# Homework 2

● Graded

**Student**

Abhishek Soundalgekar

**Total Points**

**50 / 50 pts**

**Question 1**

**(no title)** **8** / 8 pts

✔ **+ 8 pts** Fully correct

**+ 3 pts** if bound correctly found as O(n)

**+ 5 pts** Provides a correct explanation of the runtime

**+ 0 pts** Incorrect

**Question 2**

(no title) **12** / 12 pts

2.1 **(no title)** **3** / 3 pts

✔ **+ 3 pts** Fully Correct

**+ 1 pt** Correct T/F claim

**+ 2 pts** Provides a correct explanation or counterexample

**+ 0 pts** Incorrect

2.2 **(no title)** **3** / 3 pts

✔ **+ 3 pts** Fully Correct

**+ 1 pt** Correct T/F claim

**+ 1 pt** Imprecise Explanation

**+ 2 pts** Provides a correct explanation or counterexample

**+ 0 pts** Incorrect

2.3 **(no title)** **3** / 3 pts

✔ **+ 3 pts** Fully Correct

**+ 1 pt** Correct T/F claim

**+ 2 pts** Provides a correct explanation or counterexample

**+ 0 pts** Incorrect

2.4 **(no title)** **3** / 3 pts

✔ **+ 3 pts** Fully Correct

**+ 1 pt** Correct T/F claim

**+ 2 pts** Provides a correct explanation or counterexample

**+ 0 pts** Incorrect

**Question 3**

**(no title)** **10** / 10 pts

✔ **+ 10 pts** Fully Correct

**+ 4 pts** (only) successfully detecting whether G contains a cycle

**+ 3 pts** (only) finding (the edges in) a cycle if G contains one

**+ 7 pts** successfully detecting whether G contains a cycle, and finding (the edges in) a cycle if G contains one

**+ 0 pts** Incorrect

**Question 4**

**(no title)** **10** / 10 pts

✔ **+ 10 pts** Fully Correct

**+ 7 pts** Correctly utilizing BFS/DFS

**+ 0 pts** Incorrect

**Question 5**

**(no title)** **10** / 10 pts

✔ **+ 10 pts** Fully Correct

**+ 8 pts** 4 out of 5 are correct

**+ 6 pts** 3 out of 5 are correct

**+ 4 pts** 2 out of 5 are correct

**+ 2 pts** 1 out of 5 are correct

**+ 0 pts** 0 out of 5 are correct

Question assigned to the following page: 1

CSCI 570 Spring 2025

Homework 2

Name : Abhishek Soundalgekar

USC ID : 2089011000

Question 1

Answer 1

In the procedure given the 'for loop' contains 'i' operations.
And for the 'while loop' when 'i' becomes 1 (one) the while loop will terminate

The outer while loop halves the value of i for each iteration and an inner for loop that runs from 1 to the current value of i.

The total operations and time is

$$n + \left[\frac{n}{2}\right] + \left[\frac{n}{4}\right] + \ldots \leq \left(1 + \frac{1}{2} + \frac{1}{4} \ldots n\right) \leq 2n = O(n)$$

This simplifies to $O(n)$.
And the complexity of outer loop is $O(\log n)$
and $O(n)$ is greater than $O(\log n)$.
∴ The tight upper bound is $O(n)$

Questions assigned to the following page: 2.1 and 2.2

**Question 2**
**Answer 2**

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

Given: $f_1(n) = O(g_1(n))$
$f_2(n) = O(g_2(n))$

By the given definition there exist constants $P_1, P_2 > 0$ such that

$f_1(n) \leq P_1 \cdot g_1(n)$ & $f_2(n) \leq P_2\, g_2(n)$
(for $n$ being very large enough).

(a) The statement is Tenue.
$f_1(n) \cdot f_2(n) \leq P_1 g_1(n) \cdot P_2 g_2(n) = (P_1 P_2) \cdot (g_1(n) \cdot g_2(n))$
Hence it is Tenue.

b] $f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$
Let $G(n) = \max(g_1(n), g_2(n))$
By given definition there exists constants $P_1, P_2, n_1, n_2$
$f_1(n) \leq P_1 G(n)$ & $f_2(n) \leq P_2 G(n)$
(for $n$ very large) [$\forall\, n \geq \max(n_1, n_2)$]
Adding these inequalities:
$f_1(n) + f_2(n) \leq (P_1 + P_2)\, G(n)$
Hence Proved statement is Tenue.

Questions assigned to the following page: 2.3 and 2.4

[c] The given statement is True.

Proof:

We take given $f_1(n) = O(g_1(n))$

There exist constant $p, n'$ such that

$$f_1(n) \leq p \cdot g_1(n) \quad \forall n \geq n'.$$

Now we square both the sides we get

$$f_1(n)^2 \leq p^2 \cdot g_1(n)^2 \quad \forall n \geq n'.$$

Hence proved $f_1(n)^2 = O(g_1(n)^2)$.

[d] The given statement is False.

We will prove this using a counter example

let $f_1(n) = 2$ & $g_1(n) = 1$ for all $n$.

Here, $f_1(n) = O(g_1(n))$ (with $c = 2$)

But

$$\log_2 f_1(n) = \log_2 2 = 1, \quad \log_2 g_1(n) = \log_2 1 = 0$$

There is no constant $C$ such that $1 \leq C \cdot 0$

Hence Proved $\log_2 f_1(n) \neq O(\log_2 g_1(n))$.

Question assigned to the following page: 3

Question 3
Answer 3

Given : An undirected graph $G$ with $n$ nodes and $m$ edges

To design : An $O(m+n)$ algorithm to detect whether $G$ contains a cycle.
Output cycle if $G$ contains one.

Algorithm :

Step 1 : Run BFS and construct a BFS tree $T$ $(O(m+n))$
→ Start BFS from an arbitrary vertex $s$ and construct a BFS tree $T$.
→ Maintain a parent array to store the parent of each node in the BFS tree.
→ If $G = T$, then $G$ is a tree and has no cycles
→ If an edge $e = (u, v)$ exists in $G \setminus T$, then $G$ contains a cycle.

Step 2 : Find the cycle $O(m+n)$
→ If $e = (u, v)$ is an edge not in $T$, then :
⇒ Find the least common ancestor (LCA) $w$ of $u$ and $v$ in $T$.
→ There exist two unique paths :
$P_1$ : Path from $u$ to $w$ in $T$.
$P_2$ : Path from $v$ to $w$ in $T$.
→ The cycle is formed by concatenating $P_1$, $e$ and $P_2$.

Question assigned to the following page: 3

Step 3: Output the cycle
→ Traverse back using the parent array to extract $P_1$ and $P_2$.
→ Return the edges forming the cycle.

Pseudocode

```
Function DetectCycle(G):
    Initialize queue Q
    Initialize visited set
    Initialize parent dictionary

    For each node s in G:
        If s is not visited:
            Enqueue s into Q
            Mark s as visited

            While Q is not empty:
                Dequeue node u
                For each neighbour v of u in G:
                    If v is not visited:
                        Mark v as visited
                        Set parent[v] = u
                        Enqueue v
                    Else If v is not parent of u:
# Cycle detected →          Store back edge (u,v)
                        Go to Step 2
```

Question assigned to the following page: 3

Return "No cycle detected"

Step 2 :- Find cycle

Find LCA w by tracking back from u & v using parent dictionary
Construct path P1 from u to w
Construct path P2 from v to w
Return the cycle as (P1 + (u,v) + P2)

Time complexity Analysis

BFS construction takes : O(m+n)
cycle detection takes : O(m)
Path Extraction : O(n)
∴ Total complexity = O(m+n)

Question assigned to the following page: 4

Question 4
Answer 4

Delete edge e (from u to v) from G to obtain a new graph G'. Perform a BFS or DFS starting from v in G'. If u is reachable from v in G', then G contains a cycle that includes e; otherwise, it does not.

→ Cycle formation: If u is reachable from v in G', the path v ⇝ u in G' combined with the deleted edge e (from u → v) forms a cycle containing e.

→ Runtime: Removing e takes O(1) time (assuming adjacency list manipulation or simulated edge skipping during traversal). BFS/DFS on G' runs in O(|V|+|E|), as every node and edge (except e) is processed once.

→ Correctness: The algorithm ensures no false positives/negatives because the cycle must include e, and G' explicitly excludes e. If a path v ⇝ u exists in G', adding e creates the

Question assigned to the following page: 4

cycle $u \to v \rightsquigarrow u$.

Algorithm

Step 1 : Remove the edge $e = (u, v)$
→ temporarily remove the edge e from the graph to obtain a new graph G'.

Step 2 : Perform BFS or DFS from Node V
→ start a graph traversal (BFS or DFS) from node v in G'.
→ keep track of visited nodes.

Step 3 : check if u is Reachable
→ If the traversal reaches node u, it means there is a path from v to u in G', which implies the existence of a cycle containing e in G
→ Otherwise, no such cycle exists.

Step 4 : If required add e edge back to original graph restoration

Question assigned to the following page: 4

Pseudocode :

Function HasCycleThroughEdge (G, u, v) :
1. Remove edge (u, v) from G to obtain G'
2. Initialize an empty set Visited
3. Initialize a stack or queue Q
4. Add node v to Q and mark it as visited
5. While Q is not empty :
    a. Dequeue a node x from Q
    b. For each neighbor y of x in G' :
        i] If y is not visited :
            - Mark y as visited
            - Add y to Q
        ii] If y == u :
            Restore edge (u,v) in G (if needed)
            Return "cycle containing e exists"
6. Restore edge (u, v) in G (if needed)
7. Return "No cycle containing e"

Time complexity :
Removing edge e : $O(1)$
Running BFS or DFS : $O(|V|+|E|)$
Checking reachability : $O(|V|+|E|)$
Restoring edge (optional) : $O(1)$

∴ Total time complexity : $O(|V|+|E|)$

Question assigned to the following page: 5

Question 5

Answer 5

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

$f(n)$ grows proportional to $\frac{n^4}{\log(n)}$

$g(n)$ grows proportional to $n(\log(n))^4$

Here, the polynomial component of $f(n)$ i.e, $n^4$ is higher degree than that of $g(n)$

Hence, $n^4$ runs faster than $n(\log(n))^4$

As $n$ grows large, $\frac{n^4}{\log n)}$ dominates $n(\log(n))^4$

$f(n) = \Omega(g(n))$ $\quad$ indicates $f = \Omega(g)$

Question assigned to the following page: 5

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

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

If we ignore the logarithmic part we get
for $f(n)$: $n$ & $g(n)$: $2n^2$
we ignore the constant 2 in $g(n)$ we get
$f(n) = n$ & $g(n) = n^2$

We know that, $n^2$ grows faster than $n$

$f(n)$ grows linearly in $n \log(n)$, while
$g(n)$ grows quadratically as $2n^2 \log(n)$
As $n$ grows large $g(n)$ dominates $f(n)$

$f(n) = O(g(n))$ indicates $f = O(g)$

Question assigned to the following page: 5

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

$f(n) = \log(n)$ $\quad g(n) = \log(n) + \log(5)$

Now, there can exist two constants $C_1$ & $C_2$ such that $0 \leq C_1 * g(n) \leq f(n) \leq C_2 * g(n)$

Hence $f(n) = \Theta(g(n))$ indicates $f = \Theta(g)$

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

Taking log of both $f(n)$ & $g(n)$ we get

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

$f(n) = \frac{1}{3}\log(n)$ $\quad g(n) = 3\log(\log(n))$

Let $x = \log(n)$, substituting $x$ on both

$f(n) = \frac{x}{3}$ $\quad g(n) = 3\log x$

Here, $f(n)$ is a polynomial function as compared to $g(n)$ which happens to be a logarithmic function

Hence $f(n)$ grows faster than $g(n)$.

$f(n) = \Omega(g(n))$ indicates $f = \Omega(g)$

Question assigned to the following page: 5

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

$f(n) = 2^n$ $\quad g(n) = 8^n$

Here $8^n$ & $2^n$ comparison, both are exponential components.
But, $8^n$ grows much faster as compared to $2^n$.

Hence $f(n) = O(g(n))$

Indicating $f = O(g)$