Q1

This code will shorten the representation of that string by adding the number of consecutive characters

```
Function shortenString (Argument One){
    If argument one is empty
        return “empty”
    initialize StringBuilder str with argument one
    initialize count to one
    initialize n1 to 0
    initialize n2 to 1
    while true{
        try{
            while character at index n1 of str is equal to character at index n2 of str{
                delete the character at index n2
                add one to count
            }convert count to string and insert to str at index n1+1
            set count equal to 1
            add one to n2
            set n1 equal to n2
            add one to n2
            if n1 equals to length of str -1
                break from the loop
        }catch String index that are out of bound{
            convert count to string and insert to str at index n1+1
            convert str to string and replace all character that have “1” to “”
            return str
    }
```

```
        convert str to string and replace all character that have “1” to “”
        return str
}
```

i) Time complexity is O(n)
ii) Space complexity is O(1)

Q2)

```
Function findSmallestDifference(Array of integer num[]){
        Initialize n to length of array of integer
        Initialize difference to maximum value of integer
        Initialize index1 to 0
        Initialize index2 to 1
        Initialize num1 to 0
        Initialize num2  to 0
        initialize i to 0
        initialize j to 0
        For i less than n - 1, i ++ {
                For j less than n, j++{
                        If num[i]-num[j] less than difference{
                                Set index1 to i
                                Set index2 to j
                                Set num1 to num[i]
                                Set num2 to num[j]
                                Set difference to num[i]-num[j]
                        }
                }
```

```
        }
        Print “respective index and values”
}
function findLargestDifference(Array of integer num[]){
        Initialize n to length of array of integer
        Initialize difference to minumum value of integer
        Initialize index1 to 0
        Initialize index2 to 1
        Initialize num1 to 0
        Initialize num2 to 0
        initialize i to 0
        initialize j to 0
        For i less than n - 1, i ++ {
                For j less than n, j++{
                        If num[i]-num[j] more than difference{
                                Set index1 to i
                                Set index2 to j
                                Set num1 to num[i]
                                Set num2 to num[j]
                                Set difference to num[i]-num[j]
                        }
                }
        }
        Print respective index and number
```

ii) The code go through every array of elements and compare them one by one. By using for loop, it can update the num, minimum and maximum difference and the index to the variable. Lastly, print those index and values in the output

iii) The time complexity of my solution is $O(n^2)$, There are two loops that will take in input two times. Which makes it n*n. For each variable initialization is just a constant, but since big O notation takes in the biggest n, therefore is $O(n^2)$

iv) Since stack growth is equivalent to space complexity and the number of data input remains constant, the stack growth remains the same. The space complexity wasn't affected by data input at all.

Q3)

a) $n^{15}\log n + n^9$ is $O(n^9 \log n)$

false, big O notation supposed to take in worst case scenario

The correct solution is $O(n^{15}\log n)$

b) $15^7n^5 + 5n^4 + 8000000n^2 + n$ is $\Theta(n3)$

False, Big O is n^5 while big Omega is n. Big theta can't be n^3

c) n^n is Ω (n!)

True. $n^n > n!$ based on function hierarchy.

d) $0.01n^9 + 800000n^7$ is $O(n^9)$

True, Big O always assume the worst case, and n^9 is the dominating term

e) $n^{14} + 0.0000001n^5$ is $\Omega(n^{13})$

True, since $O(n^{14}) > \Omega(n^{13}) > \Omega(n^5)$

f) n! is $O(3^n)$

False, the dominant term is n!. It is O(n!) instead