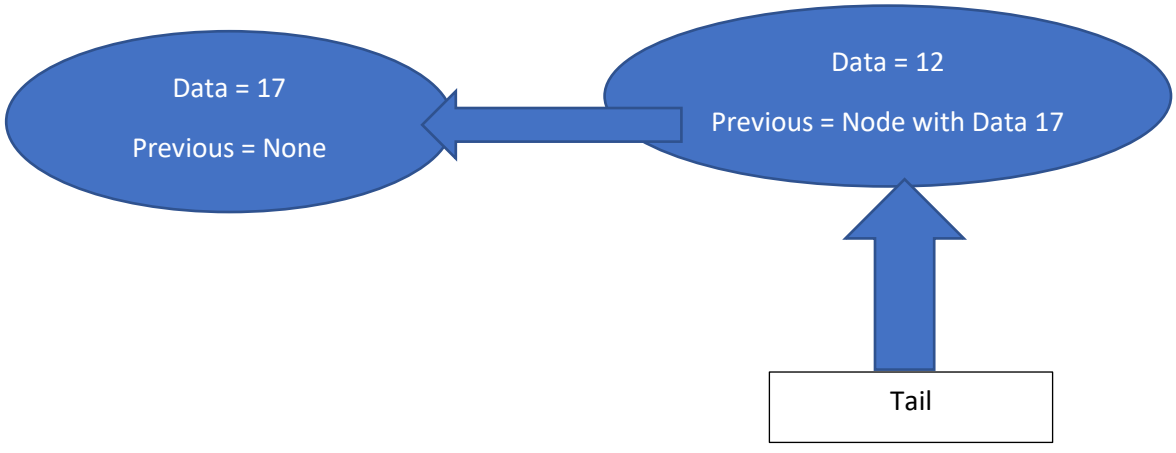
Step 1: Empty Data Structure



Step 2: Insert Node with Data of 17



Step 3: Insert Node with Data of 12



Conduct the following:

1. Create the Data Structure (storage data type and the organizer)
2. Create the insert() function
3. Create the delete() function. Predict its behavior from the insertion flow shown above.
4. Create search() function that returns true/false based on if given data point is stored in any of the nodes (i.e. search (12) → true, search (20) → false)

```
first_node = Node(11)
linked_list = LinkedList(first_node)
linked_list.insert(3)
linked_list.insert(6)
linked_list.insert(5)

print("The Linked List is (after insertion):")
linked_list.print_list()


linked_list.delete(6)

print("The Linked List is (after deleting 6):")
linked_list.print_list()

print("Does 5 exist in the Linked List?")
print(linked_list.search(5))

print("Does 17 exist in the Linked List?")
print(linked_list.search(17))
```

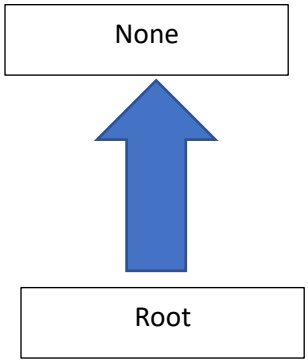
```
The Linked List is (after insertion):
5  6  3  11

Node deleted is 6
The Linked List is (after deleting 6):
5  3  11

Does 5 exist in the Linked List?
True
Does 17 exist in the Linked List?
False
```

Question 7 (20 points)

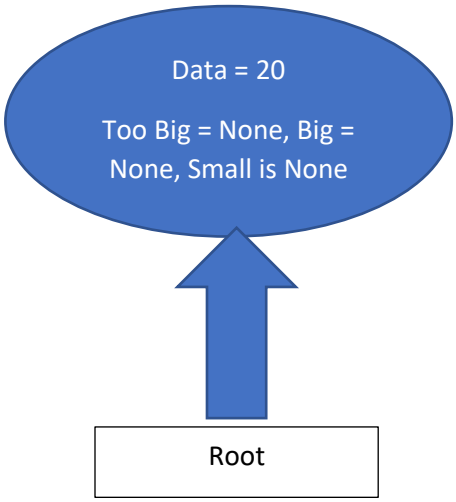
Create Custom Tri-tree in which every node contains up to three children: **too-big, big and small**. Your root node is always the first node that gets inserted in an empty tree.

- **Too-big children** is identified if the difference between the new data and the node's data is 10 or more (i.e. new data – current data >= 10)
- **Big children** is identified if the difference between the new data node and current data is less than 10 (i.e. 0 =< new data – current data < 10)
- **Small children** is identified if the new data is less than the current data (i.e. new data – current data < 0)

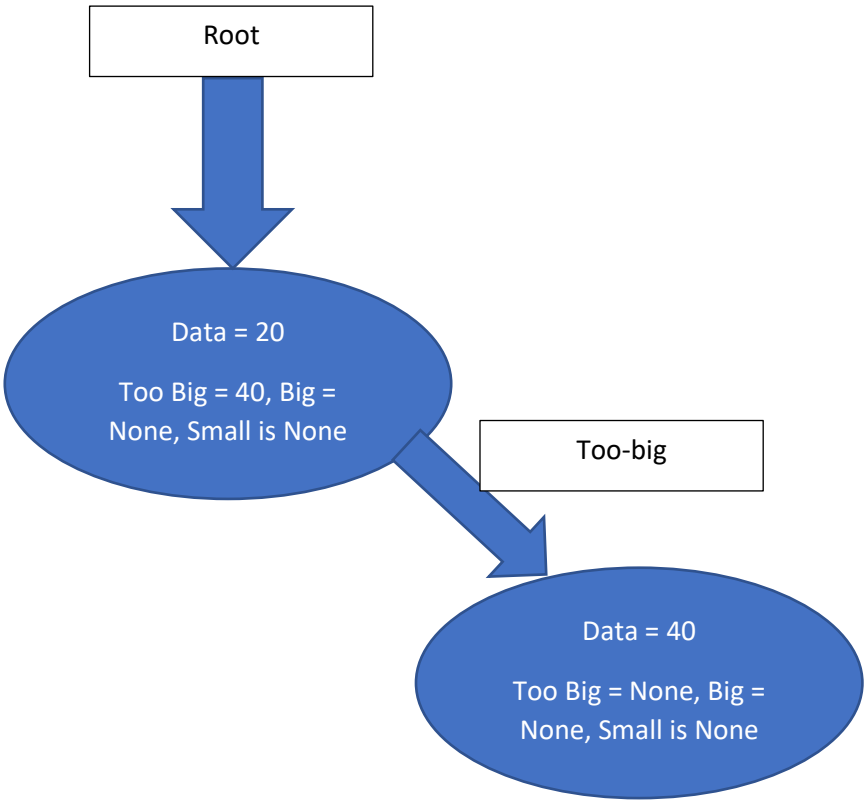
Step 1: Empty Tree



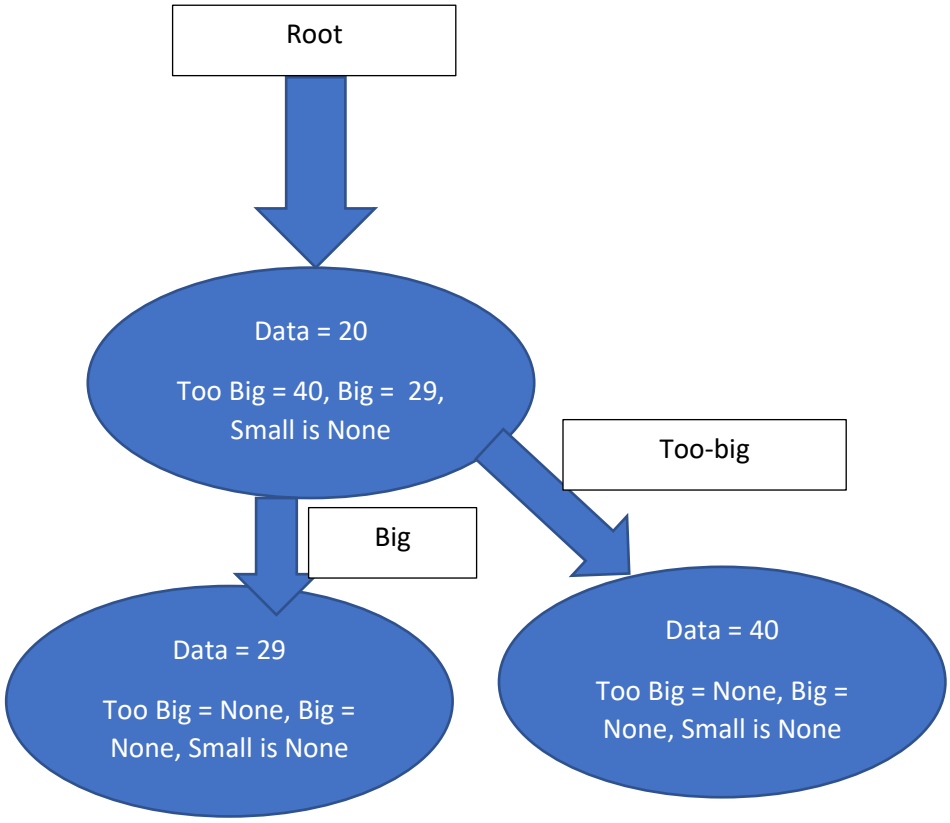
Step 2: Insert Node with Data of 20



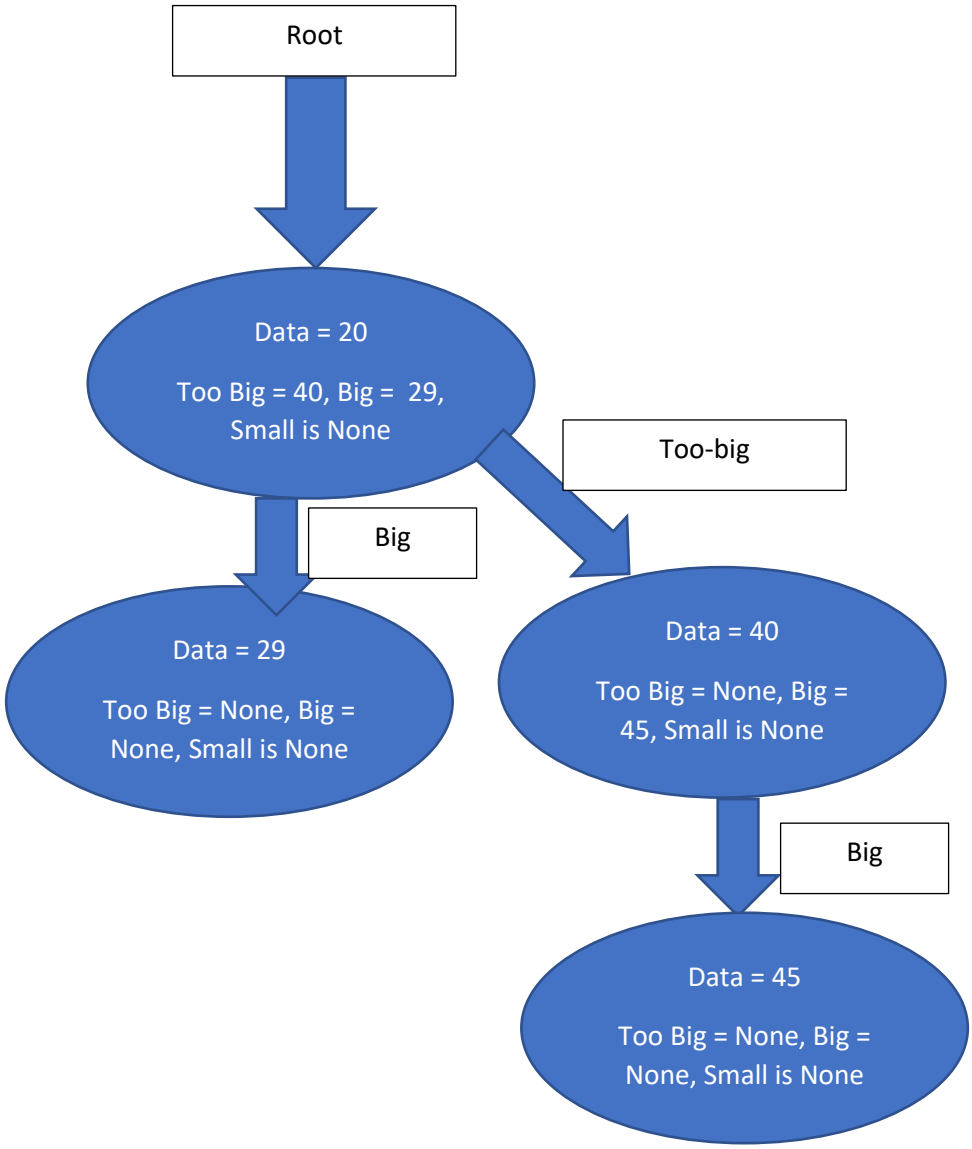
Step 3: Insert Node with Data of 40



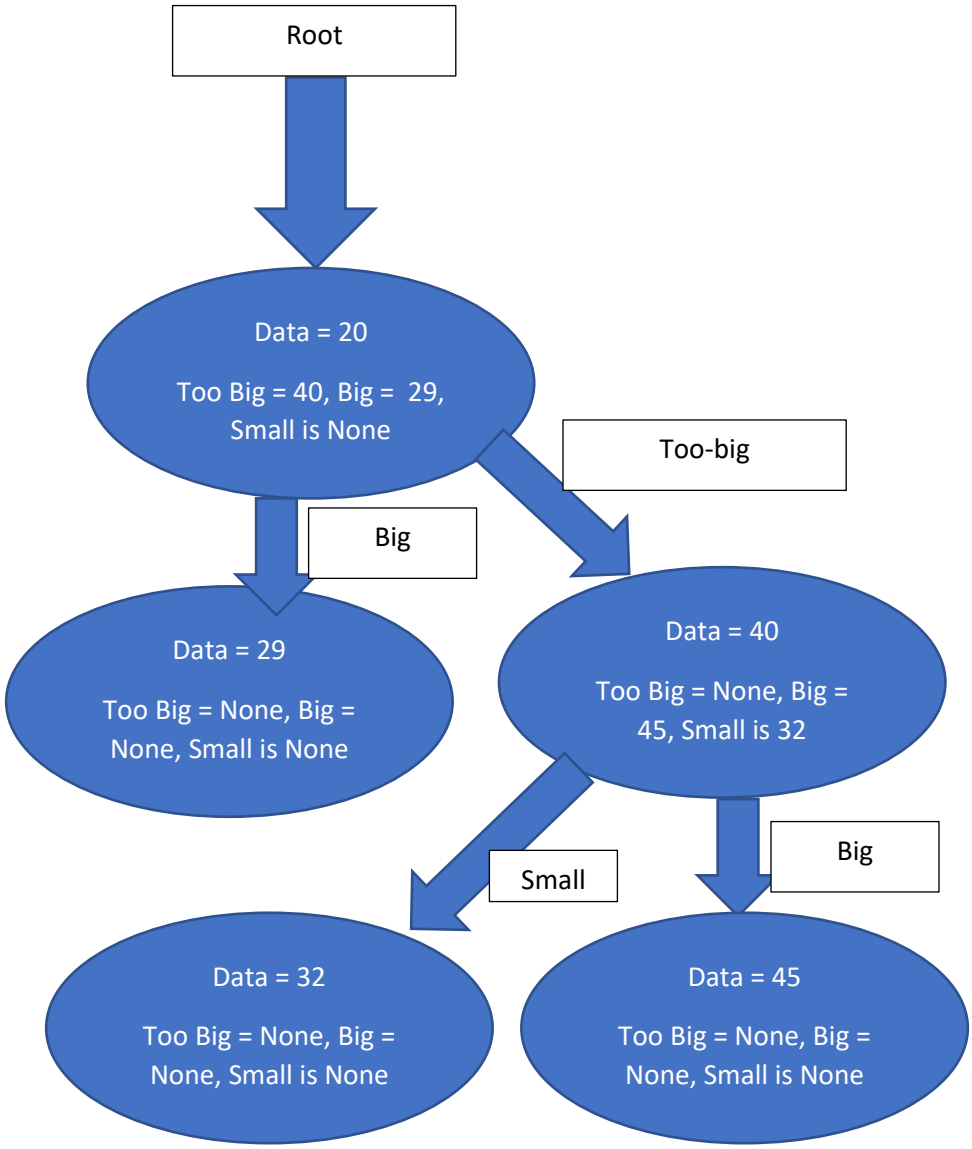
Step 4: Insert Node with Data of 29



Step 5: Insert Node with Data of 45



Step 6: Insert Node with Data of 32



Based on the examples above, conduct the following tasks:

1. Create the data structure (custom data type and the organizer)
2. Create the insert() function for this data structure
3. Create the delete () function for this data structure
4. Create traversal() function for this data structure. Your traversal should start with the small child, then the root node, followed by big child, then too-big child. Your function should be recursive.

```
root = Node(20)
root.insert(40)
root.insert(45)
root.insert(32)

print("Tree contents after insertion using the traversal():")
root.traversal()

root.delete(45)

print("Tree contents after deleting 45 using the traversal():")

root.traversal()
```

```
Tree contents after insertion using the traversal():
20
32
40
45
Tree contents after deleting 45 using the traversal():
20
32
40
```

Common Penalties:

- Your GitHub repository is not public: 100% reduction (won't be graded)
- Late submissions on Canvas or GitHub: 100% reduction (won't be graded)