Tanish Arora SID : 862012234
CS 141, Fall 2018 Assignment 1
Posted: September 28th, 2018 Due: October 12th, 2018, 11:59pm

Notice

- Include your full name and student ID in your solution
- You are expected to work on this assignment on your own
- Use pseudocode, Python-like or English to describe your algorithms. Absolutely no C++/C/Java
- When designing an algorithm, you are allowed to use any algorithm or data structure we explained in class, without giving its details, unless the question specifically requires that you give such details
- Always remember to analyze the time complexity of your algorithms
- Homework has to be submitted electronically on iLearn by the deadline. Late submission allowed for 20% penalty for a calendar day.

Problem 1. (20 points) The Fibonacci numbers are the numbers in the following integer Sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,...

In mathematical terms, the sequence F n of Fibonacci numbers is defined by the recurrence relation:

$F_n = F_{n-1} + F_{n-2}$, *where* $F_0 = 0$, $F_1 = 1$

Discuss the correctness of the following algorithms that calculate Fibonacci numbers:

1. Algorithm 1

```
fib1 ( n )
{
        if ( n!=1)
            return   fib1 ( n−1) + fib1 ( n−2);
        else
            return   n ;
 }
```

Answer algorithm: The algorithm is right for all numbers which are not 1, i.e., all numbers greater than 1, but the algorithm is not correct when *n = 0* , there's no result into the fibonacci theorem or when the n is less than zero. .

2. Algorithm 2

```
fib2 ( n )
{
        integer array f[ n +1];
        f [ 0 ] = 0 ;
```

```
        f [ 1 ] = 1 ;
        for  ( i  =  2 to n )
        f [ i ]  =  f [ i −1] + f [ i −2]
        return  f[ n ] ;
}
```

Answer algorithm: The above given algorithm is correct, since it works for every other "n" value. It also terminates as we can see from the "for" loop as it reaches the "n" iterations.

3. Algorithm 3

```
 fib3 ( n )
{
        a = 0
        b = 1
        if ( n = 0 )
          return a
        i = 1
        while ( i < n )
        {
                c = a + b
                b = c
                a = b
         }
          return b ;
}
```

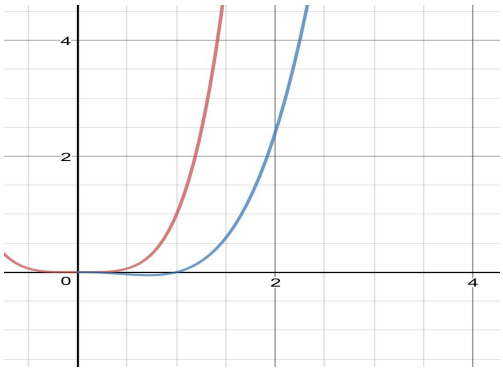
Answer algorithm: The above given algorithm is incorrect for all the values of n to be greater than 1, as if you see in the while loop, ti should have been ≥ n to make the algorithm correct and in order to terminate it.

Problem 2. (20 points) Is the following statement true or false?

$$n^4 = \Omega(n^3 \log n)$$

Justify your answer using the basic definition of the Ω-notation

Answer 2.



Blue: - $n^3 \log n$
Red: - n^4

As the definition of the Ω notation states that for every $f(n)=\Omega(g(n))$, $g(n)$ is the asymptotic lower bound of $f(n)$. As we can see in the graph that $n^3 \log(n)$ is an

asymptotic lower bound of n^4. Hence, by the definition the above given statement is true.

Problem 3. (20 points) Give a tight bound (using the big-theta notation) on the time complexity of following method as a function of n. For simplicity, you can assume n to be a power of two.

Algorithm Weird Loop (n : **integer**)
```
 for (i = n/2; i ≤ n; i = i + 1)
    for (j = 1; j ≤ n; j = 2j)
      k ← 1
   while k ≤ n do
      k ← 2k
```

Answer 3.

```
for (i = n/2; i ≤ n; i = i + 1)              …….      O(n/2) ….. O(n)
    for (j = 1; j ≤ n; j = 2j)               …….. O(log n)
      k ← 1
      while k ≤ n do                         …… O(log n)
        k ← 2k
```

Total complexity T(n) = $n * log(n) * log(n) = n \log^2(n)$. Since all the loops are nested together

Problem 4. (20 points) Order the following list of functions by the big-Oh notation, i.e., rank them by order of growth. Group together (for example, by underlining) those functions that are big-Theta of one another. Logarithms are base two unless indicated otherwise.

$3n+2$	n^3+n^2	$\log^2 n$	$n!$	2^{2n}	4^{n-1}
$2^{\log n}$	$2\sqrt{2 \log n}$	$\sqrt{n}$	n^3	1	$3^{n/3}$
$(n + 1)!$	2^{2n+1}	e^n	$n^{\log \log n}$	$\log n$	$(\log n)^2$
2^n	n^{3n}	$4^{\log n}$	$4n^2/\sqrt{n}$	$\sqrt{\log n}$	$n \log n$

Answer 4 :

a) $2^{2n} = 4^{n-1} > n! > n^3+n^2 = \log^2 n > 3n+2$

b) $3^{n/3} = 2\sqrt{2 \log n} = 2^{\log n} > \sqrt{n} = n^3 > 1$

c) $2^{2n+1} = e^n > (n+1)! > n^{\log \log n} > \log n = (\log n)^2$

d) $2^n = n^{3n} = 4^{\log n} > 4n^2/\sqrt{n} > n \log n > \sqrt{\log n}$

The reasoning i used here was the basis on the lower bound facts like
Exponentials > factorials > linear > log > constant (constant being having the lowest bound of all)
And the reason i have the equality signs is that I grouped them together by their big theta importance.

Problem 5. Given n non-negative integers a1, a2, ..., an, where each represents a point at coordinate (i, ai). n vertical lines are drawn such that the two endpoints of line i is at (i, ai) and (i, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.

1. Solve the problem with brute-force approach. What is the time complexity of this approach?
2. Is there any better solution? If so, describe your solution with time complexity analysis.

Answer 5 :

i) **Brute -force:** Since we have to select 2 lines to make a container to fill up water

So, choosing two possible lines to make the maximum area in order to find the one with maximum volume will give us the output. Hence the time complexity would be $O(n^2)$

ii) According to me the best way to solve the above given problem would be "**Divide and Conquer"** algorithm. It will be a recursive algorithm from the fact that we will be using "n" number of bars(rectangular) in order to find the maximum area.

Generic Time complexity of a recursive algorithm :

$T(n) = T(n-1) + O(n \log(n))$ (we can find n number of bars in an array from 1,2,3…. n by log n and since its a recursive algorithm we will use n number of those relations, hence O(nlog n). Also, I chose n-1 since we need to find maximum area and the maximum number of total bars reaches "n-1")

Which will give the Time complexity of the "Divide and Conquer" algorithm would be O(n log n) which is way better than brute -force algorithm which states $O(n^2)$