Gabriel Emerson (gte0002)

COMP 3270-002

Homework 1

Due 2/2/21

1. Remember: Computer can compute 2 * 10^7 steps in one second
    a. O(n) – to solve problem with size 20 * 10^7
    This means 20/2 seconds gives us the answer since the * 10 is to the same power
    O(n) takes **10 seconds** to solve this problem

    b. O(nlogn) – to solve problem with size 20 * 10^7

    It will take 60 * 10^7 steps since 20log(20) is 59.9146

    This means 60/2 gives us the answer since they are to the same power

    O(nlogn) takes **30 seconds** to solve this problem

    c. O(n^2) – to solve problem with size 20 * 10^7

    It will take 20 * 10^14 steps

    This means (20 * 10^7)/(2 * 10^7) gives us the answer

    O(n^2) takes **8.192 * 10^10 seconds** to solve this problem (A VERY LONG TIME)

2. State problem specifications in Inputs, data representation, and outputs
    a. Inputs –
    We would first need to know where the user is.
    We also need the locations of the five parking spots either given by the user (maybe another user) or by the API
    Also need to know where the user wants to park
    b. Data Representation –

    We need the google map API to know where the roads are to be able to guide the user directly to the parking spot located

    We also need the API to be able to state the longitude, latitude of the given spots found

    We need to know where there are open spots either given by another user or perhaps a different API used to say whether or not a parking spot is open

    c. Desired outputs –

We need to know where the parking spot is and that it is available

We need to know how to get to the closest/best parking spot chosen

If pinged by AU parking app, we need to know that there is a parking spot open

3. Test to see if a_i is equal to a_(n/2), if it is return a_(n/2) and we are done

   If not test if a_i > a_(n/2), if yes then we only need to deal with the elements greater that a_(n/2), if not then we only need to deal with the elements less than a_(n/2)

   Then repeat for whichever direction was chosen (either greater than or less than a_(n/2))

   This will lead to either finding a_i when testing if it equals the middle element, or will be given when a_i is one element greater or less than the current middle element.

4. Answer the following:
   a. Output when array = a

      The output is 55

   b. Output when array = b

      The output is 0 since none of the numbers in the array are bigger than the original number, 0

   c. Output when array = c

      The output is 0

   d. Output when array = d

      The output is 9

   e. What does output return when input contains all negatives

      It returns 0, since none of the numbers in the input are greater than 0, which sum and max are originally set to.

   f. What does output return when input contains all non-negatives

      It will return the sum of all those numbers combined, since it will always be greater than 0, they will be added to the overall sum

5. Calculate approximate complexity

Step	Big-Oh Complexity
1	O(1)
2	O(1)
3	O(n)
4	O(1)
5	O(n^2)
6	O(1)
7	O(n)
8	O(1)
9	O(1)

6. Calculate detailed complexity

Cost of each execution	Total # of time's executed
c1	1
c2	1
c3	n+1
c4	n
c5	n(n+1)
c6	n(n)
c7	n
c8	n
c9	1

T(n) = c1+c2+c3(n+1)+c4(n)+c5[n(n+1)]+c6[n(n)]+c7(n)+c8(n)+c9

= c+c(n)+c(n+1)+c[n(n)]+c[n(n+1)]

7. **This Algorithm is incorrect since it does not specify if it is comparing on the same element in the file. This means every time i=f[1] and j=f[1] (or for any other spot in the file), it will count at least one since it is on the same element, they will equal the same number.**

8. Base cases: (n must be a non-negative integer)

   n=0; Then n will be returned as 0

   n=2; Then function will be returned with n as input and get, 5g(2-1)-6g(2-2)

   Since n is always a non-negative number, we can assume k to be k+2, since that will always trigger our function formula.

   This gives 5g(k+2-1)-6g(k+2-2) which simplifies to 5g(k+1)-6g(k)

   This shows k will never be a negative number when k+2 is applied

   If intended to compute 3^n-2^n the same induction applies

   Base cases: n=0; This would make the function 3^0-2^0 with would equal 0, which is still a non-negative integer.

   For any number k, 3^k-2^k, you can add the numbers together and leave the exponent the same

   Since n is always the same base, we can always add 3 and -2

   This gives 3-2=1, and put the exponent back = 1^k

   This shows for all n, the equation 3^n-2^n, will always be equal to 1^n

   Since n is always the same base, we can always add 3 and -2

9. Prove by Loop invariant

   Initialization: Before the first iteration of the loop with i=2, we notice the sorting of this array does not matter (since it iterates over the entire array and compares each element). We know we will iterate over the entire array, because the max is already set to the first element, and i begins at the second and does not stop until n.

   Maintenance: It will hold for i-1 because at base case 2, i-1 then is equal to the first element which was originally set as max before entering the loop. Then at i+1, we will continue to iterate through the loop until it has reached n, then we exit the loop.

   Termination: Showing that we iterate through the entire array, and can successfully compare elements i-1, i, and i+1, we can say that this algorithm will compare all the elements in this array.

10. Prove by Loop invariant

Initialization: Before the first iteration we notice to start with whatever t is introduced in the input. This means to begin we must have any positive integer. We start with our case of t2^k+m. We know this n will work because first case k=0, this means the rest of that formula will be t+m. We then see this will give us 2m, that later gets divided by two, and equals m. This shows the first case works.

Maintenance: Now we show that when we enter the ith iteration, we still get t2^i+m. And since k is incremented on every iteration of this loop, we can assume the same for i+1. Since t must be a positive number, and i will be a positive number as well, this means we can continue to search through the positive array and get m.

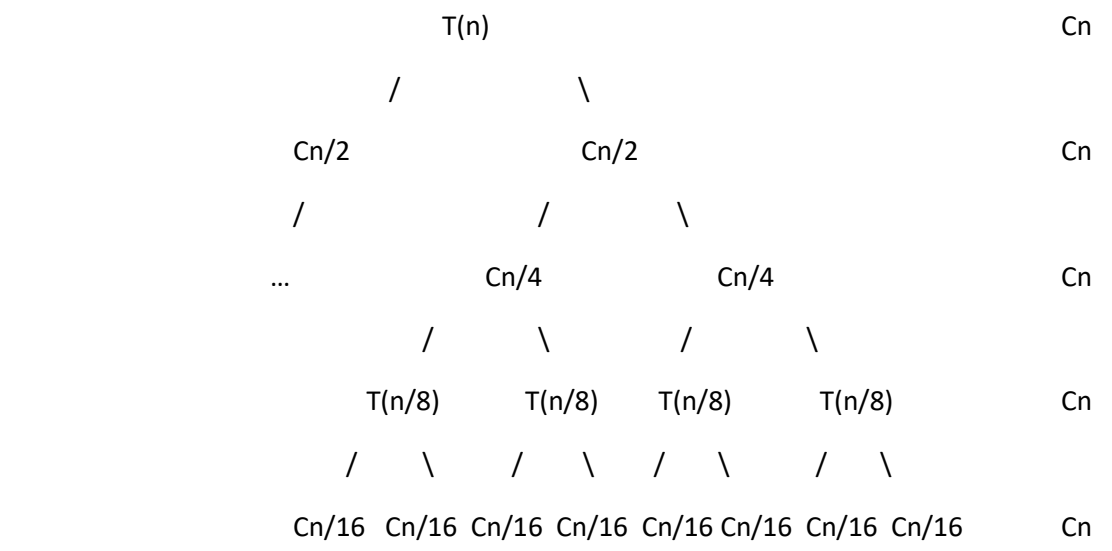
Termination: Since we have shown the base case for the while loop, and shown how we can continue to look through all the elements at i, or i+1, in the array, we can conclude that this is a working algorithm that will search through the array to find m.

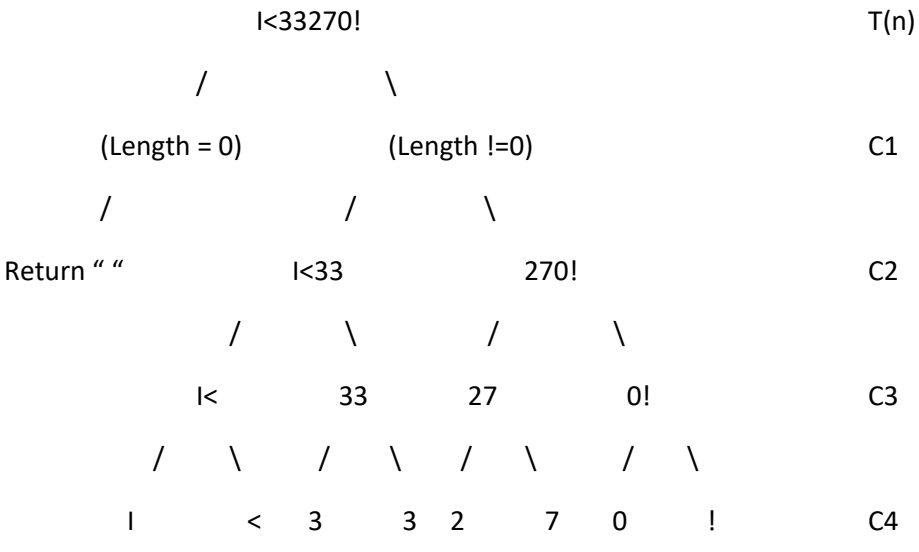
11. Algorithm design
    a. -

If the string is empty or a single character, return in unchanged
Otherwise
Remove the first character
Reverse the remaining string
Add the first character above to the reversed string
Return the new string

```
String A[p..q]

Int i, j, n;

Int length = A.length();

String reverseStringRecursively(string A[p..q])

    If (length == 0)  //stop recursion

        return " ";

    else

        n = length/2

        for (i=0; i<=n; i++)

            j=length-i

            return reverseStringRecursively(swap(A[i], A[j]))
```

b. –


T(n) Cn
/ \
Cn/2 Cn/2 Cn
/ / \
… Cn/4 Cn/4 Cn
/ \ / \
T(n/8) T(n/8) T(n/8) T(n/8) Cn
/ \ / \ / \ / \
Cn/16 Cn/16 Cn/16 Cn/16 Cn/16 Cn/16 Cn/16 Cn/16 Cn


I<33270! T(n)
/ \
(Length = 0) (Length !=0) C1
/ / \
Return " " I<33 270! C2
/ \ / \
I< 33 27 0! C3
/ \ / \ / \ / \
I < 3 3 2 7 0 ! C4