Algorithms and Data Structures

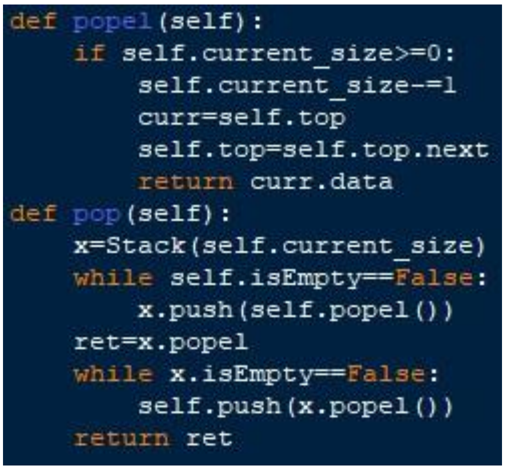
Homework 7

Dragi Kamov

08 April 2018

Problem 1

a. My code for this problem is in “stack.py”
b. We could achieve this by having one stack where we would add elements and whenever we would need to pop elements to achieve the effect of a queue we would transfer every element to the second stack. And then when we would pop elements from the second stack they would come out as if they were in a queue. Here is a snippet of code of how I would implement this.



```
def popel(self):
    if self.current_size>=0:
        self.current_size-=1
        curr=self.top
        self.top=self.top.next
        return curr.data
def pop(self):
    x=Stack(self.current_size)
    while self.isEmpty==False:
        x.push(self.popel())
    ret=x.popel
    while x.isEmpty==False:
        self.push(x.popel())
    return ret
```

Problem 2

a. My code for this problem is in “reverse.py”. It is an in-situ algorithm because I am not using any other data structure to store the elements, but I am just using three nodes as cursors to move through the linked list.
b. My code for this problem is in “binary_search.py”. And the time complexity of this algorithm is Θ(n), because in the transferring from binary tree to linked list I am going recursively through each element only once. And in the function called inorder(x,root) I am starting from the right side and then going to the left side because while pushing elements in the linked list I am always inserting elements on the left side in order to save on time complexity in the way that I wouldn’t move to the end of the list to insert the element.
c. My code for this problem is in “toLinkedList.py”. And the time complexity for creating the binary tree is Θ(n). The only problem is that I am starting to insert elements in the binary tree from the beginning of the linked list which is sorted so my binary tree wouldn’t have a left side because everything would be on the right side considering that every next element in the linked list is greater than the previous one.