Assignment 4

Kevin G date / /

1. Stacking - push(8): stack has [8].
   push(2): Stack has [8,2].
   pop(1): Top element removed [8].
   push(pop(1)*2): pop 8 [ ]. Multiply 8*2=16. Add to stack [16].
   push(10): stack has [16,10]
   push(pop(1)/2): pop 10 [16]. Divide 10/2=5. Add to stack [16,5].

   Final stack: [16,5]

2. Queue - push(4): Queue has [4]
   push(pop(1)+4): pop 4 [ ]. Add 4+4=8. Add to queue [8].
   push(8): Queue has [8,8].
   push(pop(1)/2): pop(1)8 has [8]. Divide 8/2=4. Add to queue [8,4].
   pop(1): Front/first element 8 is removed. Queue [4].
   pop(1): remove last element. Queue [ ].

   Final Queue: [ ].

3. Find in deque: dust like a DLL, we can modify the deque to be able to traverse from the end of the list. thus making making faster time for some operations. An algorithm could be:

```
Find Deque (Deque<Integer> q, int x) {
  int left = 0;
  int right = q.size() -1;
 Integer[] dArray = q.toArray(new Integer[0]);
  while(left <= right){
    if(dArray[left]==x){ return left;}
    if(dArray[right]==x){return n-1-right;
    left ++;
    right--; }
    return -1; }
```

Assignment 4 Cont... date / /

7. Balanced Brackets - Time complexity is O(n) where n is the length of the input string s. This is because we loop through each character once. This takes O(n) time.
As for the rest of the stack operations, for each character we perform O(1) operations like add, remove, and peek. since we do this for every character, the time is O(n).
$O(n) + O(n) = O(2n) = O(n)$

Space complexity is O(n) since a worse case would be when the stack is storing opening brackets which grows at size O(n).

Decode String - Time complexity is O(n) where n is the lenght of the output string. This function iterates over each character once so its O(n). The stack operations push and pop run in O(1) complexity. Then there is appending string when reaching a closing bracket. while this gets repeated a k amount of times, all substrings are pushed and popped proportional to the length of output string. thus time complexity is O(n)

Space complexity is O(m+n) where m is the length of the input string and n is the length of the output string. the countStack and stringStack store intermediate counts and strings. the m is important to the depth of nested brackets.

Infix to Postfix: Time complexity is O(n) where n is the length of infix array. the loop iterates through each character and does push and pop O(1) operations at most once. each character gets appended which stays at O(n).
Space Complexity is O(n) where n is the infix array since we store the output postfix expression and have a stack that can hold n amounts of elements.