Name: YOUR NAME GOES HERE____________________________

COSC 40403 - Analysis of Algorithms: Fall 2018: Homework 2

Due: 23:59:59 on September 11, 2018

Question	Points	Score
1	5	
2	5	
3	5	
4	5	
5	5	
6	5	
Total:	30	

1. (5 points) Show that $f(n) = n^2 + 3n^3 \in \Theta(n^3)$. That is, use the definitions of O and Ω to show that $f(n)$ is both $O(n^3)$ and $\Omega(n^3)$.

Solution:

Big-O : O(g(n)) = { f(n) : there exist positive constants c_1 and n_0 such that $0 \leq f\text{(n)} \leq c_1.\text{g(n)}$ for all n $\geq n_0$ }.

$3n^3 + n^2 \leq c_1.n^3$ —1—

Big-Ω : Ω(g(n)) = { f(n) : there exist positive constants c_2 and n_0 such that $0 \leq c_2.\text{g(n)} \leq f\text{(n)}$ for all n $\geq n_0$ }.

$c_2.n^3 \leq 3n^3 + n^2$ —2—

Big-Θ : Θ(g(n)) = { f(n) : there exist positive constants c_1,c_2 and n_0 such that $0 \leq c_2.\text{g(n)} \leq f\text{(n)} \leq c_1.\text{g(n)}$ for all n $\geq n_0$ }.

From 1 & 2, we get:

$c_2.n^3 \leq 3n^3 + n^2 \leq c_1.n^3$ —3—

$c_2 \leq 3 + 1/\text{n} \leq c_1$

So, for n $\geq$ 1, $c_1 \geq 4$ & $c_2 \leq 3$

equation 3 always holds true and satisfies the conditions for Big-Θ.

Hence, $f(n) = n^2 + 3n^3 \in \Theta(n^3)$.

2. (5 points) Suppose you have a computer that requires 1 minute to solve problem instances of size $n = 1000$. Suppose you buy a new computer that runs 1000 times faster than the old one. What instance sizes can be run in 1 minute, assuming the following time complexities $T(n)$ for our algorithm?

   (a) $T(n) = n$

   (b) $T(n) = n^3$

   (c) $T(n) = 10^n$

Solution: The old computer could do 1000 instances per min. Now, the new one can do 1000*1000 instances per min

(a) $T(n) = n$ — 1,000,000 = n

(b) $T(n) = n^3$ 1000^3*1000 = $10,000^3$, hence n = 10,000

(c) $T(n) = 10^n$ 10^{1000}*1000 = 10^{1003} , hence n = 1003

3. (5 points) Let $f(n)$ and $g(n)$ be asymptotically nonnegative functions. Using the basic definition of Θ-notation, prove that $\max(f(n), g(n)) = \Theta(f(n) + g(n))$.

Solution:

Theorem 3.1:
For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$, if and only if, $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

$f(n) \le f(n) + g(n)$ and $g(n) \le f(n) + g(n)$

So, $\max(f(n), g(n)) = \emptyset(f(n) + g(n))$ (Upper-Bound). –1–

$f(n) + g(n) \le \max(f(n), g(n))$

$\rightarrow 1/2(f(n) + g(n)) \le \max(f(n), g(n))$

So, $\max(f(n), g(n)) = \Omega(f(n) + g(n))$ (Lower-Bound). –2–

Hence, from $Theorem 3.1$, 1 & 2 we have proved that $\max(f(n), g(n)) = \Theta(f(n) + g(n))$.

4. (5 points) Show that the golden ratio ϕ and its conjugate $\hat{\phi}$ both satisfy the equation $x^2 = x + 1$.

Solution: $\Phi = \dfrac{1+\sqrt{5}}{2}$ & $\hat{\Phi} = \dfrac{1-\sqrt{5}}{2}$

Now, $x^2 = x + 1$

$\rightarrow (\dfrac{1+\sqrt{5}}{2})^2$ - $\dfrac{1+\sqrt{5}}{2}$ - $1 = 0$

$\rightarrow \dfrac{1 - 2 * \sqrt{5} + 5 - 2 + 2 * \sqrt{5} - 4}{4} = 0$

Hence proved.

5. (5 points) Prove by induction that the ith Fibonacci number satisfies the equailty

$$F_i = \frac{\phi^i - \hat{\phi^i}}{\sqrt{5}}$$

where ϕ is the golden ratio and $\hat{\phi^i}$ is its conjugate.

Solution: *Proof* :

For n = 0,

$$0 = \frac{1-1}{\sqrt{5}}$$

For n = 1,

$$1 = \frac{(1+\sqrt{5})^2 - (1-\sqrt{5})^2}{\sqrt{5}.2}$$

Assume that

$$F_i = \frac{\phi^i - \hat{\phi^i}}{\sqrt{5}}$$

holds true for n-1 and n. We need to show that

$$F_{n+1} = \frac{\phi^{n+1} - \phi^{\hat{n+1}}}{\sqrt{5}}$$

We know from Fibonacci series,

$$F_{n+1} = F_n + F_{n-1}$$

$$F_{n+1} = \frac{\phi^n - \hat{\phi^n}}{\sqrt{5}} + \frac{\phi^{n-1} - \phi^{\hat{n-1}}}{\sqrt{5}}$$

$$F_{n+1} = \frac{\phi^n - \hat{\phi^n} + \phi^{n-1} - \phi^{\hat{n-1}}}{\sqrt{5}}$$

$$F_{n+1} = \frac{\phi^n + \phi^{n-1} - \hat{\phi^n} - \phi^{\hat{n-1}}}{\sqrt{5}}$$

Since $\phi.\hat{\phi}$ = -1

$$F_{n+1} = \frac{(\hat{\phi}^2 - \hat{\phi}^1)\phi^{n+1} + (\phi^2 - \phi^1)\phi^{\hat{n+1}}}{\sqrt{5}}$$

Since $(\hat{\phi}^2 - \hat{\phi}^1) = (\phi^2 - \phi^1) = 1$ { from Question 4 }

$$F_{n+1} = \frac{\phi^{n+1} - \phi^{\hat{n+1}}}{\sqrt{5}}$$

Hence proved by induction.

6. (5 points) Consider the following algorithm:

```
PRINT-I-J(n)
for (i = 2; i < n; i++)
    for (j = 0; j ≤ n)
        PRINT i, j
        j = j + ⌊n/4⌋
```

(i) What does this algorithm do?

(ii) What is the output when $n = 4$, $n = 16$, $n = 32$?

(iii) What is the time complexity $T(n)$. You may assume that n is divisible by 4.

Solution:

(i) The algorithm prints out pairs of {i,j} where i takes values from {2 to n} and j takes values in a geometric progression with the first value as 0, the ratio of $\frac{n}{4}$ and the last value as n.

For n = 4,	For n = 16,	For n = 32,
loop i = 2	loop i = 2	loop i = 2
i - 2, j - 0	i - 2, j - 0	i - 2, j - 0
i - 2, j - 1	i - 2, j - 4	i - 2, j - 8
i - 2, j - 2	i - 2, j - 8	i - 2, j - 16
i - 2, j - 3	i - 2, j - 12	i - 2, j - 24
i - 2, j - 4	i - 2, j - 16	i - 2, j - 32
loop i = 3	loop i = 3	loop i = 3
i - 3, j - 0	i - 3, j - 0	i - 3, j - 0
....	....	....
i - 3, j - 4	i - 15, j - 16	i - 31, j - 32
loop i = 4	loop i = 16	loop i = 32
i - 4, j - 0	i - 16, j - 0	i - 32, j - 0
i - 4, j - 1	i - 16, j - 4	i - 32, j - 8
i - 4, j - 2	i - 16, j - 8	i - 32, j - 16
i - 4, j - 3	i - 16, j - 12	i - 32, j - 24
(ii) i - 4, j - 4	i - 16, j - 16	i - 32, j - 32

Attached python file to check the outputs.

(iii) Algorithm:

```
n = 32                                  c1          1
for i in range(2, n+1):                 c2          n
  j = 0                                 c3          n−1
  while (j < n + 1):                    c4          (n−1)∗6
    print(i, j)                         c5          (n−1)∗5
    j = j + math.floor(n/4)             c6          (n−1)∗5
```

T(n) = c1*1 + c2*n + c3*n-1 + c4*(n-1)*6 + c5*(n-1)*5 + c6*(n-1)*5 T(n) = $\theta(n)$