HomeWork 2

Ritu Pravakar, 5857424113

1. 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)
end while
return c
```

Provide a brief explaination for your answer.

Ans.

- There are n operation in the loop to be completed, since $i = n$.

- $n \to 1^{st}$ iteration
  $n/2 \to 2^{nd}$ iteration
  $n/4 \to 3^{rd}$ iteration
  $n + n/2 + n/4 + \dots < 2n$

- worst case $\to O(n \log n)$

2. Arrange these functions under the O notation using $=$ (equivalent) or $\subset$ (strict subset of):

(a) $2^{\log n}$
(b) 2^{3n} /
(c) $n^{n \log n}$ /
(d) $\log n$
(e) $n \log(n^2)$
(f) n^{n^2} /
(g) $\log(\log(n^n))$

E.g. for the function $n, n+1, n^2$, the answer should be

$$O(n+1) = O(n) \subset O(n^2)$$

Provide brief explaination for your arrangement.

Ans. n^{n^2}, $n^{n \log n}$, 2^{3n} are purely exponential.

(i) Addiy $\log n$ to ~~log~~ all.

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

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

$\log_n(2^{3n}) < n \log n < n^2$

$\Rightarrow 2^{3n} < n^{n \log n} < n^{n^2}$

ii) $2^{\log n}$, $n\log(n^2)$ are polynomial

Taking $\log_2$ both sides,

$\log_2(2^{\log n})$, $\log_2(n\log(n^2))$

$\log(n)$, $\log_2(\underline{n\log(n^2)})$.

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

iii) $\log n$, $\log(\log(n^n))$ are logarithmic

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

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

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

$O(n^{n^2})$

Ques 3. Given functions f_1, f_2, g_1, g_2 such that $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$. For each of the following statements, decide whether it is true or false and brief explain why.

(a) $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$
(b) $f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$
(c) $f_1(n)^2 = O(g_1(n)^2)$
(d) $\log_2 f_1(n) = O(\log_2 g_1(n))$

Ans. Given, $f_1(n) = O(g_1(n))$ —— ①
$f_2(n) = O(g_1(n))$ —— ②

① $\Rightarrow$ $f_1(n) \leq C_1 g_1(n)$ —— ③
② $\Rightarrow$ $f_2(n) \leq C_2 g_1(n)$ —— ④

(a) Multiplying equations ③ and ④,

$$f_1(n) \cdot f_2(n) \leq C_1 C_2 g_1(n) g_2(n)$$
$$= O(g_1(n) \cdot g_2(n))$$

Hence, (a) is <u>true</u>.

(2) Adding equations ③ and ④,

$$f_1(n) + f_2(n) \leq C_1 g_1(n) + C_2 g_1(n)$$
$$\leq (C_1 + C_2)(g_1(n) + g_2(n))$$
$$= O(\max(g_1(n), g_2(n)))$$

$\Rightarrow \leq 2(C_1 + C_2) \max(g_1(n), g_2(n))$

Hence (b) is <u>true</u>.

(c) equation (3) $\Rightarrow$

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

Squaring both sides,

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

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

Hence (c) is true.

(d) $f_1(n) \leq c_1 g_1(n)$ — equation (3)

Using $\log_2$ on both sides,

$$\log_2 f_1(n) \leq \log_2 (c_1 g_1(n))$$

$$\leq \log_2 c_1 + \log_2 g_1(n)$$

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

But when $g_1(n) = 1$, and $f_1(n) = 5$ (or anything else)

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

$$\log_2(5) \neq O(0)$$

Hence (d) is false

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

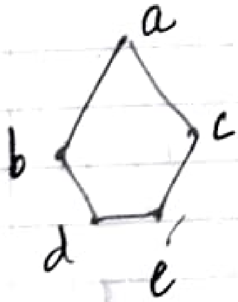
Ans. BFS or DFS can be used to find connected components of a graph.

- Assuming the graph is connected for the below algorithm.
- We start with a node a and we use ~~BFS or~~ DFS to generate its connected components.
- Say the graph G contains m edges. To iterate over m edges $\rightarrow O(m)$ time

Initialize visited[n] = 0
Initialize parent[n] = -1 // maintained to store parent of each node

Recursively call dfs on node b.
For edg (a, b);
parent[b] = a.
~~visited[b] = 1.~~
~~dfs(...)~~ if visited[b] is true

For each node where visited [a] = 0 :
visited [a] = 1
parent ~~[a]~~ = -1 .
Recursively call this function for all vertices adjacent to a (here pay, b&c if edge is (a, b) and (a, c)) until all nodes are visited :
if b ≠ parent
If b was visited [b] == 1 :
return True
else
parent = b .
visited [b] = 1 .
else
~~ign~~ call next node adjacent to a, (here c) .
end if
end for .
return False



To run over each node → n, O(n) time .
total time O(m + n) .

Ques 5. Solve Kleinberg and Tardos, Chapter 3, Exercise 6

We have a connected graph $G=(V,E)$ and a specific vertex $u \in V$. Suppose we compute a depth-first search tree rooted at u, and obtain a tree T that includes all nodes of G. Suppose we then compute a breadth-first search tree rooted at u, and obtain the same tree T.

Prove that $G = T$.

(In other words, if T is both a depth-first search tree and a breadth-first search tree rooted at u, then G cannot contain any edges that ~~does~~ do not belong to T.)

Ans. Proof by contradiction,

- Assume a edge (a, b) is not there in T but exists in G.

- Let T be a DFS tree, let (a, b) be nodes in T and (a, b) a edge in G, which is not in T.
When (a, b) is discovered in DFS, say b is it is not added, because b is already there in DFS tree.

- Since b was not discovered when DFS was started, it was discovered when DFS(a) was started and end of DF~~S~~(a).

- b is a decendant of a.

→ Hence, if T is a DFS tree, one of the nodes (a,b) is ansestor of other.

- Say a and b differ by atmost 1 layer. If a belogs to Layer$_x$, the nodes discovered during BFS from a will belong to Layer $x+1$ if b is a neighbour be discovered and belong to Layer L_{x+1} or earlier.

⇒ If T is BFS tree, the distance between a and b should differ by atmost one layer.

∴ T is BFS tree, a and b differ by exactly one layer and therefore ~~edge~~ edge (a, b) should be in BFS tree T.

This contradicts our assumption. Hence there cannot be an edge (a,b) in T which does not exist in G.