Michael Neas

CSE 2500

Lab 2 Analysis

In order to conduct asymptotic analysis for this lab, a stack-based method entitled StackANSVP.java was compared against BigOhComparison.java. During the comparison, both classes utilized base line implementation of singly linked lists, used user input, were created and run in the same IDE, and processed with the same computer to allow as minimal runtime interference as possible.

**1. Primitive Operations**

To be able to calculate the primitive operations in the algorithm it is important to first note that basic operations will all occur at a constant time in RAM. A primitive operation is any sort of evaluating, assigning, indexing, method calling, or value returning that takes place in an algorithm. For the class StackANSVP.java, primitive operations that can be defined for the main algorithm include:

1. for(int i = 0; i < x.length; i++) which is a 2n operation because of the evaluation and comparison for n operations.

2. while(!S.isEmpty() && S.top() >= x[i]) is a 0 to n operation because the nested while loop will allow for no more than n iterations for any specific i in the for loop.

3. S.pop(); is O(1) operation because its completes one task if while is true

4. if(S.isEmpty()) is 2n operations due to the comparison and the evaluation.

5. System.out.print("- "); is O(1) operation, a return value.

6. else System.out.print(S.top() + " "); O(1) operation of a returned value call.

7. S.push(x[i]); is O(1) because it completes 1 task

As for the comparison, BigOhComparison, which will now be referenced by Tns(N), has these identified primitive operations:

1. for(int i = 0; i < x.length; i++) is 2n because of being a for loop.

2. for(int j = i; j >= 0; j--) also is 2n, which now become n2 in final calculation.

3. if(nS.isEmpty()) is 2n for operating under the call to isEmpty and comparison.

4. System.out.print("- "); is O(1) because it is an output.

5. nS.addFirst(x[i]); is O(1) for being a trivial list operation.

6. else if(x[i] > nS.first()) 2n because of if statement.

7. System.out.print(nS.first() + " "); is O(1) for output.

8. nS.addFirst(x[i]); O(1) for list operation

9. else nS.removeFirst(); is O(1) for the same reason.

**2. Manual Calculation Ts(N) and Tns(N)**

Pseudocode evaluation is required to calculate the maximum number of operations executed as a function of input size. From the previous section’s definitions, each operation has to be counted up to find the final calculation, where a is fastest and b is slowest time.

Stack implementation: a(4n + 4) ≤ Ts(N) ≤ b(5n+4)

No Stack: Tsn(N) ≤ b(4n2 + 4n + 5)

Both methods of implementation are bound between these two functions a(n) and b(n). The reason for the no stack to receive 4n2 is because of the nested for loop which turns the run time into being the product of each for loop due to their dependence, something that the while loop does not possess during the stack implementation.

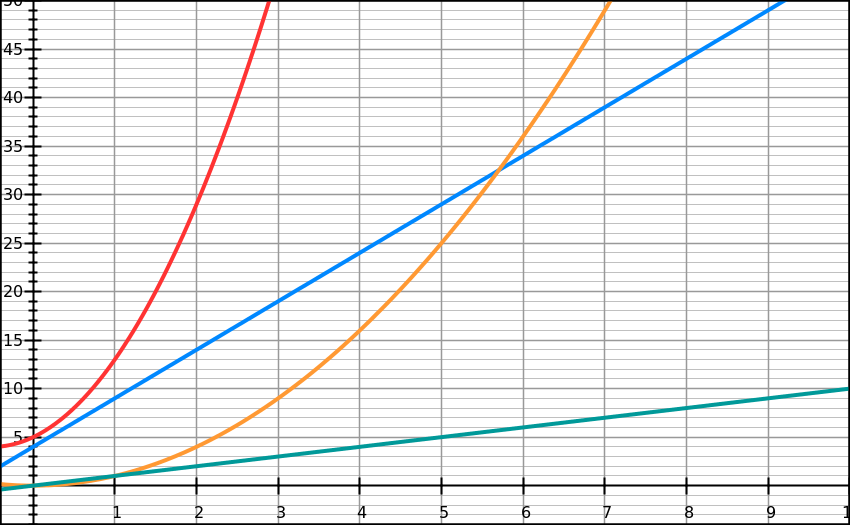
**3. Show that O(N) == Ts(N) and O(N2) == Tns(N)**

From the calculation above Ts(N) can be understood to be in the order of N or O(N) and Tns(N) can be acknowledged as the order of N\*N due to the nested for loops, otherwise known as O(N2). When the bounded functions of the stack time operation calculation resulted in linear time, the conclusion that the process occurs in O(N) time can be made. Big-Oh is determined by the slowest time it takes to complete a program and otherwise if it were not linear, the function would eventually stray outside of the upper bound. This logic holds true for the quadratic upper bound of the nested loop implementation, if it were to be linear it would draw too far outside of the upper bound at so therefore the only way to fit is to say O(N2). Another reason for the equivalence is the fact that the coefficients and following variables are discarded in Big-Oh notation to provide the simplest and smallest expression.

**4. Plot and Conclude Input Threshold Size**

O(N2)[orange] vs O(N) [green]

With upper bounds functions: Ts(N)[blue] vs Tsn(N2) [red]



Time (s)

Input size (N)

In order to determine the input threshold size, the knowledge of the definition of Big-Oh notation is required, which allows the equation f(n) ≤ c\*g(n), for no ≤ n to be used in the calculation. For stack, the equation is:

5n + 4 ≤ cn,

It becomes reduced to: 4/(c-5) ≤ n

Further with the input of c=6, the threshold input size, otherwise known as no = 4. This is the minimum value that should be used when determining Big-Oh algorithm time. Then the nested for loop equation is:

4n2 + 4n + 5 ≤ cn2,

This equation solved out produces the value of c = 3.2 and has a threshold input size of around 4 which is demonstrated in the graph above with the upper bound function at the y axis. C takes the role of any constant rate in the Big-Oh function that provides a multiplication factor for the input size.