HOMEWORK-2
CSCI 570

SRI MANVITH VADDEBOYINA

USC ID: 1231409457

Q1. What is the worst case runtime performance of the procedure below?

```
c=0
i=n
while i>1 do
  for j=1 to i do
     c= c+1
  end for
   i= floor(i/2)
  endwhile
return c
```

Sol: As per the above code, the values of i are $n, \lfloor n/2 \rfloor, \lfloor n/4 \rfloor$ so on upto $i=1$. when $i=1$ the while loop exits.

As we know that the value of $n/2 >= \lfloor n/2 \rfloor$, we can derive the time complexity by the following Calculation.

$T.C = \lfloor n \rfloor + \lfloor n/2 \rfloor + \ldots +$ But as we know that the sum of $n + n/2 + n/4 + \ldots$ value we can say that the T.C is less than or equal to the sum of $n + n/2 + \ldots$.

i.e $\lfloor n \rfloor + \lfloor n/2 \rfloor + \ldots \leq n + n/2 + \ldots$

$\Downarrow$

Geometric Progression.

$n + n/2 + \ldots =$

$= n\left[\dfrac{1}{1-1/2}\right]$

$= 2n.$

$\therefore \lfloor n \rfloor + \lfloor n/2 \rfloor + \cdots \leq 2n$

$\boxed{T.C = \Theta(n)}$

Q2. Arrange these functions under the O notation using "=" or "⊂".

$2^{\log n}$, 2^{3n}, $n^{n\log n}$, $\log n$, $n\log(n^2)$, n^{n^2}, $\log(\log(n^n))$

A B C D E F G

Sol: Given 7 functions, our aim is to find the order i.e increasing order of their values with 'n'.

① Divide the functions into 3 categories

i. Logarithmic

ii. Exponential

iii. Polynomial

We get,

Logarithmic: $\log n$, $\log(\log(n^n))$

Exponential: 2^{3n}, $n^{n\log n}$, n^{n^2}

Polynomial: $n\log(n^2)$, $2^{\log n}$.

(i) Logarithmic

$\log(\log(n^n))$ and $\log n$.

Let $\log n = k$

$\log(\log n^n) = \log(n\log n) = \log(n \times k)$

$\Rightarrow \underbrace{\log n + \log k}_{\text{inf}} > \underbrace{\log n}_{\text{constant}} \Rightarrow \underbrace{k + \log k}_{\text{inf}} > \underbrace{k}_{\text{constant}}$

$\therefore O(\log n) = O(\log\log(n^n))$

(ii) Exponential: $2^{3n}, n^{n^2}, n^{n\log n}$

let $n^{n^2} = k$ | $2^{3n} = k$ | $n^{n\log n} = k$

$\log(n^{n^2}) = \log k$ | $3n \log_2 2 = \log k$ | $n(\log n)^2 = \log k$

$n^2 \log n = \log k$

Comparing $n^2 \log n$, $3n$ and $n(\log n)^2$ we get,

$$O(3n) \subset O(n(\log n)^2) \subset O(n^2 \log n)$$

$$O(2^{3n}) \subset O(n^{n\log n}) \subset O(n^{n^2})$$

(iii) Polynomial: $2^{\log n}$, $n\log(n^2)$

let $2^{\log_2 n} = k$ | $n \log n^2 = k$

$\Rightarrow n = k$ | $2n \log n = k$

$O(n)$

As we know, $O(n) \subset O(n\log n)$

$\rightarrow O(2^{\log n}) \subset O(n \log n^2)$

Finally as we know Exponential fn grows faster than polynomial fn and polynomial fn grows logarithmic fn.

Therefore, order is

$$\boxed{O(\log n) = O(\log(\log(n^n))) \subset O(2^{\log n}) \subset O(n\log(n^2)) \subset O(2^{3n}) \subset O(n^{n\log n}) \subset O(n^{n^2})}$$

3. $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$

a) $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n)) \Rightarrow$ TRUE

As given in the question above,

$f_1(n) = O(g_1(n))$ | $f_2(n) = O(g_2(n))$

$f_1(n) \leq c_1 * g_1(n)$ → (1) | $f_2(n) \leq c_2 * g_2(n)$ (2)

Using (1) & (2)

$$f_1(n) \cdot f_2(n) \leq c_1 * g_1(n) * c_2 * g_2(n)$$
$$\leq \underbrace{c_1 c_2}_{\text{constant}} g_1(n) * g_2(n)$$
$$= O(g_1(n) * g_2(n))$$

$\therefore f_1(n) \cdot f_2(n) = O(g_1(n) * g_2(n))$ (TRUE)

b) $f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$ (TRUE)

Using (1) & (2)

$$f_1(n) + f_2(n) \leq c_1 * g_1(n) + c_2 * g_2(n)$$
$$\leq c_1 * \max(g_1(n), g_2(n)) + c_2 * \max(g_2(n), g_1(n))$$
$$\leq (c_1 + c_2) \max(g_1(n), g_2(n)).$$

$f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$.

(TRUE)

c) $f_1(n)^2 = O(g_1(n)^2)$

(TRUE)

Using (1)

$f_1(n) \leq c_1 * g_1(n)$

squaring on both sides

$f_1(n)^2 \leq c_1^2 * g_1(n)^2$

$\therefore f_1(n)^2 = O(g_1(n)^2)$

(TRUE)

d) $\log_2 f_1(n) = O(\log_2 g_1(n))$

let us assume, $f_1(n) = 2$ and $g_1(n) = 1$

$$\log_2 f_1(n) = \log_2 2 = 1$$

$$\log_2 g_1(n) = \log_2 1 = \log_2 2^0 = 0$$

$$\log_2 f_1(n) \neq O(\log_2 g_1(n))$$

(FALSE)

4. Given an undirected graph G with n nodes and m edges, design O(m+n) algorithm to detect whether G contains a cycle. Your algorithm should output a cycle if G contains one.

Sol: Given a graph G we can find it contains a cycle or not using Breadth-first search (BFS) (or) Depth-first search (DFS).

In this case let us consider the use of DFS to find cycle if present.

Algorithm:

Let G be graph with m edges and n nodes. i.e G(n, m)

Consider an array ar of size 'n' ar[n] which stores the values which represent whether a node is visited (or) not. visited [n]

For a current node, consider a variable parent that stores the information about parent or its previous node.

For each non-visited node n ∈ N:
  Make n as visited and set parent = -1
    Make calls recursively for all n' ∈ adj (n):
      if n' ≠ parent:
        If for n', if visited[n'] was previously visited, then the graph has a cycle
        return TRUE

      Else
        parent = n and make visited[n'] visited

        Continue recursive calls until all nodes which are adjacent are visited

Else
    Consider node n and ignore n'
  ENDIF
 ENDFOR return FALSE
ENDFOR

After the algorithm is completed visiting all nodes, then there is no cycle, return False.
The DFS tree would take O(m) time.
The algorithm would take O(m+n) time in each step. Total $T.C = O(m+n)$

5. $G=(V,E)$
Given DFS tree is computed using root $u \in V$ and a tree T is obtained.
Given that BFS is also computed using root $u \in V$ and same BFS tree T is obtained.
To prove, G = T, G does not contain any edges that do not belong to T.

Proof by contradiction

* Assume that an edge (x,y) belonging to graph G that is not present in tree T.
* As per the DFS tree, either x or y is ancestor of the other.
* Similarly as per BFS tree, x and y differ by 1 Layer atmost.
* As per the above 2 statements, as either x or y is ancestor of other and they differ by atmost 1 layer, (x,y) should be present in BFS tree T.

* But this contradicts our assumption that (x, y) does not belong to T.
* Thus G cannot contain any edges that do not belong to T.
* $\therefore$ $\boxed{G = T}$