Assignment 2 Algorithms Used

Add to Head

public void AddHead(char c)

{

SET newNode(data=c, NextNode = null)

SET count = count + 1

}

Both theoretical and empirical time complexities for this operation are O(1) as it doesn’t need to traverse a variable amount of items in order to insert.

Add to Tail

public void AddTail(char c)

{

SET newNode(data=c, NextNode = null)

IF head is null

{

head = newNode

}

ELSE

{

tail.NextNode = newNode

}

tail = newNode

SET count = count + 1

}

Both theoretical and empirical time complexities for this operation are O(1) as it doesn’t need to traverse a variable amount of items in order to insert.

Delete from Head

public char DeleteHead()

{

IF head is null

{

return null

}

SET char c = head.data

SET newNode = head

head = head.NextNode

newNode.Next = null

SET count = count - 1

return c

}

Both theoretical and empirical time complexities for this operation are O(1) as it doesn’t need to traverse a variable amount of items in order to delete.

Balance Check

public int CheckBalancedParentheses(string inputString)

{

SET degree = 0

FOREACH(character in inputString)

{

IF character = "("

{

PUSH character on linked list

}

IF character = ")"

{

IF return chacter from POP operation = "("

{

SET degree = degree + 1

}

}

}

return (degree + remaining count of linked list)

}

Traversing the input string to check characters is a given O(n) for both empirical and theoretical time complexities given that the input string is of variable length. The FOREACH loop in this case is O(n) and all operations within it are O(1) since they use the methods described previously. This exact same style of parentheses checking is used both in the StackCheckBalancedParentheses and QueueCheckBalancedParentheses classes.

Converting Queue to “Stack”

public QueuePush(char c)

{

SET queue1 = new queue

SET queue2 = new queue

queue1.Enqueue(c)

FOR each item in queue1

{

FOR 1 minus the count of queue1

{

move head of queue1 to tail of queue1

}

move head of queue1 to tail of queue2

}

FOR each item in queue2

{

move head of queue2 to tail of queue1

}

}

I don’t know of any theoretical time complexity for converting a queue to a stack using two queues however I think that my implementation of this function is only going to be worse by a constant. My empirical time complexity ended up being (n(n-1)+n)/2 which is O(n^2). I saw some people online implementing a stack with only one queue however I couldn’t get that working and I think that implementation would still be O(n^2). Since this is the only algorithm that isn’t O(1), this is the only one that has an associated graph of empirical time complexity. Since I cannot find a theoretical time complexity of this operation, this only shows the empirical complexity that my implementation has.