CS721
Test 1
Fall 2020

Name: Jogesh Venkata Surya Prakash Devathi
WSU ID: n659a624

1. Let $f(n)$ and $g(n)$ asymptotically positive functions. State whether the following statements are TRUE or FALSE. In case a statement is TRUE, prove it using definitions of O, Ω, Θ, ω, o etc. In case it is FALSE, provide a counter example.

(a) (10 points) $f(n) = O((f(n))^2)$. – **False**

To prove f(n) = O(g(n)) we have lim n→∞ g(n)/f(n) > 0

Here g(n) = $(f(n))^2$ and f(n) = $f(n)$. Let $f(n)$ = 1/n g(n) =$(1/n)^2$

➔ lim n→∞ $(1/n)^2 /(1/n)$ => lim n→∞ (1/n) = 0, which is not greater than 0.

Therefore it is false.

(b) (10 points) $f(n) = \Theta(f(n/2))$. – **False**

To prove $f(n) = \Theta(f(n/2))$ we have lim n→∞ g(n)/f(n) > 0 and lim n→∞ f(n)/g(n) > 0

Here g(n) = f(n/2) , f(n) = f(n)

Let f(n) = 2^n then g(n) = 2^n/2

⇨ lim n→∞ g(n)/f(n) => lim n→∞ 2^n/2/ 2^n => lim n→∞ 1/ (2^n/2) = 0
⇨ lim n→∞ f(n)/g(n) => lim n→∞ 2^n / (2^n/2) => lim n→∞ 2^n/2 = ∞

As it do not satisfy two conditions, it is false.

(c) (10 points) $f(n) = \Omega(\sqrt{f(n)})$.- **False**

To prove Big-omega definition: f(n) = $\Omega(\sqrt{f(n)})$

$\lim_{n\to\infty} f(n)/g(n)$ = c >0

Let us assume that f(n) = 1/n

So, g(n) = √f(n) = 1/ √n

⇨ $\lim_{n\to\infty} f(n)/g(n)$ = lim n→∞ 1/n / (1/ √n) => lim n→∞ 1/ √n = 0.

As conditions fails. Given Statement is false.

(d) (10 points) $\max(f(n), g(n)) = \Theta(f(n) + g(n))$ – **True**

Suppose T(n) = max{ f(n) + g(n)},

⇨ T(n) <= f(n) + g(n)

Therefore we can say that T(n) = O(f(n) + g(n))

Again,

As T(n) >= f(n) and T(n) >= g(n)

Adding two functions, we get T(n) + T(n) >= f(n) + g(n)

⇨ 2T(n) >= f(n) + g(n)
⇨ T(n) >= ½ [f(n) + g(n)]

Therefore, T(n) = ω(f(n)+g(n))

We can say max($f(n), g(n)$) = Θ { f(n) + g(n)} is true

2. Consider the general *k-ary* search algorithm (generalization of binary search) which splits a sorted array into k subsets each of approximately size n/k, and by making $(k-1)$ comparisons, recursively searching in one of these k subsets. Clearly, the recurrence relation can be written as,

$$T(n) = T(n/k) + (k - 1) \qquad (1)$$

Here k is a variable, but it can be a function of n as well.

(a) (10 points) Suppose we set $k = \sqrt{n}$ in equation 1. Find asymptotic running of $T(n)$ in this case.

Set $k = \sqrt{n}$ we get

$T(n) = T(n/\sqrt{n}) + (\sqrt{n} - 1)$

$= T(\sqrt{n}) + (\sqrt{n} - 1) \longrightarrow 1$

Let, $n = 2^m$

Then, $\sqrt{n} = 2^{m/2} \longrightarrow 2$

By using equation 2 in equation 1 we get:

$= T(2^{m/2}) + (2^{m/2} - 1)$

As $T(n) = S(m) = T(2^m)$

$S(m) = s(m/2) + (m/2 - 1) \longrightarrow 3$

Using master's theorem on equation 3:

$T(n) = a\,T(n/b) + f(n)$

$a = 1, b = 2$,

⇨ $f(m) = m/2 - 1 \longrightarrow 4$

$n^{\log_b a} = n^{\log_2 1} = 0$

By using case3 of masters theorem:

f (n) = Ω(n logba + ε) for some constant ε > 0. Alternatively: f(n) / nlogba = Ω(n ε) Intuition: f (n) grows polynomially faster than n logba Or: f(n) dominates n logba by an n ε factor for some v > 0

$f(m) = (2^{m/2} - 1)$

$= \Omega(m^{0+\varepsilon})$

Regularity Condition:

since as it is case-3, $af(m/b) \le cf(n)$

Name: Jogesh Venkata Surya Prakash Devathi
WSU ID: n659a624

1* 2(m/2)/2 ≤ c. 2 (m/2)

Therefore, T(n) = Θ(f(m))

S(m) = Θ(2 m/2)

T(n) = Θ(√n)

(b) (10 points) Now suppose we set $k = \log n$ (where the base of the logarithm is 2) in equation (1). The above recurrence relation becomes,

$$T(n) = T(n/\log n) + (\log n - 1) \qquad (2)$$

Explain why equation 2 cannot be solved using Master theorem.

To apply master's theorem the above equation should be in the form of

$$T(n) = a\,T(n/b) + f(n)$$

Here a = 1, b=log n.

As, b is not constant we can't apply Master's Theorem.

(c) (10 points) One way to solve equation 2 will be to find explicit upper bound (O) and lower bound (Ω) for $T(n)$. Let us concentrate on the upper bound first. From the divide and conquer point of view, the recurrence relation $T(n) = T(n/\log n) + (\log n - 1)$ says that a problem of size n is divided into one sub-problem of size $n/\log n$ and "divide plus combine" step takes $(\log n - 1)$ time. To find an upper bound of $T(n)$ consider a separate divide and conquer algorithm which divides a problem of size n into one sub-problem of size $n/2$ and where the "divide plus combine" step takes $\log n$ time. Let the running time of this algorithm be $T_1(n)$. Since subproblem size $n/2$ is bigger than $n/\log n$ for all $n \geq 4$, $T(n) \leq T_1(n)$, and therefore, $T(n) = O(T_1(n))$. Thus, solving $T_1(n)$ will give an upper bound of $T(n)$. Write down the recurrence relation of $T_1(n)$ and solve for $T_1(n)$. Assume $T_1(1) = 1$. Show your steps.

T1(n) = T1(n/2) + log(n)
Let n = 2^m => n/2 = 2^{m-1}
m = log2 n
T1(2^m) = 2^{m-1} + m
let S(m) = T1(2^m)
S(m-1) = T1(2^{m-1})

Now, T1(2^m) = S(m) = 2^{m-1} + m
S(m) = S(m-1) + m
= S(m-2) + (m – 1) + m
= S(m-3) + (m- 2) + (m – 1) + m
After repeating for m times
S(m) = S(0) + 1 + 2+ 3+ … + m
= S(0) + Θ(m^2)

Assuming S(0) = 1
S(m) = Θ(m^2)
T1(n) = Θ(log2 n)

(d) (10 points) To find a lower bound of $T(n)$ consider yet another separate divide and conquer algorithm which divides a problem of size n into one sub-problem of size $n/\sqrt{n} = \sqrt{n}$ and where the "divide plus combine" step takes log $n-1$ time. Let the running time of this algorithm be $T_2(n)$. Clearly, $T(n) \geq T_2(n)$ and therefore, $T(n) = \Omega(T_2(n))$. Thus, solving $T_2(n)$ will give a lower bound of $T(n)$. Write down the recurrence relation of $T_2(n)$ and solve for $T_2(n)$. Show your steps.

T2(n) = (n/√*n)* + log n – 1
T2(n) = √*n* + log n -1
Let n = 2^m => √*n* = n^1/2 = 2^m/2

- ⇨ m = log2 n
- ⇨ T2(2^m) = 2^m/2+ m-1

Let S(m) = T2(2^m)
S(m/2) = T2(2^m/2)
S(m) = S(m/2) + (m-1)
Using Master's Theorem,
a = 1, b=2, f(m) = m-1
m^(logb a) = m^log2 (1) = m^0 = 1

Now, lim m-> ∞ f(m)/m^(logb a) => lim m-> ∞ m-1= ∞

- ⇨ Ω(mlogb (a+Ɛ))

By case 3 of Master's theorem

Therefore, T(n) = Θ(f(n))

S(m) = Θ(f(m)) = O(m-1) = O(m)

- ⇨ Θ(log2 n)

T2(n) = Θ(log2 n)

3. We have seen that algorithm for finding strongly connected components of a directed graph $G = (V,E)$ works as follows. In the first step, compute DFS on the reverse graph G^R and compute post numbers, then run the undirected connected component algorithm on G, and during DFS and then process the vertices in decreasing order of their post number from step 1. Now professor Smart Joe claims that the algorithm for strongly connected component would be simpler if it runs the undirected connected component algorithm on G^R (instead of G), but during DFS, process the vertices in increasing order of their post number from step 1.

   (a) (10 points) Explain when the algorithm proposed by prof. Smart Joe might produce an incorrect answer.

Name: Jogesh Venkata Surya Prakash Devathi
WSU ID: n659a624

Graph G

$$\begin{tikzcd}
 & 1 \arrow[dl] \\
2 \arrow[dr, bend left] & \\
 & 3 \arrow[ul, bend left]
\end{tikzcd}$$

Graph G^R

$$\begin{tikzcd}
 & 1 \\
2 \arrow[ur] \arrow[dr, bend left] & \\
 & 3 \arrow[ul, bend left]
\end{tikzcd}$$

Here, Professor Smart Joe's algorithm will not work because of the following reasons.

1) In our example, we see that graph with 3 vertices {1,2,3} with edges {1,2}, {2,3}, {2,3}

In the graph.

2) Strongly connected component {2,3} is in the graph.

3) For Graph , G^R we will have 3 vertices with edges {2,1}, {2,3} , {3,2}

4) We can a possibility of node starting at node 2, needs to explore 3 before 1.

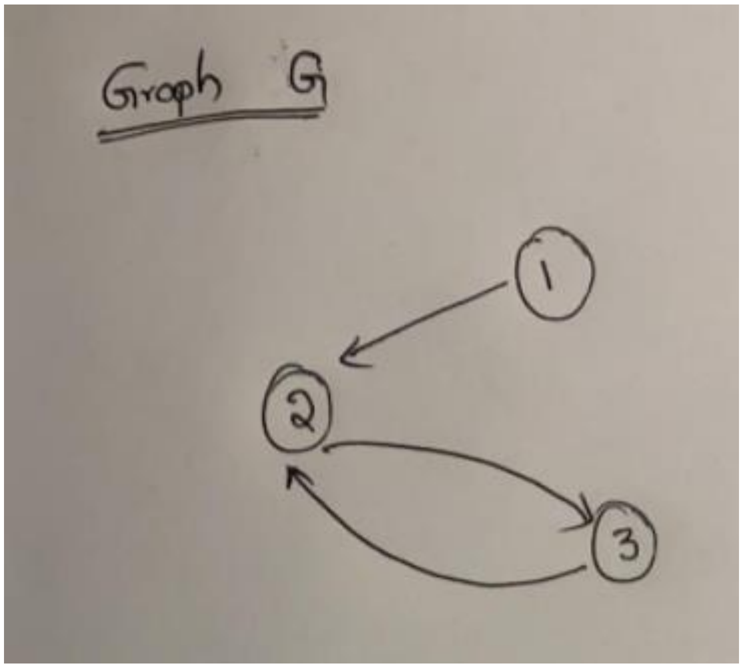
5) By this we can say Post value of 3 is lower that of 1 and 2.

6) While performing DFS starting from 3, we can reach all vertices, but in this example it is not possible as only 2 and 3 are strongly connected.

So, by this we can say professor Smart Joe's thinking is incorrect because this algorithm will return the graph has a single strongly connected component, whereas only 2 and 3 are strongly connected

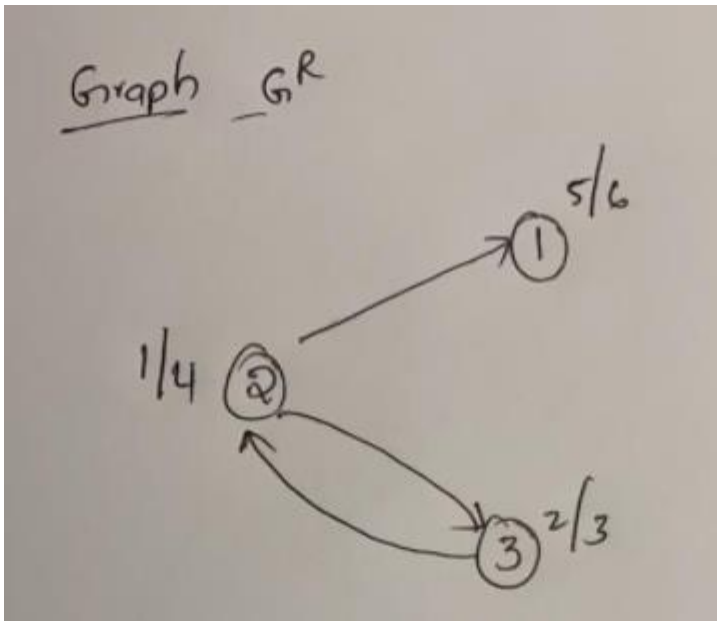
(b) (10 points) With an example, (along with post numbers) show that professor Smart Joe is wrong.

From above assumption Graph G is

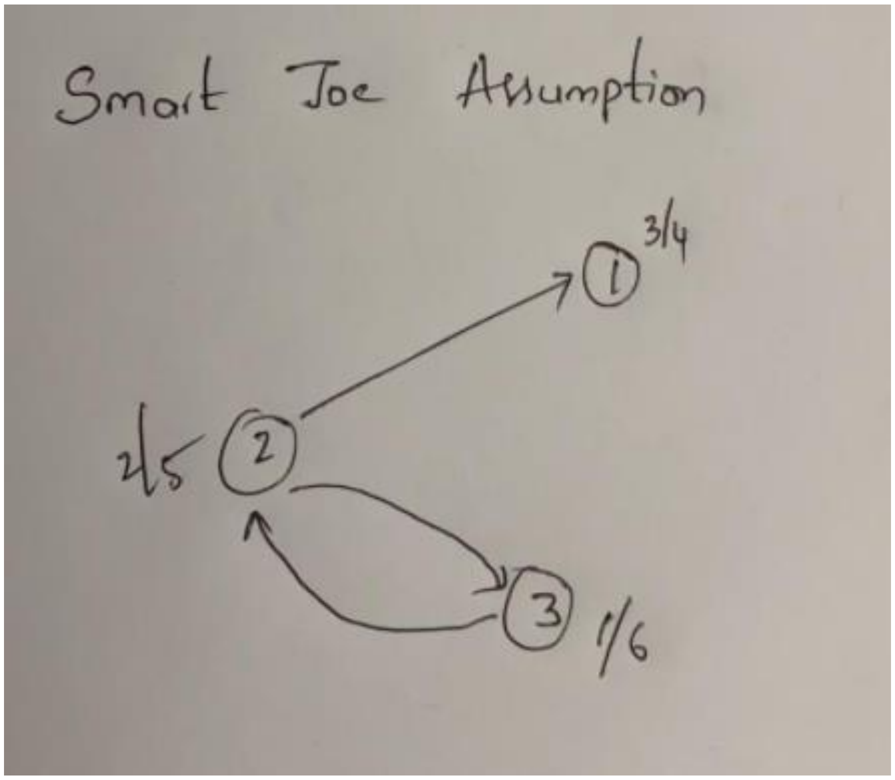


Name: Jogesh Venkata Surya Prakash Devathi
WSU ID: n659a624

For the G^R starting from Node 2 the Pre and Post numbers are as shown in image



Based on the assumption of increasing order on G^R We will explore 3 followed 2 and then 1 Below is the snapshot Starting with node 3



Therefore, starting from Node 3, we will be able to reach all given vertices, but this is not case, as it returns the entire path in a single strongly connected component. So, Professor Smart Joe's algorithm will show incorrect answers as explained above.