Math 320 Homework 2.2

Chris Rytting

September 25, 2015

2.8

```
import numpy as np
import time

def Exercise1():
    def prepend():
        list1 = []
        for i in xrange(10000):
            list1.insert(0,i)
        return list1


    def appendation():
        list1 = []
        for i in xrange(10000):
            list1.append(i)
        return list1

    start = time.time()
    prepend()
    end = time.time()
    difference = end - start
    print difference

    start = time.time()
    appendation()
    end = time.time()
    difference = end - start
    print difference

Exercise1()
```

Output:

0.0269570350647
0.0010871887207

It takes over twice as much time for the prepend algorithm because python has to shift every element of the list when prepending but only has to tack one element to the tail of the list with appending.

2.9

Implement a class called stack. We initialize the stack class with an array of length n, specified by the user. We implement the push function by putting an object at the first empty entry of the array. We implement the pull function by copying the last object of the array to a dummy variable, deleting it, and then returning the dummy variable. The best case scenario temporal complexity is $O(1)$, because we only have to perform each operation once, and the worst case scenario is $O(1)$, since we only perform each operation once.
However, if we make the array too short, we will have to initialize a bigger array to house the shorter array and thereby increase its capacity. This doesn't affect the temporal complexity, though.

2.10

Implement a class called queue. We initialize the queue class with an array of length n, specified by the user. We implement the enqueue function by putting an object at the first empty entry of the array. We implement the dequeue function by copying the first object of the array to a dummy variable, deleting it, and then returning the dummy variable, we will then shift every other element up one spot so as to reindex the array. The best and worst case scenarios of enqueue are of temporal complexity $O(1)$, because we only have to perform the operation once, and the worst case scenario of dequeue is $O(n)$ because we need to copy and delete the object, but then we need to shift every object by one index, which would result in at worst $n + 1$ operations. The best case scenario is $O(1)$, since we only perform one operation.
However, if we make the array too short, we will have to initialize a bigger array to house the shorter array and thereby increase its capacity. This doesn't affect the temporal complexity, though.

2.11

Implement a function called palindrome finder. Then taking a deque as input, we would compare the last and the first elements of the array. This will result in $\frac{n}{2}$ comparisons, so it will be in $O(n)$.

We could also use a stack, where we make a copy of the stack, then take the last element and insert it into a new stack element by element. This new stack will be the first stack but backwards, and then we can compare the two stacks element by element. This would be much slower than using a queue. I am convinced that a deque is the fastest way.