List ADT Personalization & Complexity

CSCI 230- Homework 2

Ben Hairston
Computer Science Department
College of Charleston
Charleston SC, USA
hairstonbp@g.cofc.edu

ABSTRACT

Analysis of a program and its algorithms is essential for making the most efficient program possible. Whether it is a segment of code, a method, or an entire program, it must be checked for efficiency to make the best out of limited system resources. In this analysis of the orderedList.java class, the remove, contains, printList(), and removePalindromes() will be evaluated for its computational efficiency.

CCS CONCEPTS

• Theory of computation • Mathematics of computing

KEYWORDS

Complexity, Analysis, Efficiency, orderedList

1 Theoretical Analysis

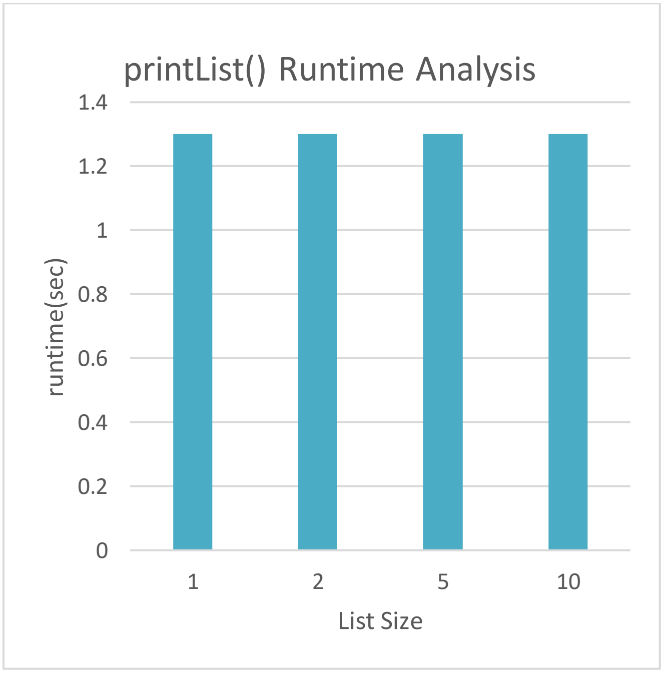
If we consider the remove method from the orderedList.java class, it can be broken down into 2 parts: the for loop and the ArrayList.remove() method call. The for loop, the segment that in the remove method would iterate such that it would be limited by n because it iterates through the list until it reaches the value that is being removed. The best case would be O(n), the worst case would also be O(n), and the average case would be O(n).

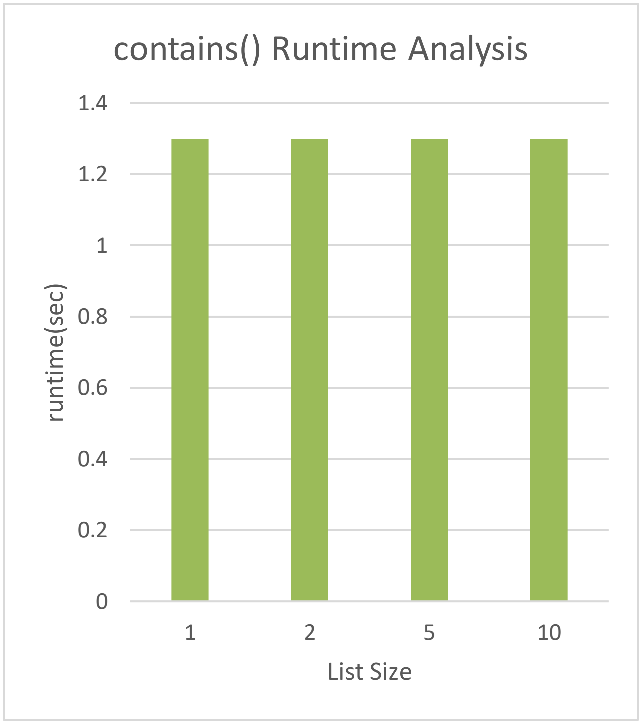
For the contains() method the complexity is very similar to the remove method. It has an if-else statement so its bounded linearly by n since it must check the entire list one-by-one for the value in question. In the best case the if statement would execute and find the value in the first position and then it would print that the value is in the list. The worst case would be that the if statement is executed and it checks the entire list for the value in question, and upon not finding it triggers the else statement. The if-else statement is bounded linearly because it iterates through the whole list and compares the values to the value input. Therefore, the worst best and average case in big-O notation would be O(n).

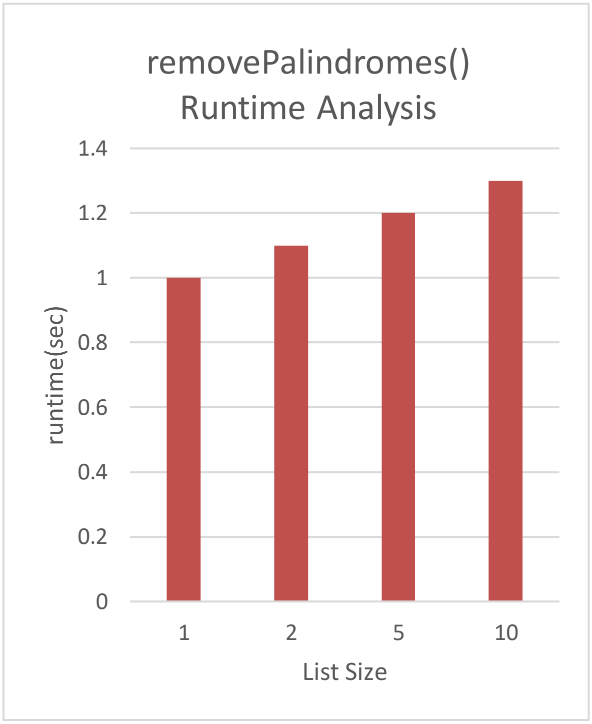
The printList() method is just a constant time method that prints the orderedList into a string for the user to see what is in the list. Because of this the time complexity is O(1) for all cases. Methods like removePalindromes() are more complex than the others because it combines contains() and remove() and makes it actually automate removing. One of the first major parts was the for loop that was bounded linearly again because it is being used to iterate through the list. Then nested inside it is a while loop that compares two integer values. Last inside the while loop is an if-else statement. The worst-case complexity would be O(n^3), the best-case would be O(n^2) and the average case would be O(n^2).

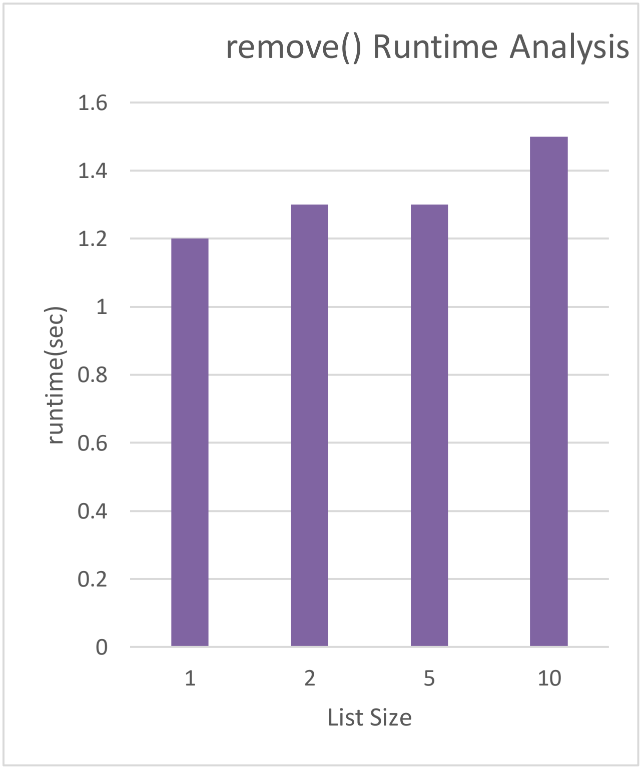
2 Practical Analysis

For the practical analysis all four methods were tested with the same list sizes, 1, 2, 5, and 10, for a controlled experiment. With such small list sizes, the times did not vary by much, but the remove and remove palindromes methods had marginal increments in which as the size of the list went increased, the runtime increased by an increment of on average 0.1 seconds as shown in the graphs below.









Conclusion

Based on the analysis of the methods, the runtime efficiency might not matter for small list sizes like 10 -100 but for a list with thousands to millions of entries the 0.1 second differential would become a major factor in coding style and what kinds of methods are used to program things like a specified remove function.

REFERENCES

[1] S.Kuredjian, Algorithm Time Complexity and Big O Notation, https://medium.com/@StueyGK/algorithm-time-complexity-and-big-o-notation-51502e612b4d
[2] https://en.wikipedia.org/wiki/ACM_Computing_Classification_System#History