CSCI 570 Spring 2025 Homework 2 -- Solution

Q1. What is the tight upper bound to the worst-case runtime performance of the procedure below? (8points)

$c = 0$
$i = n$
while $i > 1$ **do**
  for $j = 1$ to i **do**
    $c = c + 1$
  end for
  $i = \text{floor}(i/2)$
end while
return c

Solution:

There are i operations in the for loop and the while loop terminates when i becomes 1. The total time is

$$n + \left\lfloor \frac{n}{2} \right\rfloor + \left\lfloor \frac{n}{4} \right\rfloor + \cdots \leq \left(1 + \frac{1}{2} + \frac{1}{4} + \cdots\right) \cdot n \leq 2n = \Theta(n)$$

Rubric (8 pts):

- 3 pts: if bound correctly found as $O(n)$
- 5 pts: Provides a correct explanation of the runtime
- If bound given is $O(n \log n)$, can give 5 pts total (i.e., not a tight upper bound)

Q2. 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 you think it is true or false and give a proof or counterexample. (12points)

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

Solution:

By definition, there exist c_1, $c_2 > 0$ such that

$$f_1(n) \le c_1 \cdot g_1(n) \text{ and } f_2(n) \le c_2 \cdot g_2(n)$$

for n sufficiently large.

(a) True.

$$f_1(n) \cdot f_2(n) \le c_1 \cdot g_1(n) \cdot c_2 \cdot g_2(n) = (c_1c_2) \cdot (g_1(n) \cdot g_2(n)).$$

(b) True.

$$\begin{aligned} f_1(n) + f_2(n) &\le c_1 \cdot g_1(n) + c_2 \cdot g_2(n) \\ &\le (c_1 + c_2)(g_1(n) + g_2(n)) \\ &\le 2 \cdot (c_1 + c_2) \max(g_1(n), g_2(n)). \end{aligned}$$

(c) True.

$$f_1(n)^2 \le (c_1 \cdot g_1(n))^2 = c_1^2 \cdot g_1(n)^2.$$

(d) False. Consider $f_1(n) = 2$ and $g_1(n) = 1$. Then

$$\log_2 f_1(n) = 1 \ne O(\log_2 g_1(n)) = O(0).$$

Rubric (3 pts for each subproblem, 12 pts total**):**

- 1 pts: Correct T/F claim
- 2 pts: Provides a correct explanation or counterexample

Q3. 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. (10points)

Solution:

Without loss of generality assume that G is connected. Otherwise, we can compute the connected components in O(m + n) time and deploy the below algorithm on each component. Starting from an arbitrary vertex s, run BFS to obtain a BFS tree T , which takes O(m + n) time. If G = T , then G is a tree and has no cycles. Otherwise, G has a cycle and there exists an edge $e = (u, v) \in G \setminus T$. Let w be the least common ancestor of u and v. There exist a unique path T1 in T from u to w and a unique path T2 in T from w to v. Both T1 and T2 can be found in O(m) time. Output the cycle e by concatenating P1 and P2.

Rubric (10 pts):

No penalty for not mentioning disconnected case.

- 4 pts: for detecting whether G contains a cycle
- 3 pts: for finding (the edges in) a cycle if G contains one
- 3 pts: describing that the runtime is $O(m + n)$ in each step (and thus total)

Q4. Design an algorithm which, given a directed graph G = (V, E) and a particular edge e ∈ E going from node u to node v, determines whether G has a cycle containing e. The running time should be bounded by O(|V | + |E|). Explain why your algorithm runs in O(|V | + |E|) time. (10points)

Solution: Delete e from G to obtain a new graph G', run the DFS or BFS algorithm starting from v, if u can be traversed, then G has a cycle containing e, otherwise G does not have such a cycle. In the worst case, all the edges and nodes are traversed, resulting in O(|V | + |E|) time; or you can say the running time of DFS or BFS is O(|V | + |E|).

Rubric (10 pts):

- 7 pts : Correctly utilizing BFS/DFS
- 3 pts : Explanation for run time

Q5. For each of the following indicate if f = O(g) or f = Θ(g) or f = Ω(g). (10 points)

(1). $f(n) = n^4/\log(n)$ $\quad g(n) = n(\log(n))^4$

(2). $f(n) = n*\log(n)$ $\quad g(n) = n^2\log(n^2)$

(3). $f(n) = \log(n)$ $\quad g(n) = \log(\log(5^n))$

(4). $f(n) = n^{1/3}$ $\quad g(n) = (\log(n))^3$

(5). $f(n) = 2^n$ $\quad g(n) = 2^{3n}$

Solution:

(1). f = Ω(g)

(2). f = O(g)

(3). f = Θ(g)

(4). f = Ω(g)

(5). f = O(g)

Rubric (10 pts total) :

2 points for each correct answer.

Practice (Ungraded) Problems:

1. Solve Kleinberg and Tardos, **Chapter 3, Exercise 9.**

Solution: We run BFS starting from node s. Let *d* be the layer in which node t is encountered; by assumption, we have $d > n/2$. We claim first that one of the layers $L_1, L_2, ..., L_{d\text{-}1}$ consists of a single node. Indeed, if each of these layers had size at least two, then this would account for at least $2(n/2) = n$ nodes; but *G* has only n nodes, and neither *s* nor t appears in these layers. Thus, there is some layer *L*, consisting of just the node v. We claim next that deleting v destroys all s-t paths. To see this, consider the set of nodes $X = \{s\} \cup Z_1 \cup Z_2 \cup \cdot\cdot\cdot \cup L_{i\text{-}1}$ · Node s belongs to X but node *t* does not; and any edge out of X must lie in layer L_i, by the properties of BFS. Since any path from s to t must leave X at some point, it must contain a node in L_i; but v is the only node in *L*.

2.Solve Kleinberg and Tardos, **Chapter 3, Exercise 6.**

Solution: Proof by Contradiction: assume there is an edge e = (x, y) in G that does not belong to T. Since T is a DFS tree, one of x or y is the ancestor of the other. On the other hand, since T is a BFS tree, x and y differ by at most 1 layer. Now since one of x and y is the ancestor of the other, x and y should differ by exactly 1 layer. Therefore, the edge e = (x, y) should be in the BFS tree T. This contradicts the assumption. Therefore, G cannot contain any edges that do not belong to T.