Mastering Data Structures and Algorithms
Homework 3

Yiying Peng (Christine)

March 27, 2019

Problem 1

Reverse a queue using recursion.

Sol. My Python algorithm implementation is as follows.

```
class Queue:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def add(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop(0)

    def front(self):
        return self.items[0]

    def printQueue(self):
        for i in self.items:
            print(i, end=" ")


def reverseQueue(q):
    if (q.isEmpty()):
        return
    s = q.front();
    q.pop();
    reverseQueue(q)
    q.add(s)


q = Queue()
q.add(1)
q.add(2)
q.add(3)
q.add(4)
q.add(5)
reverseQueue(q)
q.printQueue()
```

The time complexity of this algorithm is $O(n)$

The space complexity is also $O(n)$.

□

Problem 2

Check for the following symbols to be balanced on a given string/text file: $\{\}[]()<>$.
Note: the following is not considered balanced: $\{<\}>$; but this is: $\{\}<>$.

Sol. My Python algorithm implementation is as follows.

```python
class Solution:
    def isBalanced(self, s):
        if s == '':
            return True

        lst = []
        dict_br = {'}': '{', ']': '[', ')': '(', '>': '<'}

        for ss in s:
            if ss not in dict_br:
                lst.append(ss)
            else:
                if lst != [] and (lst[-1] == dict_br.get(ss)):
                    lst.pop()
                else:
                    return False
        return (len(lst) == 0)
```

Using stacks method. Adding the element which hasn't been match in the list, removing it when it find a pair. If we get an empty list at the end, it is a balanced string/text.
The time complexity of this algorithm is $O(n)$ and the space complexity is also $O(n)$.

□