Matthew Kirk

Section 4

CS 235 Fall 2013 Midterm

Big O Analysis

I have also included my main function with the files if you care.

1. Constructor: DoubleLinkedList(void){code}
    a. This is O(1) because it is just initialization.
2. Destructor: ~DoubleLinkedLIst(void) {}
    a. This is O(n) because it iterates through the list deleting every node it passes.
3. addFront
    a. This is O(1) because it just has to move a couple of pointers around and the size of the list is irrelevant.
4. addRear
    a. This is an O(n) operation because it iterates through the entire list before adding the node at the end.
5. addAt
    a. This is generally O(n) because it has to iterate until the index position has been reached, which is dependent on the size of the list.
6. Remove
    a. This is generally O(n) because it also must iterate to the index position, which is dependent on the size of the list.
7. At
    a. Also O(n) for the same reasons. It must iterate to the index position from the beginning.
8. Contains
    a. O(n) for the same reasons. It must iterate until it gets to the data it is looking for. If it doesn't find it, it goes through the entire list.
9. Size
    a. O(1) because it is returning the value for numItems that has been being tracked by every other function.
10. Swap
    a. O(n) because it does not have any nested for loops, just single for loops. Realistically, it is probably more like O(4n), but we ignore the coefficient in BigO.
11. Shuffle
    a. O(n^2) because it has a for loop inside which it calls swap, which also has for loops. So, it has doubly nested for loops, making it O(n^2).
12. isPalindrome
    a. O(n) because there are no nested loops. It does three different loops that iterate through the list, so more realistically it would be O(3n), but we ignore the coefficients.

13. getKthFromRear
    a. O(n) because all it does is call the at functions, which is O(n).
14. removeDuplicates
    a. O(n^2) because it has a for loop and also calls functions that have for loops themselves. At most there is only a nest of for loops 2 deep, which makes it O(n^2)
15. Reverse
    a. O(n^2) because it has its own for loop inside which it calls swap, which is O(n). Together they result in O(n^2) efficiency.
16. Count
    a. O(n) because it has to check every node in the list to see if it has the data before adding it to currentCount.