**CPSC319 Data Structure, Algorithms and Their Applications**

**Assignment #2**

Report by: Radia Jannat UCID: 30118181

Email: [radia.jannat@ucalgary.ca](mailto:radia.jannat@ucalgary.ca) Tutorial number: T06

TA's name: Zaman Wahid

## Table of contents:

**Title Page Number**

1. **Question 1**
   1. Method ‘result’…………………………………………………….. 04
   2. Method ‘anagramOrNot’………………………………………….05

## Question 2

* 1. Class Node………………………………………………………...07
  2. Class LinkedLists………………………………………………....08
  3. Class Assign2……………………………………………………..10

1. **Conclusions**

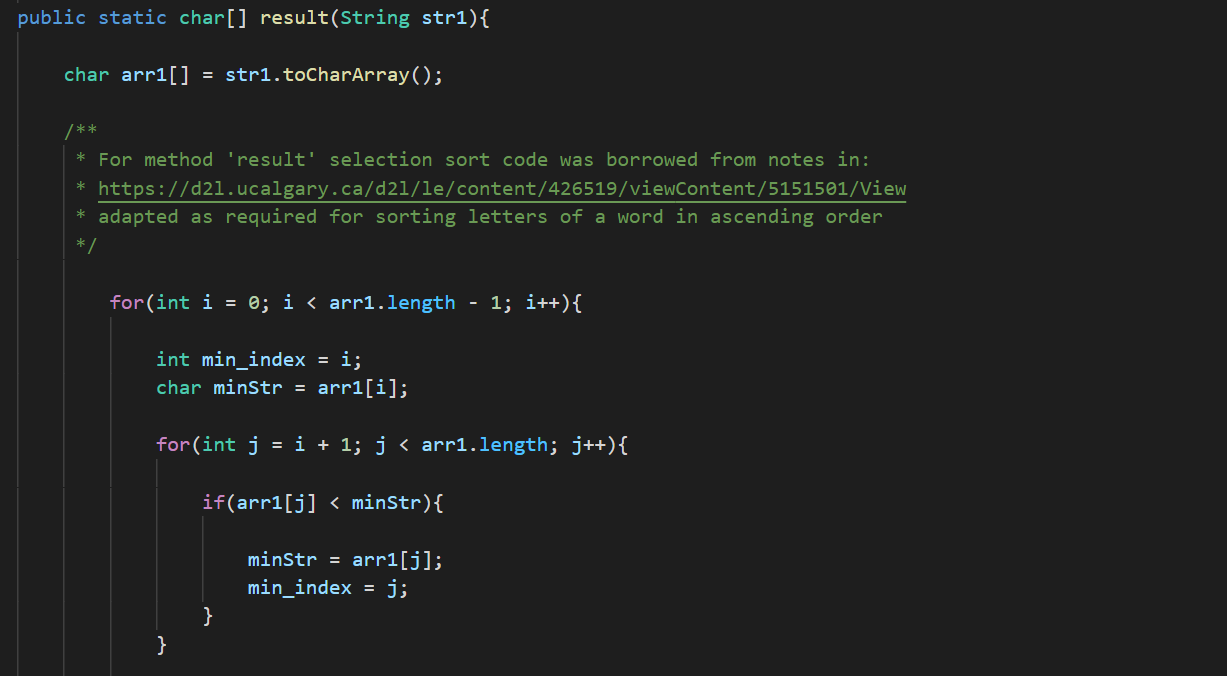
………………………………………………………………………13

**Question 1: What is the worst-case complexity of your algorithm when checking if two words are anagrams of each other? Express this using big-O notation, and use the variable *k* to represent the number of letters in each word. Support this with a theoretical analysis of your code.**

To find the worst-case complexity of my anagram checking algorithm I will explain the big-O complexity of the two methods used for checking if two words are anagrams:

The two methods used are ‘anagramOrNot’ and ‘result’, where the former methods determines whether two words are anagrams or not and the latter method is a helper method called from ‘anagramOrNot’ twice to check if two words are anagrams.

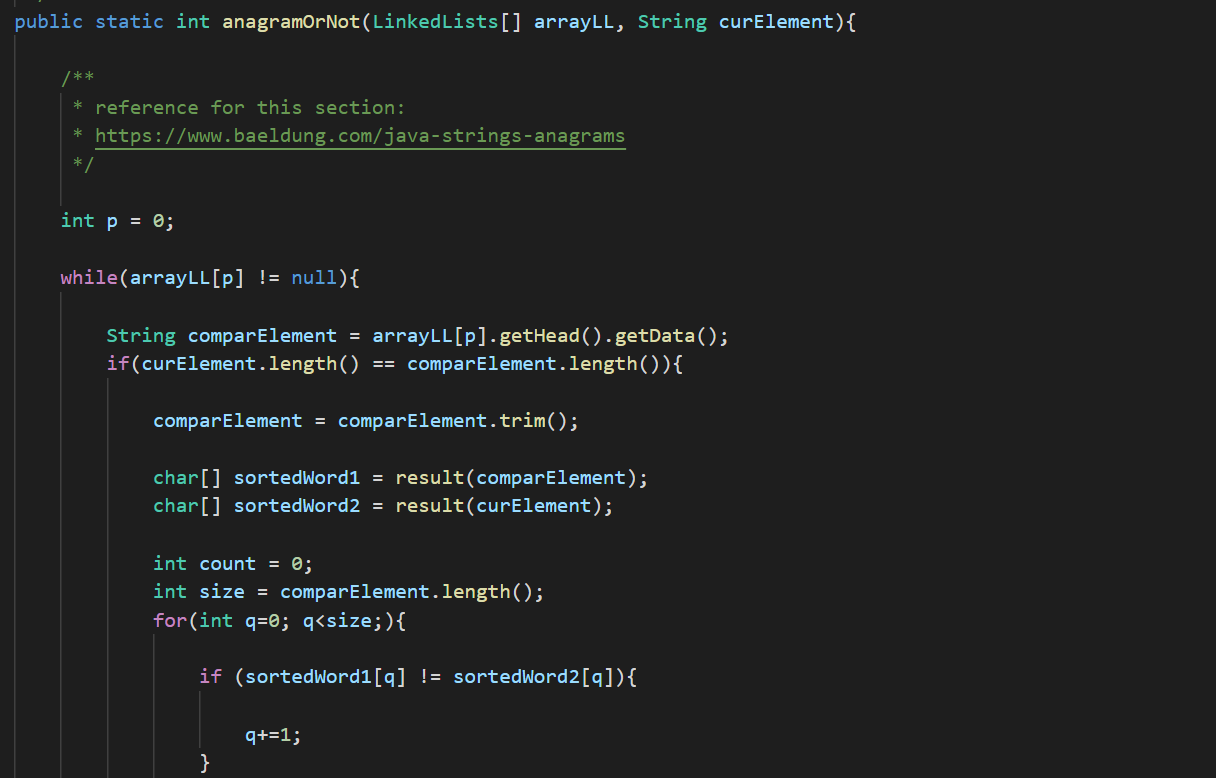
**Method ‘result’:**

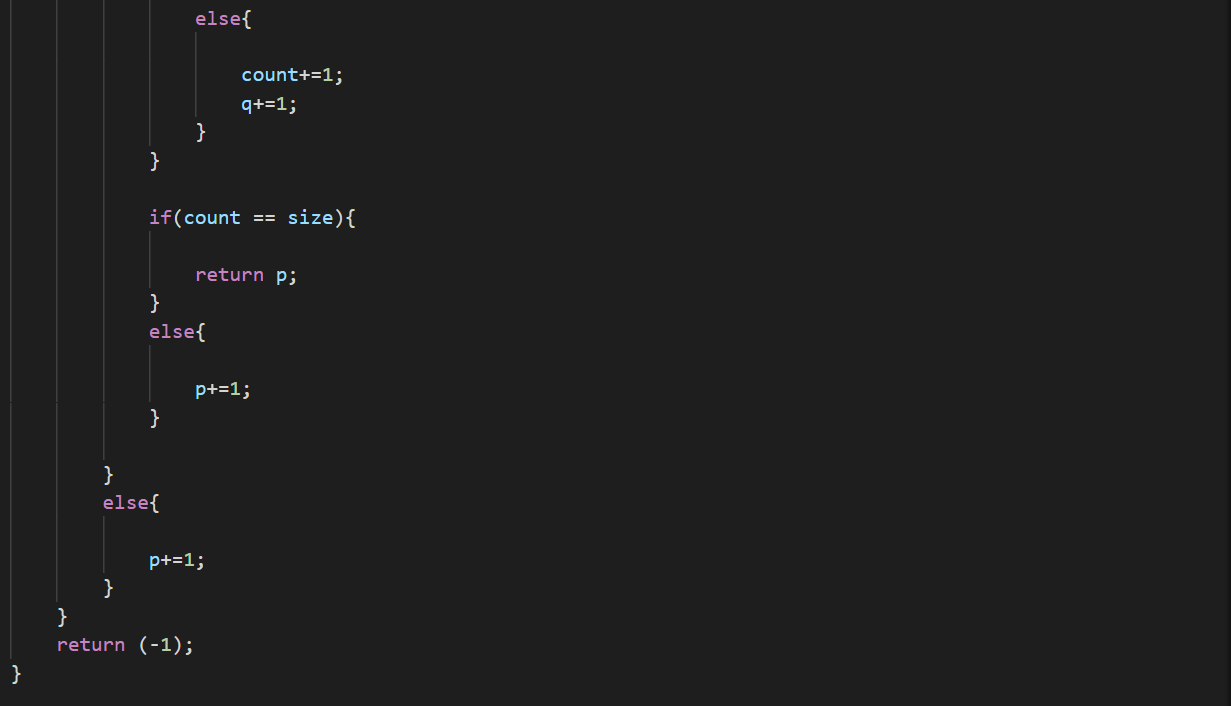




This method uses selection sort to sort the letters of a word. As we can see above, this sort algorithm uses a nested loop structure. The number of times the outer loop runs is K-1 times and the inner loop runs for K times. From this we can conclude the big-O notation time complexity of this method to be O(K^2).

**Method ‘anagramOrNot’:**

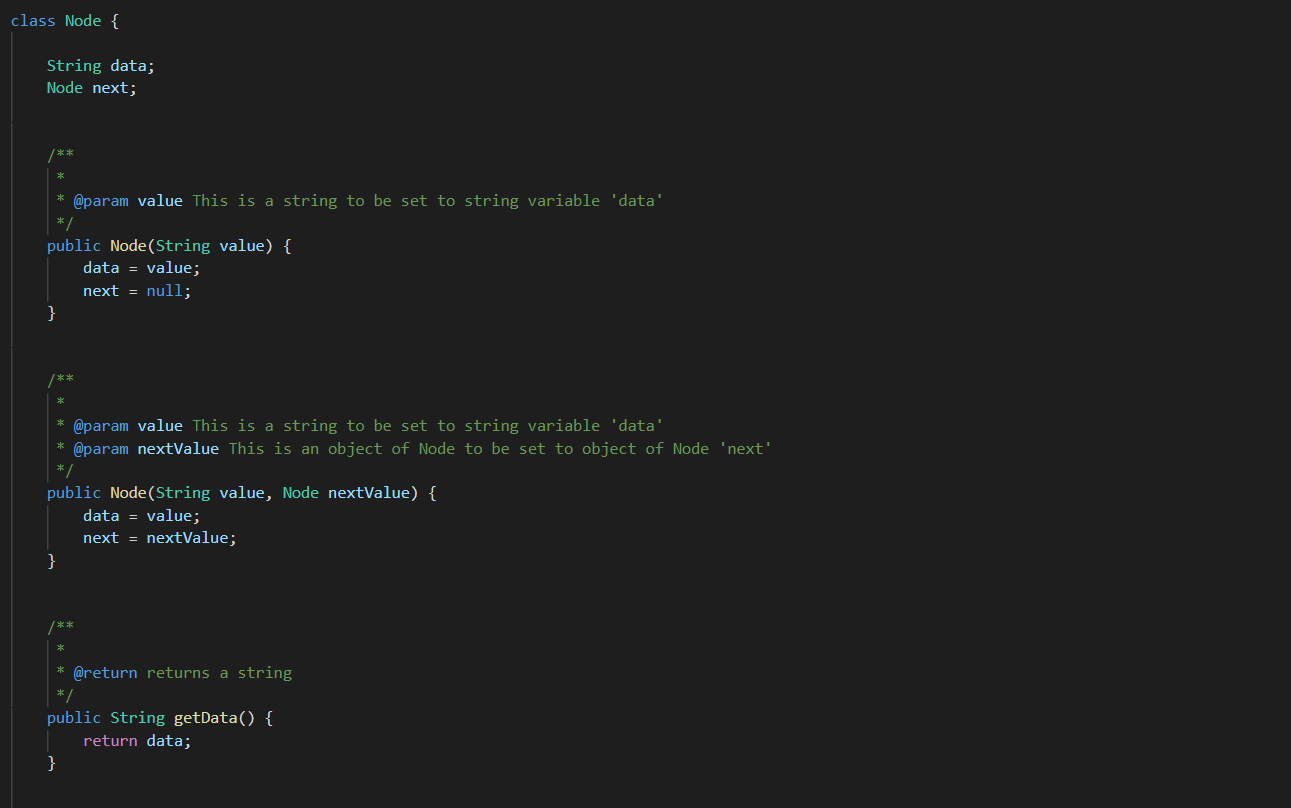


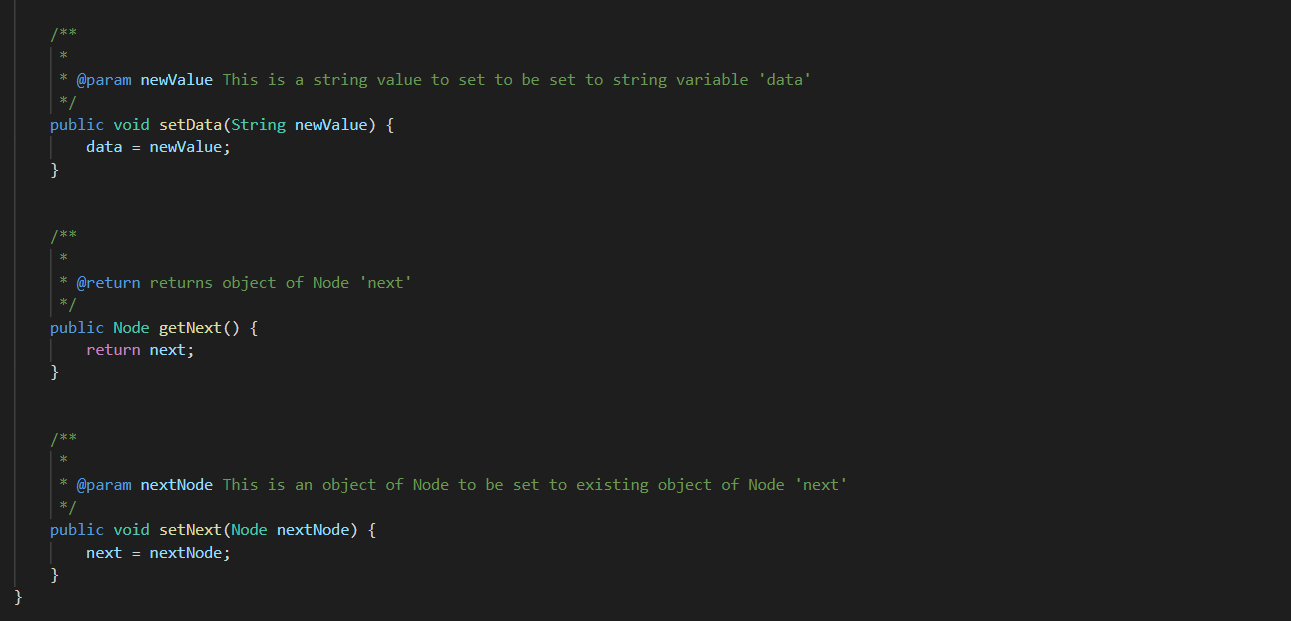


Most of the algorithm operates under a while loop for N times, which is the size of array sent from main, which gives us a big-O notation time complexity of O(N). But we also have to account for the ‘result’ method being called twice from here, which has its own big-O notation time complexity too, as shown above. This means the overall worst-case big-O notation time complexity should be O(K2 \* N). This is worst-case time complexity because in the worst-case scenario the two words will go through the whole anagram searching algorithm and end up not being anagrams of each other.

**Question 2: Let *N* be the number of words in the input word list, and *L* be the maximum length of any word. What is the big-O running time of your program? Justify your answer using both a theoretical analysis and experimental data (i.e. timing data).**

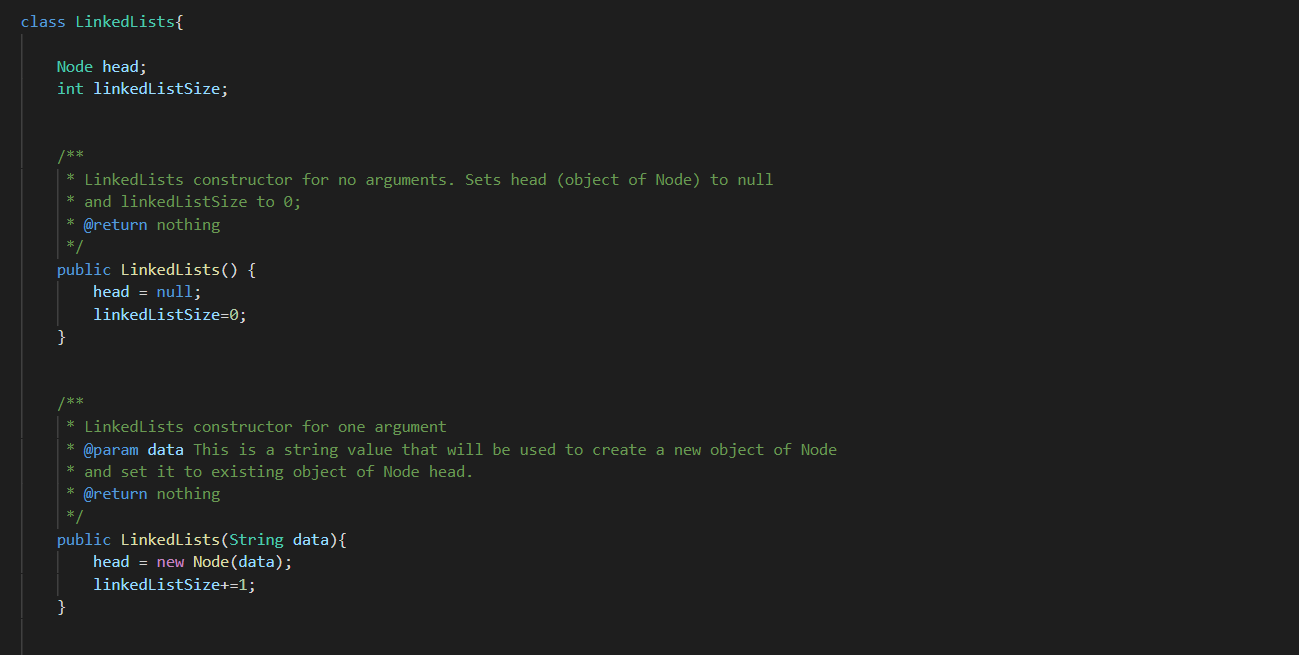
**Class Node**

****

****

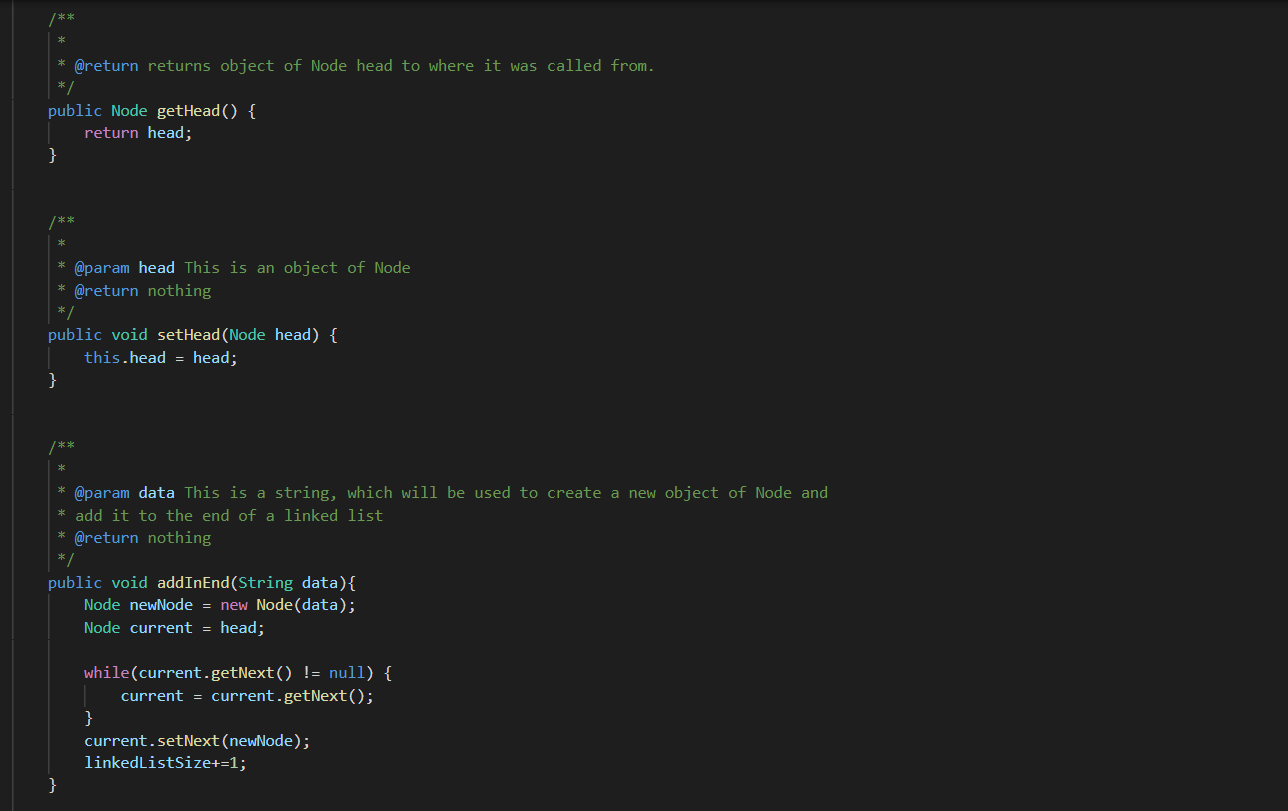
The constructors, getters and setters in class Node will take the same time regardless of the size of the array made in linked lists since it isn’t working under a loop which is why N and L doesn’t affect the time these take. Hence these all have O(1) big-O notation time complexity.

**Class LinkedLists**

****

Big-O notation time complexity: O(1) as N and L don’t affect

Big-O notation time complexity: O(1) as N and L don’t affect

****

Big-O notation time complexity: O(N) because of while loop that depends on number of words in input word list

Big-O notation time complexity: O(1) as N and L don’t affect

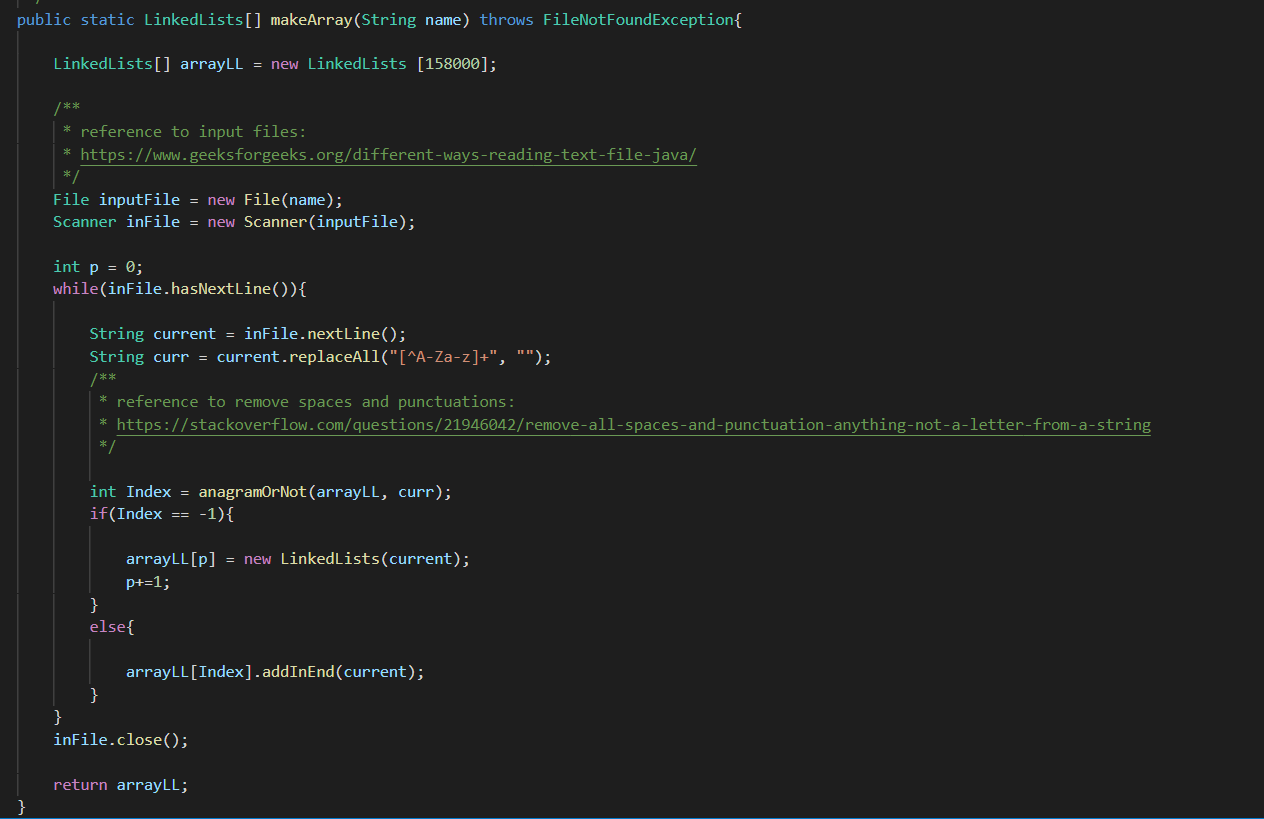
Big-O notation time complexity: O(1) as N and L don’t affect

****

Big-O notation time complexity: O(N) because of while loop that depends on number of words in input word list

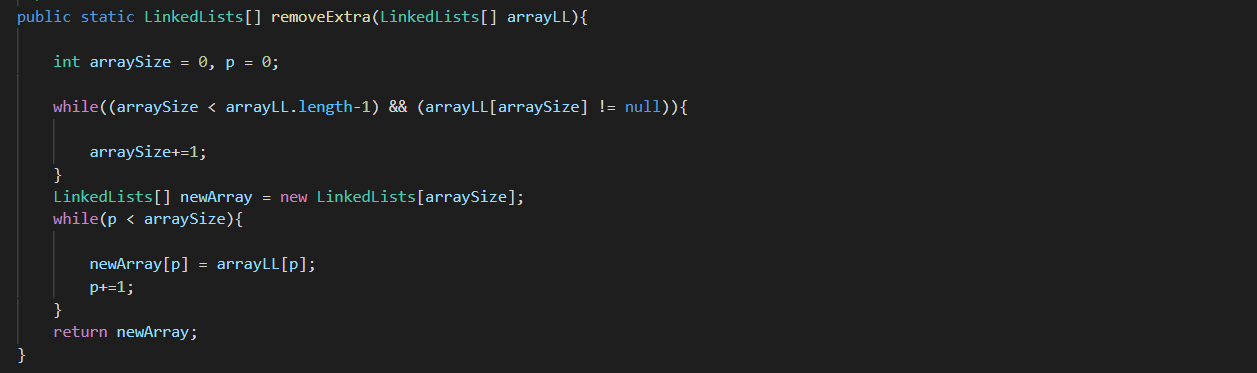
Big-O notation time complexity: O(1) as N and L don’t affect

**Class Assign2**

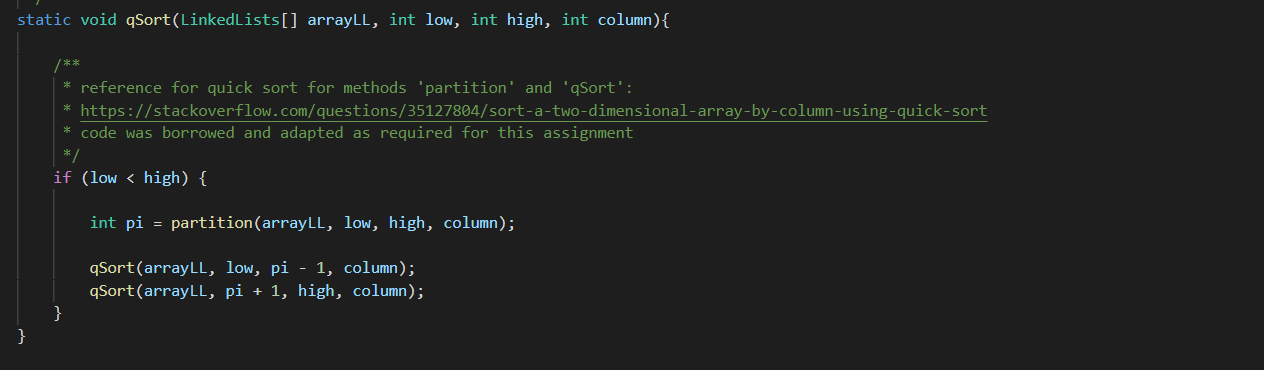
****

Back in question 1 we saw ‘anagramOrNot’ has big-O notation O(L2N). Now here we also have a while loop and a ‘addInEnd’ method from LinkedLists class which has a big-O notation of O(N).

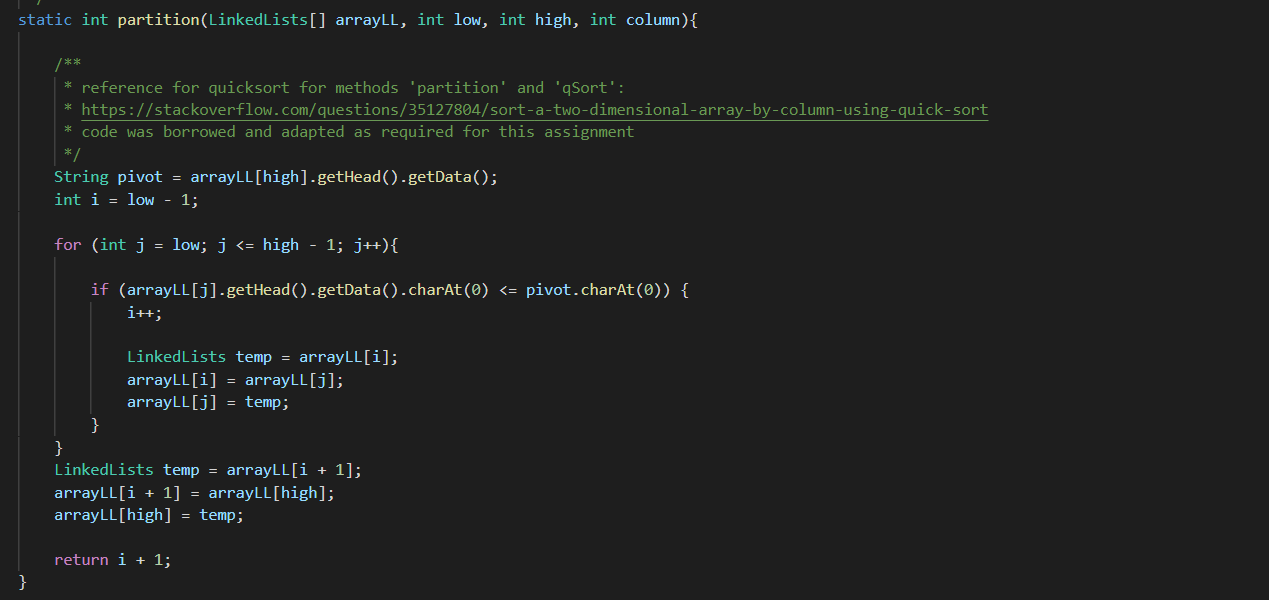
Big-O notation time complexity here will be: O(L2N3)

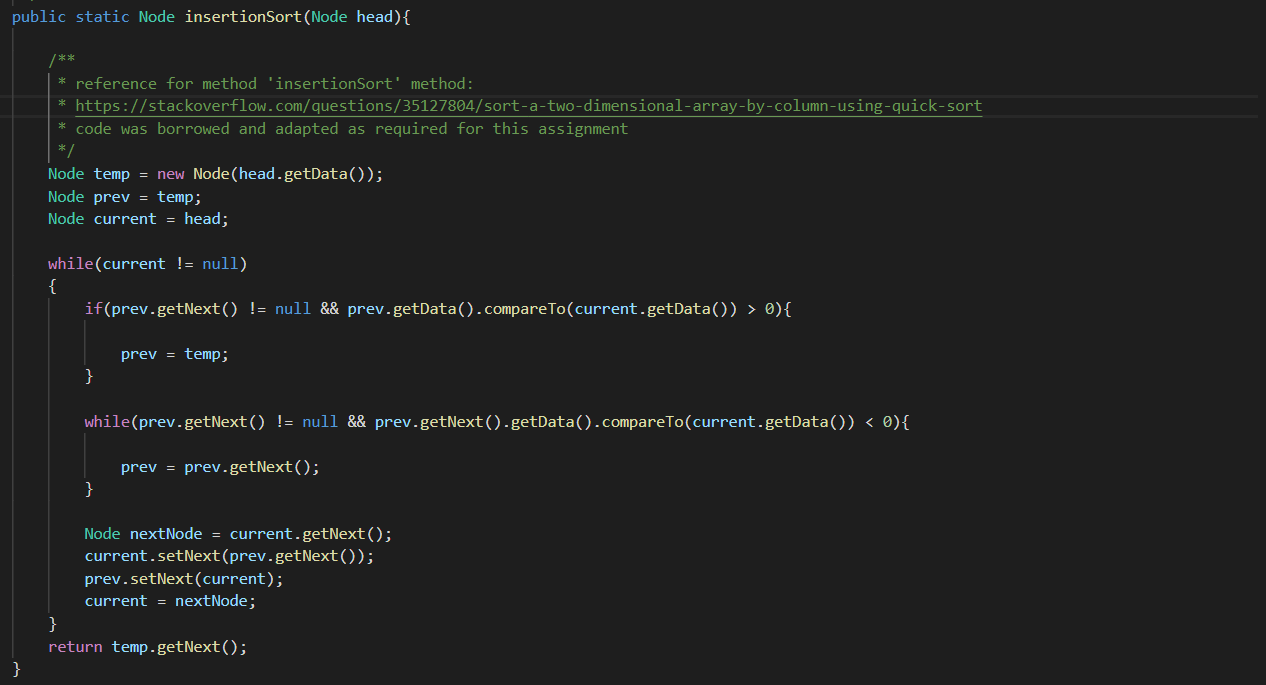
****

Big-O notation time complexity here will be: O(N2)

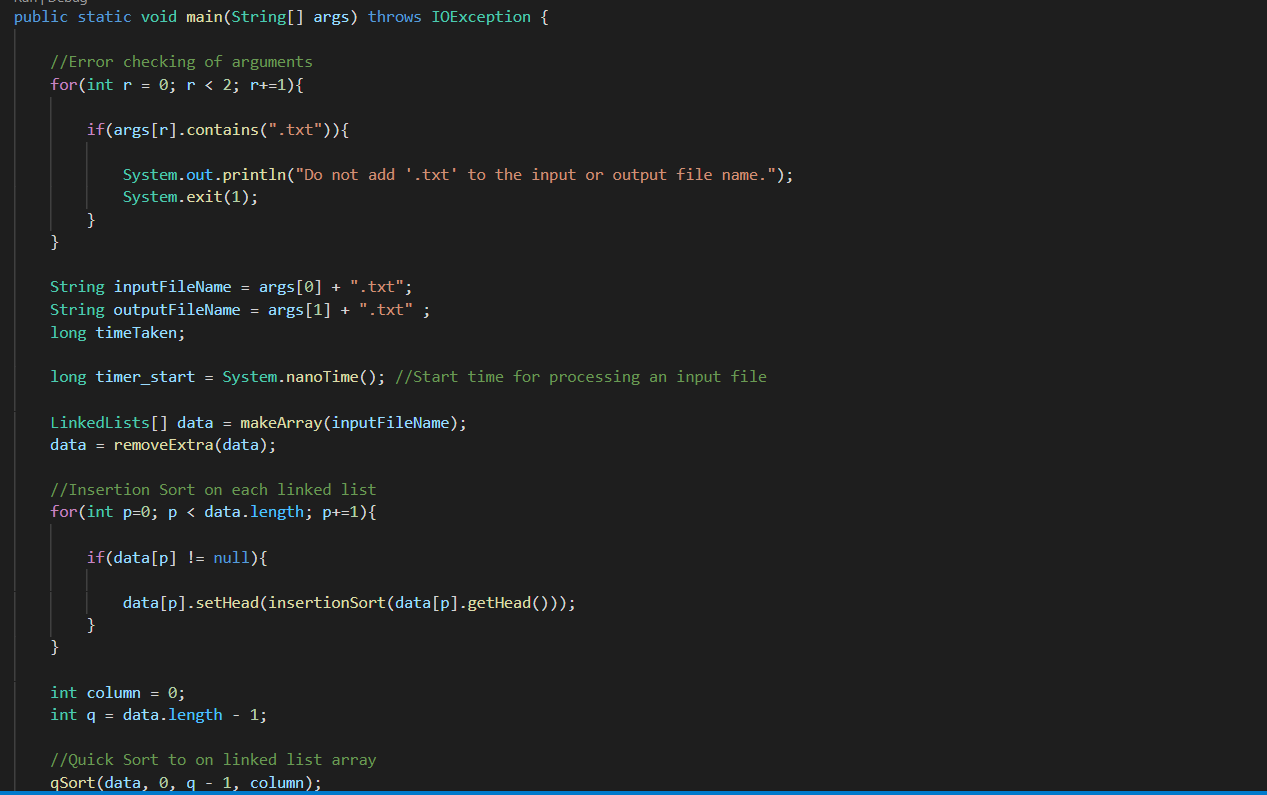
****

Big-O notation for worst case for quicksort algorithm overall: O(N2)

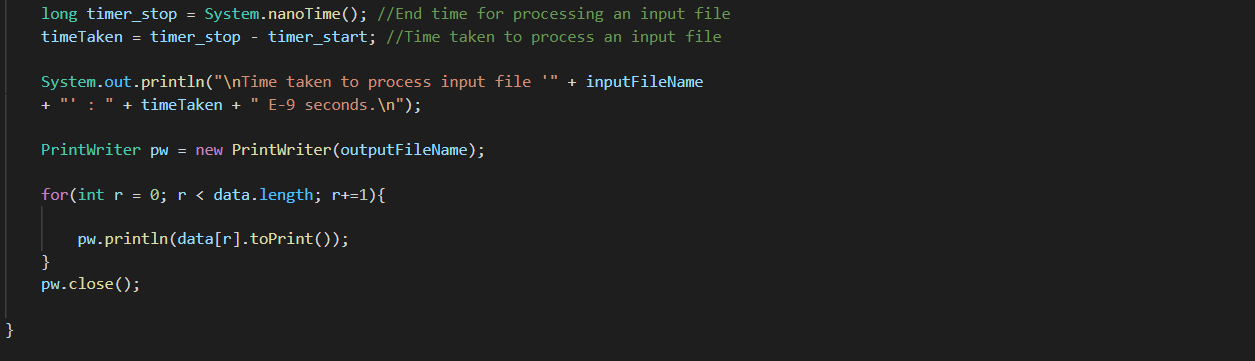
****

****

Big-O notation for worst case: O(N)

****

The main has to work with all the methods in the different classes shown above. We can say the overall worst-case big-O notation for the whole program will be O(L2N3)

****

**Conclusions**

|  |  |  |
| --- | --- | --- |
| **File name** | **Number of elements in file** | **Time in seconds** |
| example\_1 | 8 | 4.46E-02 |
| example\_2 | 40 | 5.48E-02 |
| example\_3 | 25125 | 5.69E+00 |
| small | 24868 | 1.09E+01 |
| medium | 105990 | 2.84E+02 |
| large | 157858 | 3.79E+02 |

Example\_1 and example\_2 are relatively small data so it took lesser time to go through the whole program. However, as seen on the graph we can see a sharp increase in the time taken from example\_3 as the number of words in a file rose up from 40 to 25125, and from this point onwards we can see the graph steadily increasing. From this we can understand that the number of words in a file greatly affects the time taken.