CMP 418

grass

PQ 20-21 [Questions & Answers]

SIMPLIFIED SOLUTIONS

Question 1: Asymptotic Notation Problem

a) Use emperical analysis to analyze each function given below and rearrange the functions in increasing order of growth (after computing $n = 1000$).

Use $n = 10, 50, 100, 200, 300, 400, 500, 1000$

$f_1 = n^2$, $f_2 = n$, $f_3 = n^2 \log_2 n$, $f_4 = \log_2 n^2$

b) With the aid of a diagram define and differentiate between the following 3 asymptotic notations

i. Big O notation
ii. Big Omega (Ω) notation
iii. Theta (θ) notation

(TIP: $\log_2 5 = \log 5 \div \log 2$) # Use a Calculator

i) $n = 1000$

a) # Calculate for $n = 1000$

$f_1 = n^2 = 1000^2 = 1\,000\,000$

$f_2 = n = 1000 = 1000$

$f_3 = n^2 \log_2 n = 1000^2 \times \log_2 1000 = 9965784.3$

$f_4 = \log_2 n^2 = \log_2 1000^2 = 19.9$

Rearrange each function in increasing order of growth

f_4 : grows the slowest # 19.9

f_2 : grows faster than f_4 # 1000

f_1 : grows faster than f_2 # 1000 000

f_3 : grows faster than f_1 # 9965 784.3

Final order

- $f_4 = \cancel{\log_2 1000} \log_2 n^2$
- $f_2 = n$
- $f_1 = n^2$
- $f_3 = n^2 \log_2 n$

ii) $n = 10$

Calculate for $n = 10$

$f_1 = n^2 = 10^2 = 100$

$f_2 = n = 10 = 10$

$f_3 = n^2 \log_2 n = 10^2 \log_2 10 = 332.2$

$f_4 = \log_2 n^2 = \log_2 10^2 = 6.6$

Rearrange each function in increasing order of growth

f_4 : grows the slowest # 6.6

f_2 : grows faster than f_4 # 10

f_1 : grows faster than f_2 # 100

f_3 : grows faster than f_1 # 332.2

Final order

- $f_4 = \log_2 n^2$
- $f_2 = n$
- $f_1 = n^2$
- $f_3 = n^2 \log_2 n$

iii) $n = 50$

Do this yourself, same process

iv) $n = 100$

Do this yourself, same process

v) $n = 200$

Do this yourself, same process

vi) $n = 300$

Do this yourself, same process

vii) $n = 400$

Do this yourself, same process

viii) $n = 500$

Do this yourself, same process

Final Order

- f_4
- f_2
- f_1
- f_3

O means $\leq$ (less than or equal to)

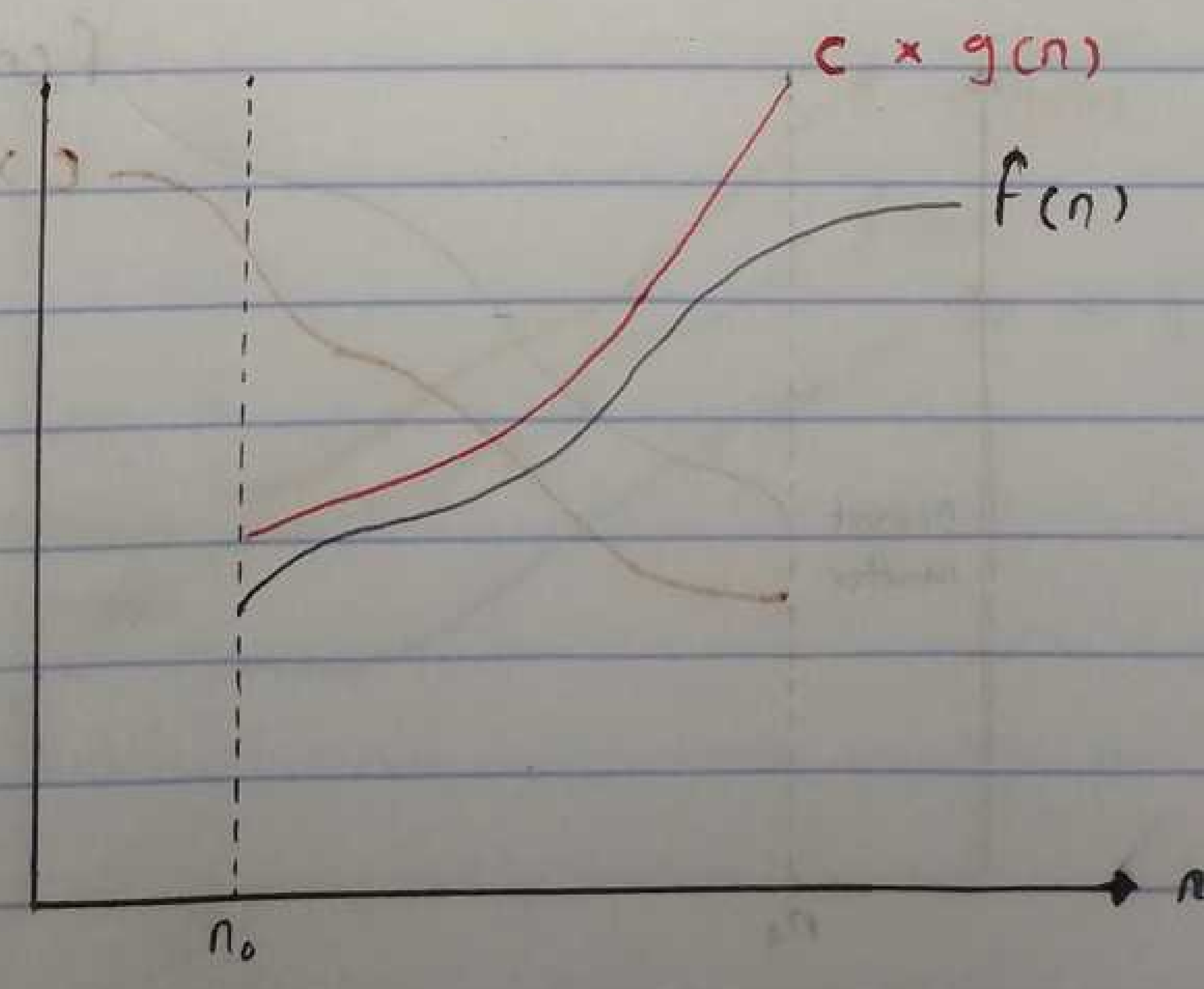
b) i) Big O notation (∈ means = or is)

Definition: A function $f(n)$ is said to be in $O(g(n))$, denoted $f(n) \in O(g(n))$, denoted $f(n) \in O(g(n))$, if $f(n)$ is bounded above by some positive constant (c) $\times g(n)$ for sufficiently large n. If we can find positive constants c and n_0 such that: $f(n) \leq c \times g(n)$ for all $n \geq n_0$

- Then $O(g(n))$ are set of functions that grow no faster than $g(n)$. Written as

same as: means less than or equal to

- $f(n) \in O(g(n))$ # $f(n)$ is $\leq g(n)$



$\}$ Ω means $\geq$ (greater than or equal to)

ii) Big Omega (Ω) notation

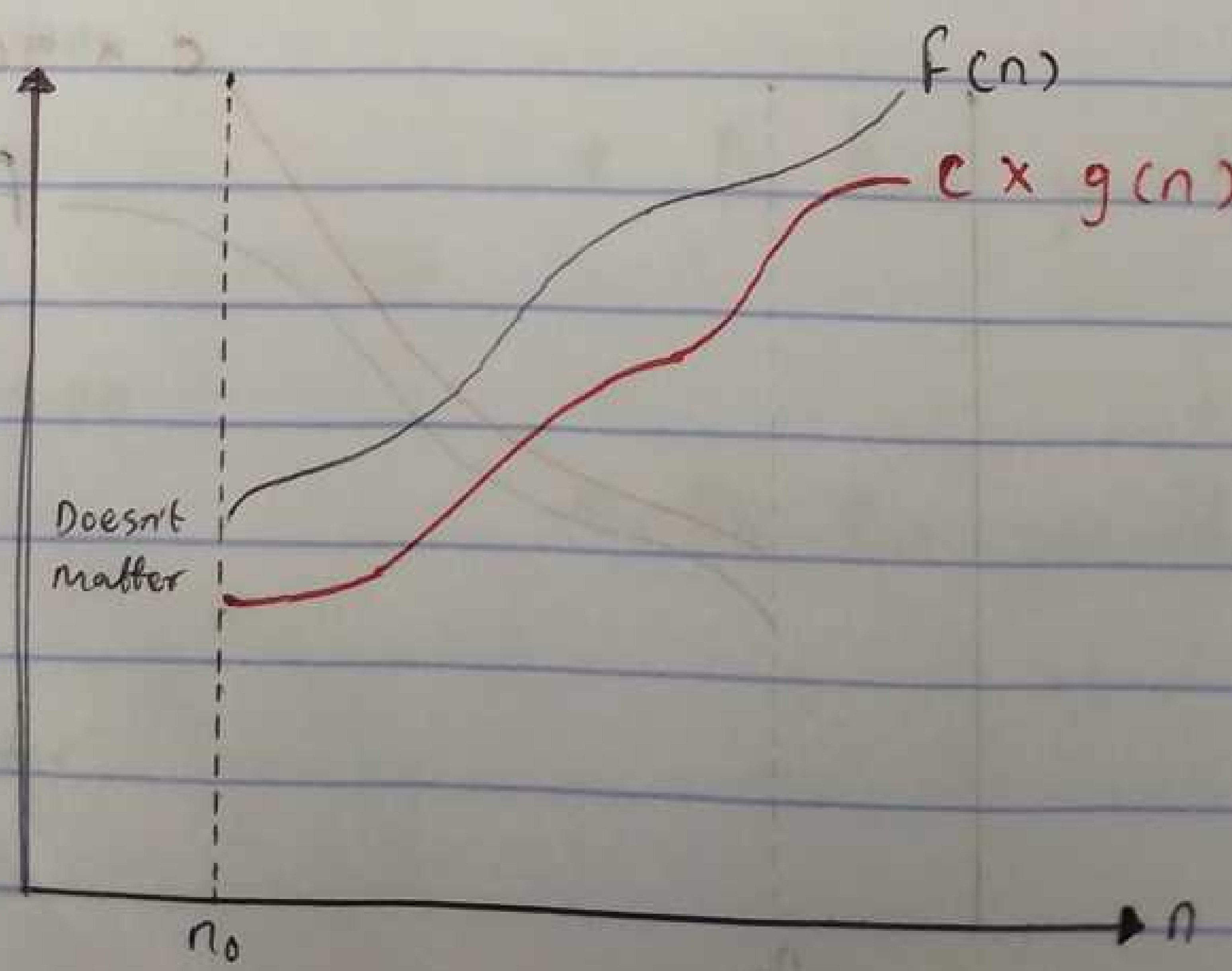
Definition: A function $f(n)$ is said to be in $\Omega(g(n))$ denoted $f(n) \in \Omega(g(n))$, if $f(n)$ is bounded below by some positive constant $(c) \times g(n)$ for all sufficiently large n.

If we can find positive constants c and n_0 such that:

$f(n) \geq c \times g(n)$ for all $n \geq n_0$

- Then $\Omega(g(n))$ are set of functions that grow at least as fast as $g(n)$. Written as:

- $f(n) \in \Omega(g(n))$ # $f(n)$ is $\overset{\text{same as}}{\geq}$ $g(n)$



$\{\Theta$ means $=$ (equal to)$\}$

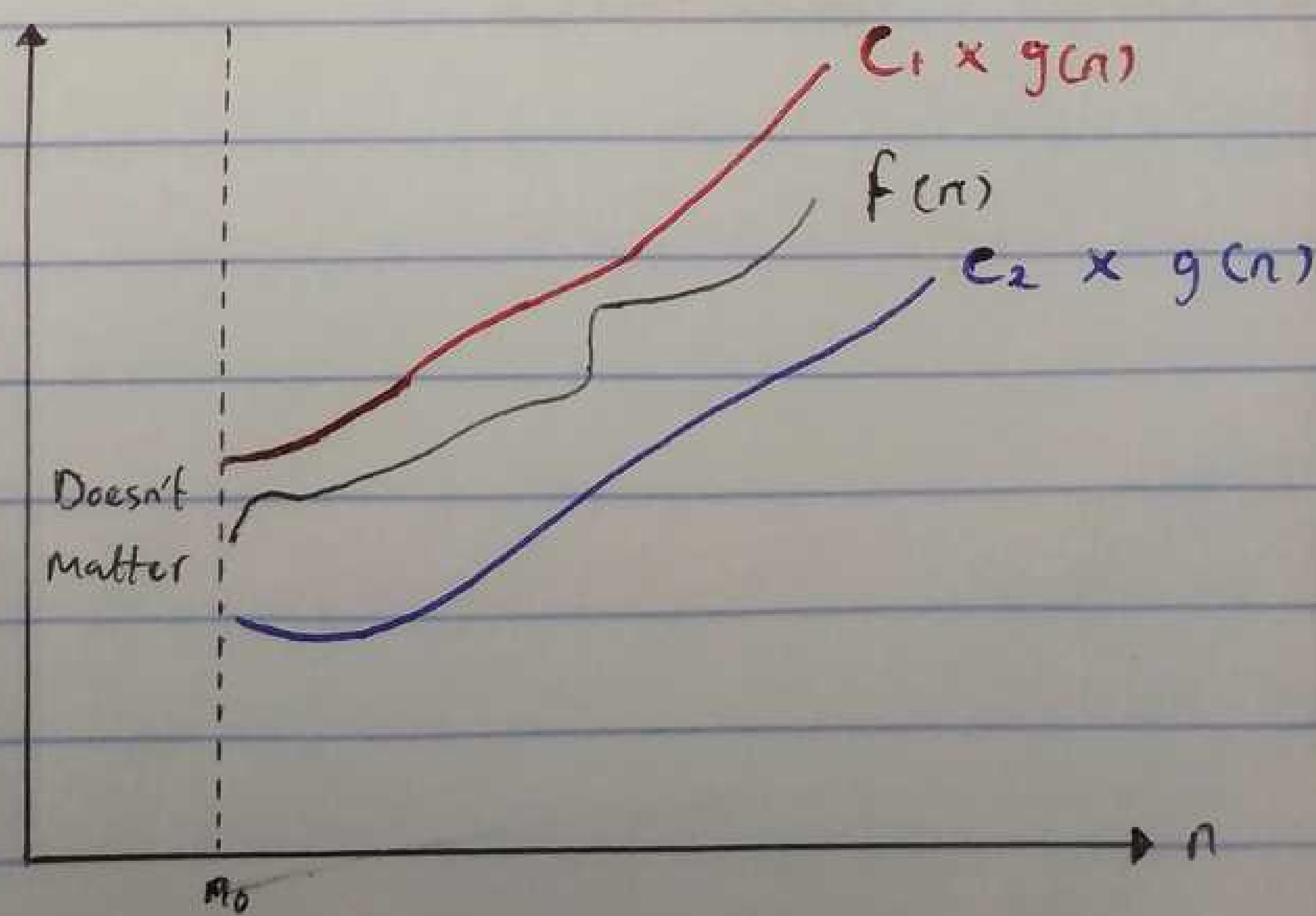
$\forall$ means "for all"

iii) Theta (Θ) notation

Definition: A function $f(n)$ is said to be in $\Theta(g(n))$ denoted $f(n) \in \Theta(g(n))$, if $f(n)$ is bounded both above and below by some positive constant (c) $g(n)$ for all sufficiently large n. If we can find positive constants c_1, c_2 and n_0 such that: $c_2 \times g(n) \leq f(n) \leq c_1 \times g(n) \ \forall n \geq n_0$

- Then $\Theta(g(n))$ are set of functions that grow at the same rate as $g(n)$. Written as:

- $f(n) \in \Theta(g(n))$ # $f(n)$ is $\overset{\text{same as}}{=}$ $g(n)$



Question 2: Analysis of Algorithm

ALGORITHM: BHU

```
for i ← 0 to n - 2 do
    for j ← i+1 to n-1 do
        if A[i] = A[j] return false
return true
```

a) Write the general plan for the analysis of non-recursive algorithms (5 marks)

b) What is algorithm BHU above computing? (1 mark)

c) Is the algorithm BHU Stable? (1 mark)

d) Is the algorithm BHU in place? (1 mark)

e) Use the 5 steps you listed in "Question 2a" to analyze the BHU algorithm (7 marks)

a) 1. Decide on parameter n indicating input size
2. Identify the algorithm's BOP
3. Determine **Worst, Best,** and **Average** cases for input of size n
4. Set up a sum of the **number of times** the BOP is executed
5. Simplify the sum using **standard formulas** and rules to establish it's order of growth

Q: What is the Algorithm Complexity?

b) Algorithm BHU is checking whether all elements in the array "A" are unique

Q: Is the Algorithm BHU stable?

c) The question of stability is irrelevant in this context, since the BHU algorithm is not a "Sorting Algorithm"

Stability in algorithms refers to whether the algorithm preserves the relative order of equal elements in a sorting process

Pro Tip: A sorting algorithm is stable if its BOP (Basic Operation) uses either > (greater than) or < (less than)

Q: Is algorithm BHU in place

d) Yes, it is

An algorithm is in-place if it requires a constant amount of extra Space (i.e., $O(1)$ additional space)

Q: Use the 5 steps in Q2a to analyze Algorithm BHU

e) STEP 1: Decide on parameter n indicating input size

In algorithm BHU, n represents the size of the array A. Which means, n is the number of elements in the array

e.g $A = [5, 2, 9]$ # $n = 3$, in array A

$C = [11, 3, 0, 15, 7]$ # $n = 5$, in array C

STEP 2: Identify the algorithms BOP (Basic Operation)

The BOP is the comparison $A[i] = A[j]$ # BOP = $A[i] = A[j]$

The BOP above checks if 2 elements in the array A are equal (in other words, it checks for duplicates)

$A[i]$ # element 1

$A[j]$ # element 2

STEP 3: Determine Worst, Best, and Average cases ...

- Best Case: The best case occurs when a duplicate is found in the first comparison (i.e., A[0] = A[1]). In this scenario, the algorithm terminates early, resulting in $O(1)$ time complexity.

1st element → 2nd element

e.g A = [2, 2, 5, 1]   # A[0] = 2, A[1] = 2
A[2] = 5, A[3] = 1

3rd element   4th element

Now if, A[0] = A[1] is true, are the same, the program terminates $\boxed{\text{return false}}$ executes

A = [2, 2, 5, 1]   if # Same, duplicates found
0 1
$\boxed{2 = 2 \text{ are the same}}$

Worst

- Case: The worst case occurss when all elements are unique, so the algorithm must perform the maximum number of comparisons resulting in $O(n^2)$ time complexity

e.g Given an array A = [8, 2, 5, 1]
No elements are the same

($\leftarrow$ means $=$)

- Average Case: In the average case, the algorithm will typically find a duplicate (or match) somewhere in between the best and worst cases (In short, the middle ~~?~~ of the array A), resulting in $O(n^2)$ still.

> e.g Given an array $A = [5, 9, \underset{2}{6}, \underset{3}{6}, 11, 2]$
>
> #
>
> # The middle elements are the same

STEP 4: Setup a sum for the number of times the BOP exec...

- The outer loop runs from $i = \underset{\wedge}{0}$ (lower) to $n-2$ (upper) which gives us the below:

Lower: $i = 0$

Upper: $n-2$

$$\sum_{\text{Lower}}^{\text{upper}} \text{ becomes } \sum_{i=0}^{n-2}$$

(for $i \leftarrow 0$ to $n-2$ do) — same as

- The inner loop runs from $j = i+1$ (Lower) to $n-1$ (Upper) which gives us the below:

Lower: $j = i + 1$

Upper: $n - 1$

$$\sum_{\text{Lower}}^{\text{Upper}} = \sum_{j=i+1}^{n-1}$$

(for $j \leftarrow i+1$ to $n-1$ do) same as

- ~~[illegible]~~ Next calculate : Outer loop × Inner loop

$$T(n) = \cancel{\sum^{n-2}} \text{ Outer loop } \times \text{ inner loop}$$

$$= \sum_{i=0}^{n-2} \times \sum_{j=i+1}^{n-1} \quad \text{\# or } \sum_{\cdots}^{\cdots}\sum_{\cdots}^{\cdots}, \text{ both means multiply}$$

- This can then be simplified to:

$$T(n) = (n-1) + (n-2) + (n-3) + \cdots + 1$$

STEP 5: Simplify the sum using standard formulas and...

$$T(n) = \sum_{k=1}^{n-1} k = \frac{(n-1)n}{2}$$

$$= \frac{n(n-1)}{2}$$

$$= \frac{n^2 - n}{2}$$

- Simplify the Order of Growth
- The dominant term in $T(n) = \frac{n^2 - n}{2}$ is $\frac{n^2}{2}$
- When analyzing "Complexity" we must discard all constants
  e.g 2 in $\frac{n^2}{2}$ is a constant so we remove it to get n^2

Final answer : $T(n) = O(n^2)$ //

Question 3: Brute Force - Exhaustive Search

You are paid to lead a software development project, which comprises of 4 subsystems; a company can implement only one subsystem at a time. That is, each company can handle exactly one subsystem and each subsystem should be handled by only one company at a time. The cost that would accrue if the ith company is awarded to develop the jth subsystem is given as Total Cost $C[i, j]$ for each pair $i, j = 1, 2, 3, 4$. As shown below in the table below:

a) Find the assignment with the most minimum total cost

b) Find the assignment with the most maximum total cost

c) How much would you have lost after all possible assignments?

Company	Subsystem 1	Subsystem 2	Subsystem 3	Subsystem 4
Company 1	9	2	7	8
Company 2	6	4	3	7
Company 3	5	8	1	8
Company 4	7	6	9	4

Lecture Slide 3

- page 28 and 29

Tip: Subsystem 1, Subsystem 2, Subsystem 3, Subsystem 4
< Company ?, Company ?, Company ?, Company ? >

Perform Exhaustive Search

•Min • Max

Round 1

Round 1

Sub 1 | Sub 2 | Sub 3 | Sub 4

1. < 1, 2, 3, 4 > => Cost = 9 + 4 + 1 + 4 = 18
2. < 1, 2, 4, 3 > => Cost = 9 + 4 + 9 + 8 = 30
3. < 1, 3, 2, 4 > => Cost = 9 + 8 + 3 + 4 = 24
4. < 1, 3, 4, 2 > => Cost = 9 + 8 + 9 + 7 = 33
5. < 1, 4, 2, 3 > => Cost = 9 + 6 + 3 + 8 = 26
6. < 1, 4, 3, 2 > => Cost = 9 + 6 + 1 + 7 = 23

Round 2

Sub 1 | Sub 2 | Sub 3 | Sub 4

7. < 2, 1, 3, 4 > => Cost = 6 + 2 + 1 + 4 = 13
8. < 2, 1, 4, 3 > => Cost = 6 + 2 + 9 + 8 = 25
9. < 2, 3, 1, 4 > => Cost = 6 + 8 + 7 + 4 = 25
10. < 2, 3, 4, 1 > => Cost = 6 + 8 + 9 + 8 = 31
11. < 2, 4, 1, 3 > => Cost = 6 + 6 + 7 + 8 = 27
12. < 2, 4, 3, 1 > => Cost = 6 + 6 + 1 + 8 = 21

Round 3

Sub 1 | Sub 2 | Sub 3 | Sub 4

13. < 3, 1, 2, 4 > => Cost = 5 + 2 + 3 + 4 = 14
14. < 3, 1, 4, 2 > => Cost = 5 + 2 + 9 + 7 = 23
15. < 3, 2, 1, 4 > => Cost = 5 + 4 + 7 + 4 = 20
16. < 3, 2, 4, 1 > => Cost = 5 + 4 + 9 + 8 = 26
17. < 3, 4, 1, 2 > => Cost = 5 + 6 + 7 + 7 = 25
18. < 3, 4, 2, 1 > => Cost = 5 + 6 + 3 + 8 = 22

ROUND 4

Sub 1 | Sub 2 | Sub 3 | Sub 4

19. $\langle 4, 1, 2, 3 \rangle \Rightarrow$ Cost $= 7 + 2 + 3 + 8 = 20$

20. $\langle 4, 1, 3, 2 \rangle \Rightarrow$ Cost $= 7 + 2 + 1 + 7 = 17$

21. $\langle 4, 2, 1, 3 \rangle \Rightarrow$ Cost $= 7 + 4 + 7 + 8 = 26$

22. $\langle 4, 2, 3, 1 \rangle \Rightarrow$ Cost $= 7 + 4 + 1 + 8 = 21$

23. $\langle 4, 3, 1, 2 \rangle \Rightarrow$ Cost $= 7 + 8 + 7 + 7 = 29$

24. $\langle 4, 3, 2, 1 \rangle \Rightarrow$ Cost $= 7 + 8 + 3 + 8 = 26$

a) Minimum Total Cost Assignment : 13

b) Maximum Total Cost Assignment : 33

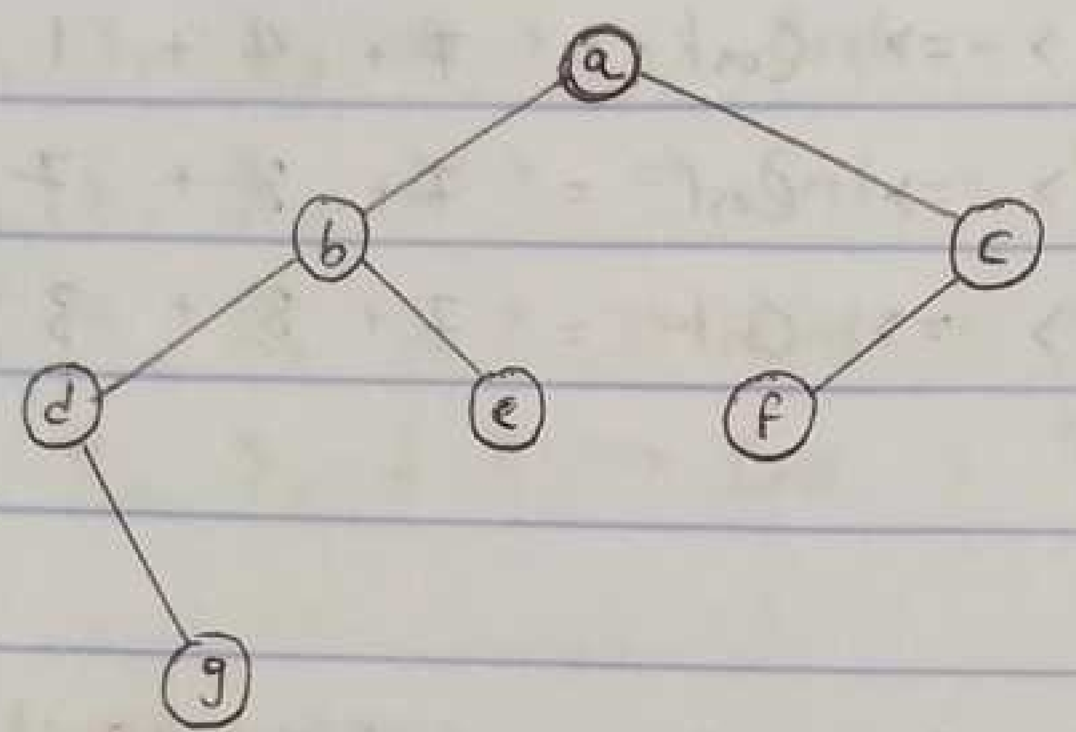
c) Loss = Maximum Cost − Minimum Cost

$= 33 - 13$

$= 20$

Question 4: Decrease and Conquer

a) What is the preorder, inorder, and postorder representation of the following tree?



TIP

Root Node

Nodes (a, b)

Leaf Nodes (e, f)

Leaf Node (g)

b) There are 3 major methods of implementing decrease and conquer. List and Explain each

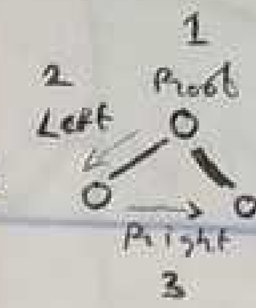
c) How many iterations do you need to search for $k = 70$, $k = 85$, & $k = 31$ when you apply a binary search algorithm?

0	1	2	3	4	5	6	7	8	9	10	11	12
3	14	27	31	39	42	55	70	74	81	85	93	98

Tip ⓐ => Unvisited

ⓐ => Visited

1

a) • Preorder Traversal (Root, Left, Right) # 

a , b , d , g , e , c , f

EXPLANATION # How did we get the above ans

We start at (a) (Root)

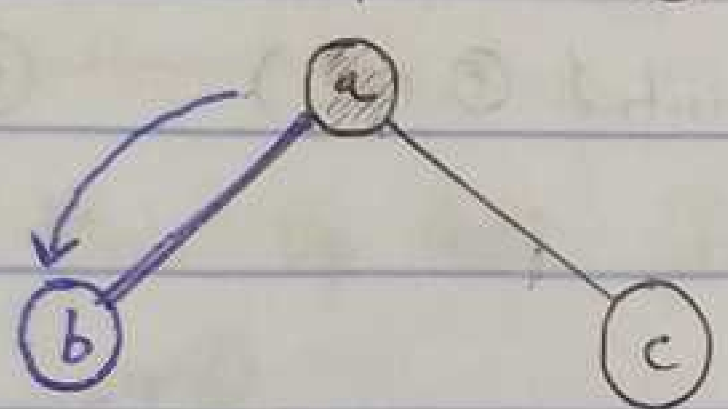
RULE

• Root , Left, Right
1 2 3

(a)

Preorder : a

Move Left (b) (Left)



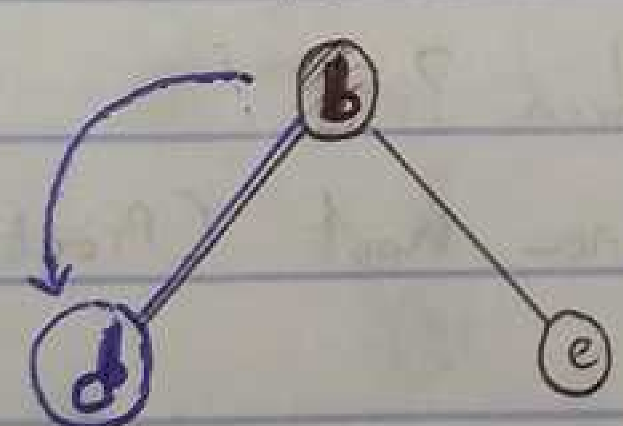
Add b

Preorder : a b

Ask : Does (b) have any children ? Ans : Yes,

Since it has children, (b) is our new Root (Root)

Move Left (d) (Left)



Add d

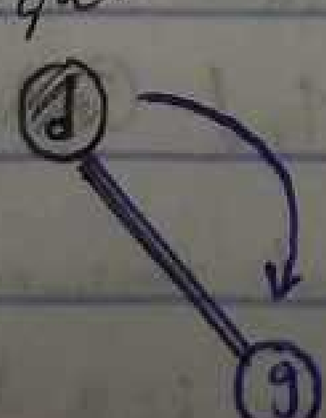
Preorder : a b d

Ask : Does (d) have any children ? Ans : Yes,

Since it has children, (d) is our new Root (Root)

Move Left No Left

Move Right (g) (Right)

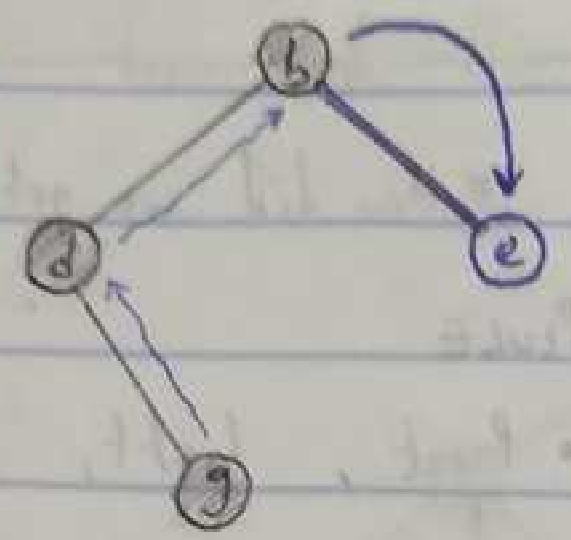


Add g

Preorder : a b d g

g

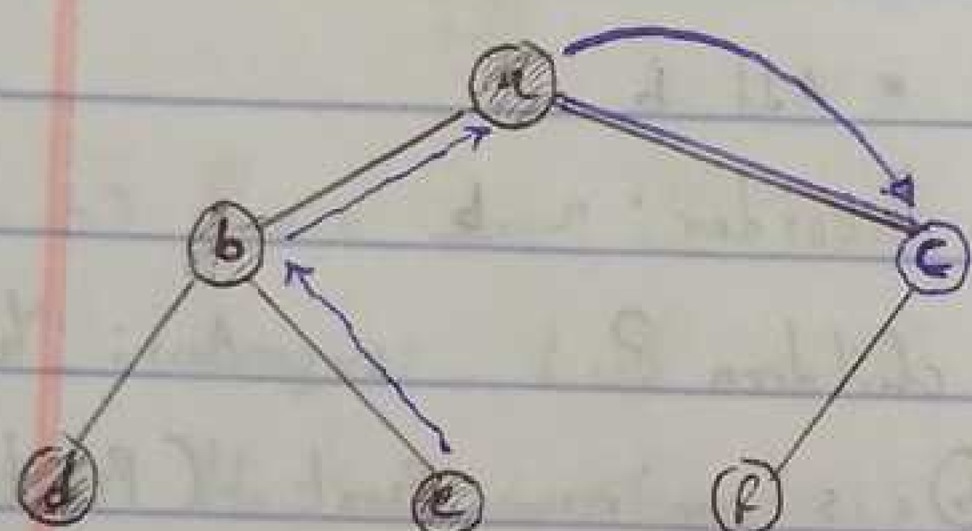
Ask: Does g have any children? Ans: No
Move Right (to the nearest unvisited ? Node) e (Right)



Add e

Preorder: a b d g e

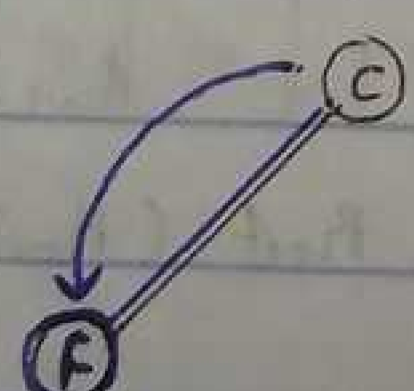
Ask: Does e have any children? Ans: No
Move Right (to the nearest unvisited ? Node) c (Right)



Add c

Preorder: a b d g e c

Ask: Does c have any children? Ans: Yes
Since it has children, c is our new Root (Root)
Move Left f (Left)



Add f

Preorder: a b d g e c f

Ask: Does f have any children? Ans: No
Move Right (to the nearest unvisited ? Node) No Right

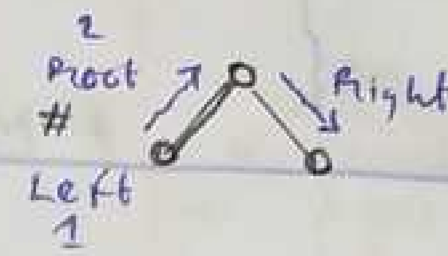
Final answer

DONE

Preorder: a b d g e c f

Messi ∈ ω(Ronaldo) . ω means >

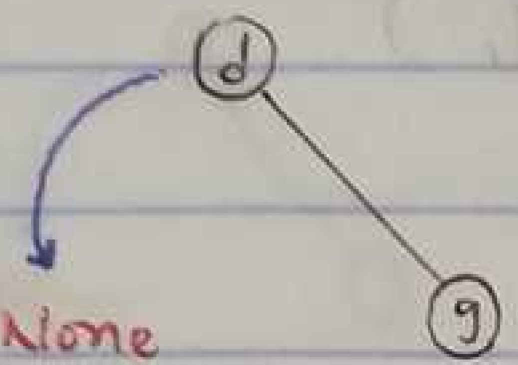
2 • Inorder Traversal (Left, UP or Root, Right)



d, g, b, e, a, f, c

EXPLANATION

We start at the leftmost node ⊗ (Left)



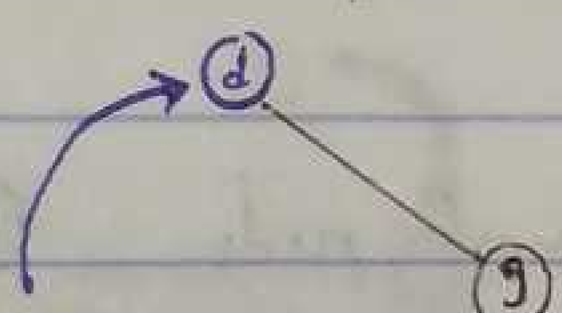
None
(No Leftmost Node)

RULE

∘ Left, Root, Right
1 2 3

Inorder:

Move Up ~~Root~~ d (Root)

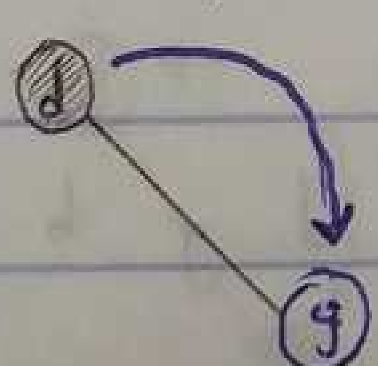


Add d

Inorder : d

Move Right and then to the leftmost node None (No Node)

Move Right g (Right)

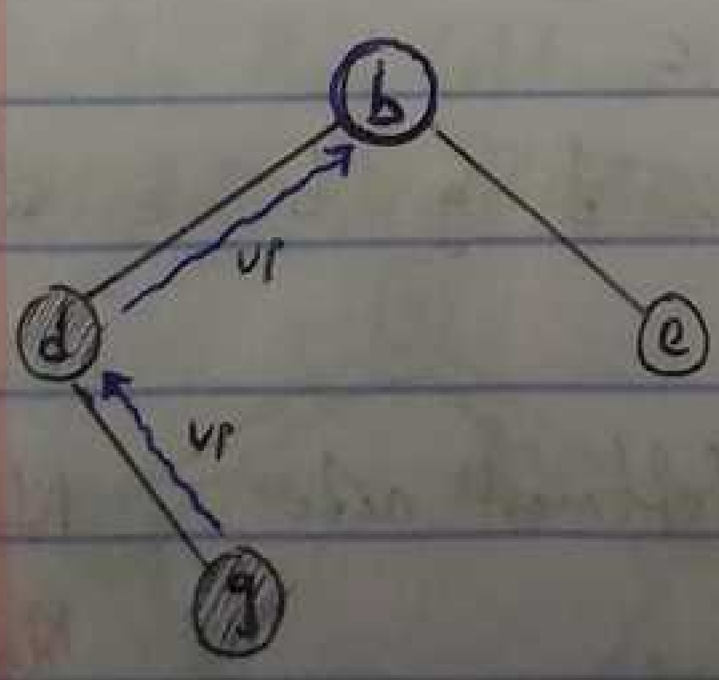


Add g

Inorder : d g

#

Move Up (to the 1st unvisited node found)

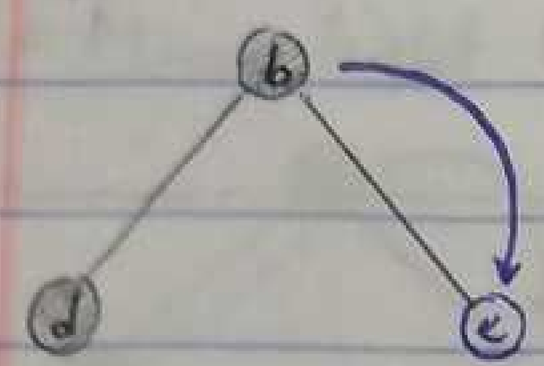


Add b

Inorder : d g b

Move Right and then move to thea leftmost node None (No Node)

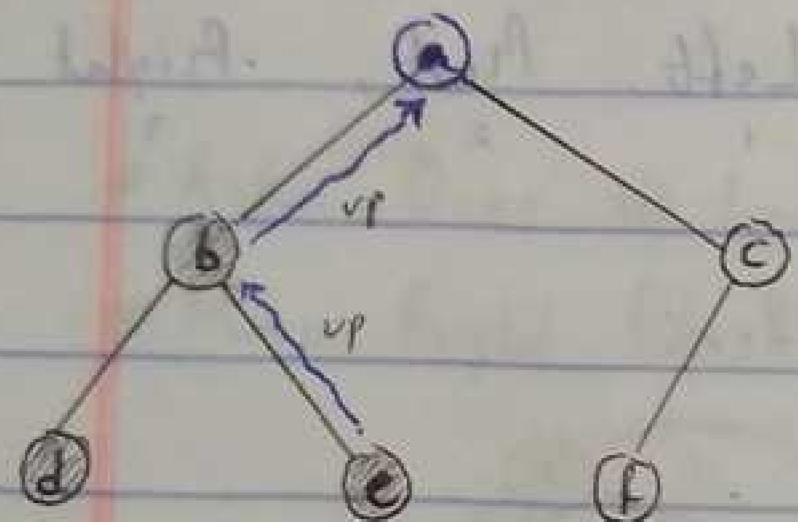
Move Right (e) (Right)



Add e

Inorder : d g b e

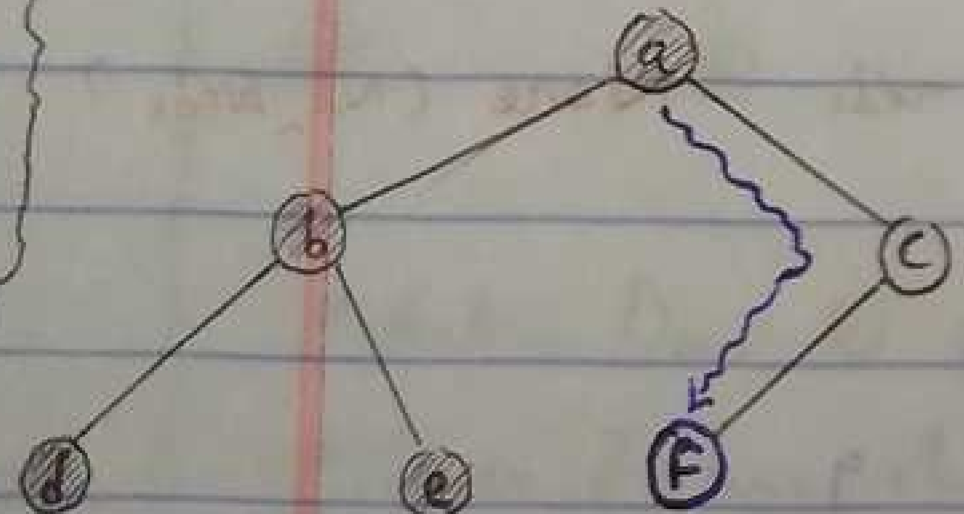
Move Up (to the 1st unvisited node)



Add a

Inorder : d g b e a

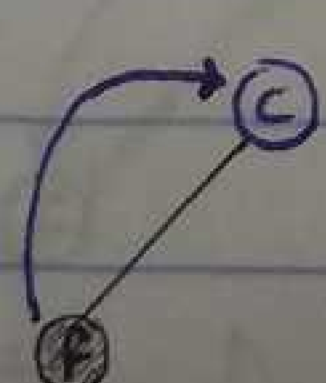
Move Right and then to the leftmost node (F) (Left)



Add f

Inorder : d g b e a f

Move Up (C) (Root)



Add c

Inorder : d g b e a f c

Move Right and then to the leftmost node None (No node)

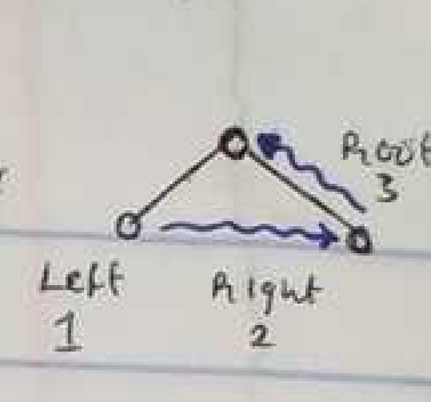
Move Right None (No node)

Move Up (to the 1st unvisited node) — None (No node)

DONE

Inorder : d g b e a f c

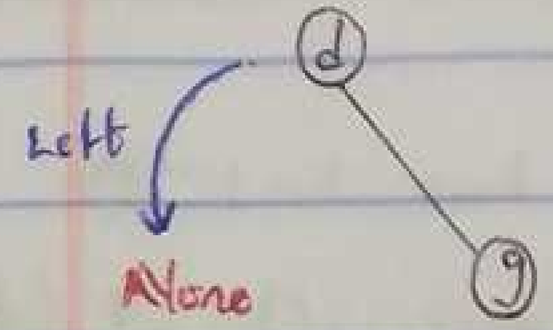
3 • Postorder Traversal (Left, Right, Root) #



g, d, e, b, f, c, a

EXPLANATION

We start at the leftmost Node ⓧ (Left)

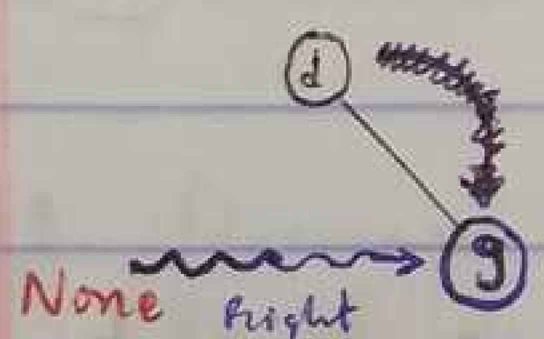


RULE

• Left, Right, Root

Postorder:

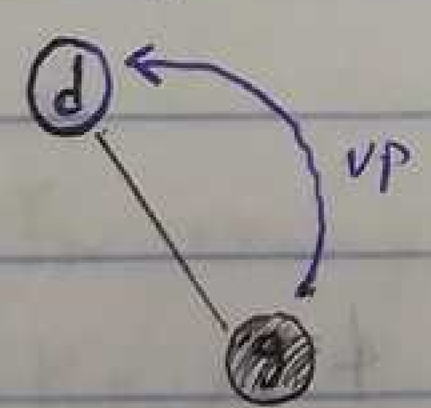
Move Right ⓖ (Right)



Add g

Postorder: g

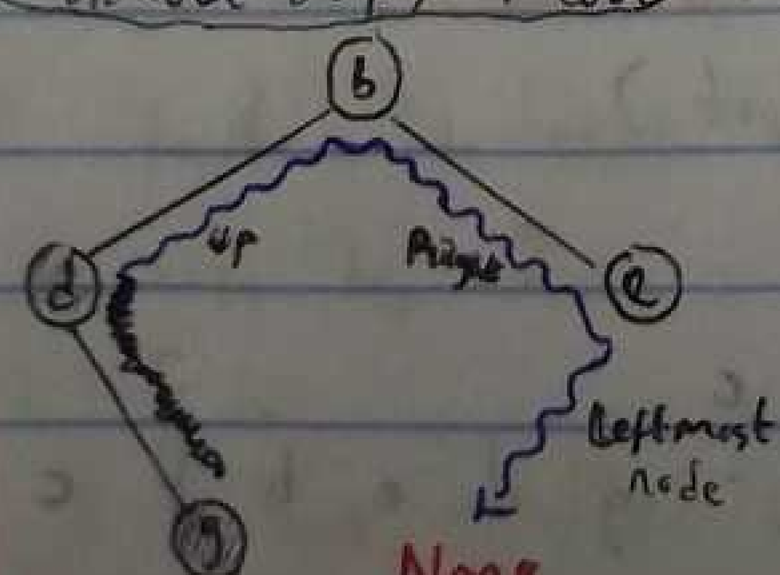
Move Up ⓓ (Root)



Add d

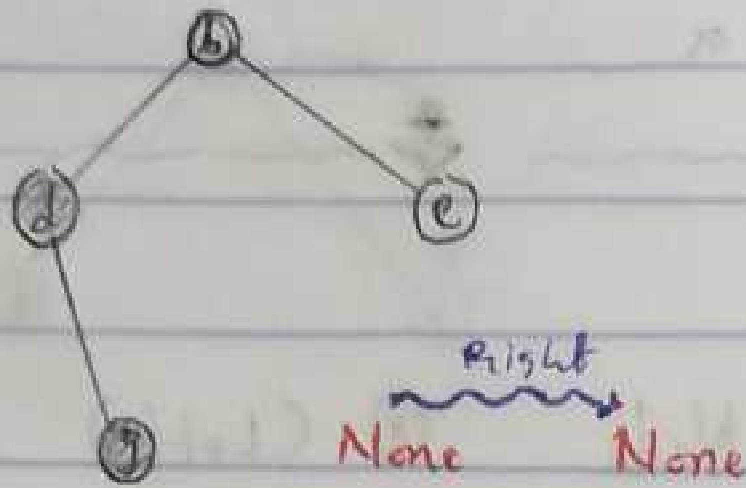
Postorder: g d

Move Up, Move Right and then move to the leftmost node None (No node)



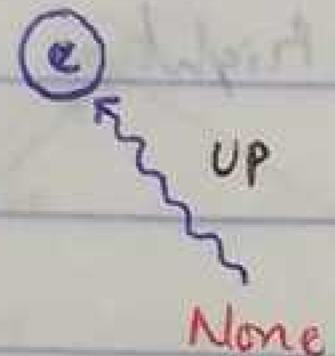
Postorder: g d

Move Right None (No node)



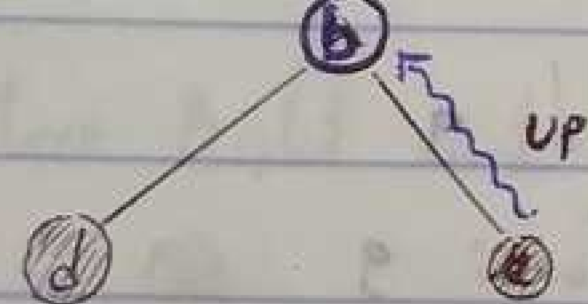
Postorder: g d

Move Up ⓔ ~~Root~~ (Root)



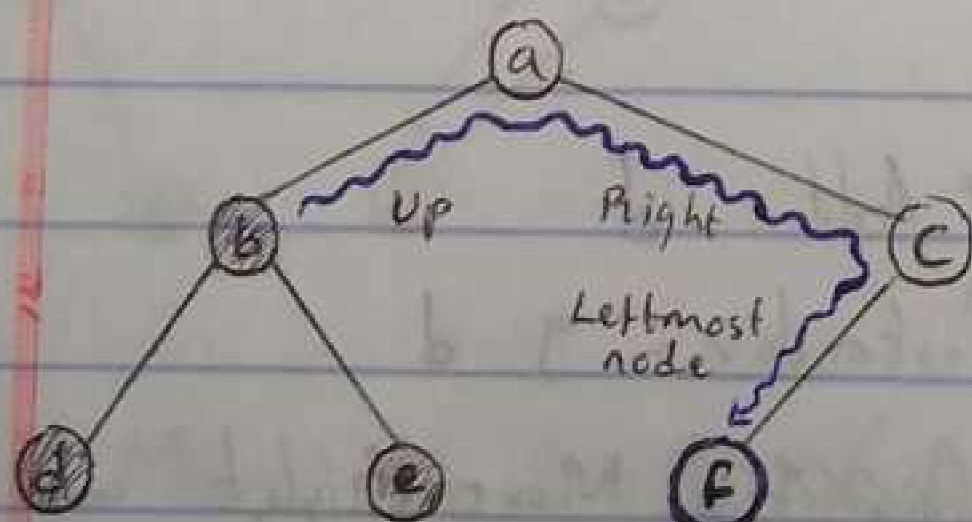
Add e
Postorder: g d e

Move Up ~~Move Right and then move to the leftmost node~~



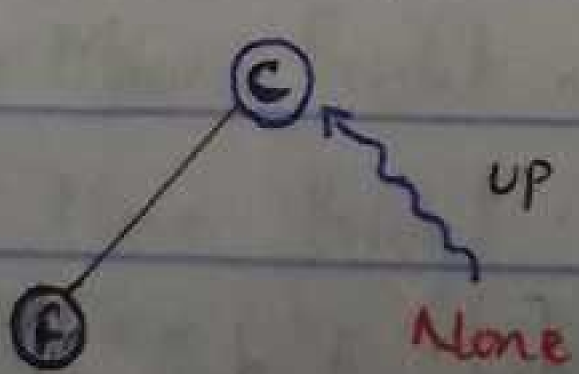
Add b
Postorder: g d e b

Move Up, Move Right and then move to the leftmost node



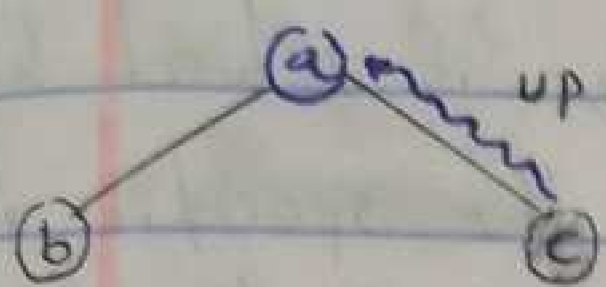
Add f
Postorder: g d e b f

Move Right None (No node)
Move Up ⓒ (Root)



Add c
Postorder: g d e b f c

Move Up



Add a

Postorder: g d e b f c a

DONE

Postorder: g d e b f c a

Q: List and explain the 3 decrease and conquer implementation method

b) 1. Decrease by a Constant
2. Decrease by a Constant factor
3. Decrease by Variable Size

1. Decrease by a Constant
In this method, the problem size is reduced by a constant factor in each step, typically by 1.
E.x
For example, in an iterative linear search, the size of the problem by 1 element after each comparison

2. Decrease by a Constant factor
In this method, the problem size is reduced by a constant factor in each step, typically by dividing the problem size by 2.
E.x
This is often seen in algorithms that split the problem in half

RHS : Right Hand Side
LHS : Left Hand Side

3. Decrease by Variable Size
In this method, the problem size is reduced by a variable amount depending on the specific instance of the problem. The amount by which the problem is reduced is not fixed, it can vary with each step.

c) i) Search for $k = 70$

$A = [3, 14, 27, 31, 39, 42, 55, 70, 74, 81, 85, 93, 98$

0	1	2	3	4	5	6	7	8	9	10	11	12

- Iteration 1

\# Find the middle index (m)

$m = n / 2$ # n is the number of elements in A

$= 13 / 2$

$= 6.5$ # Discard .5

$= 6$

\# Get middle value

$V = A[m]$

$= A[6]$

$= 55$

\# Check if, $k > V$. We pick RHS
\# Check if, $k < V$. We pick LHS
\# Check if, $k = V$. We stop

k = 70

V = 55

~~Therefore~~ Since k > V so we pick RHS

RHS

A = [3, 14, 27, 31, 39, 42, 55, 70, 74, 81, 85, 93, 88]

0 1 2 3 4 5 6 7 8 9 10 11 12

#New array A = [RHS]

A = [70, 74, 81, 85, 93, 88]

0 1 2 3 4 5

- Iteration 2

Find the middle index (m)

m = n / 2 # n is the number of elements in New A

= 6 / 2

= 3

Get middle value (V)

V = A[m]

= A[3]

= 85

Check if, k > V. We pick RHS

Check if, k < V. We pick LHS

Check if, k = V. We stop

k = 70

V = 85

Since k < 85 we pick LHS

A = [70, 74, 81, 85, 93, 88]
0 1 2 3 4 5

# New array A = [LHS]
A = [70, 74, 81]
0 1 2

- Iteration 3

Find the middle index (m)

m = n / 2 # n is the number of elements in New A
= 3 / 2
= 1.5 # Discard .5
= 1

Get middle value (V)

V = A[m]
V = A[1]
V = 74

Check if, k > V . We pick RHS
Check if, k < V . We pick LHS
Check if, k = V . We stop

k = 70
V = 74

Since k < V we pick LHS

A = [70, 74, 81]
0 1 2

New array $A = [LHS]$

$A = [\underset{0}{70}]$

- Iteration 4

Since we only have just 1 element $V = A[0]$
$= 70$

Check if, ...

...

$k = V$. We stop

$k = 70$

$V = 70$

Since $k = V$ we stop

Therefore, it took 4 iteration to search for $k = 70$

It took 4 iterations to find $k = 70$ //

ii) Search for $k = 85$

$$A = [\underset{0}{3}, \underset{1}{14}, \underset{2}{27}, \underset{3}{31}, \underset{4}{39}, \underset{5}{42}, \underset{6}{55}, \underset{7}{70}, \underset{8}{74}, \underset{9}{81}, \underset{10}{85}, \underset{11}{93}, \underset{12}{88}]$$

- Iteration 1

Find the middle index (m)

$m = n / 2$ # n is the number of elements in A

$= 13 / 2$

$= 6.5$ # Discard .5

$= 6$

Get middle ~~index~~ value (V)

~~# Get middle value (V)~~
~~V = A~~

V = A [m]

= A [6]

= 55

Check if, k > V. We pick RHS

Check if, k < V. We pick LHS

Check if, k = V. We stop

k = 85

V = 55

Since k > V. We pick RHS

A = [3, 14, 27, 31, 39, 42, 55, 70, 74, 81, 85, 93, 88]

A =	3	14	27	31	39	42	55	70	74	81	85	93	88
	0	1	2	3	4	5	6	7	8	9	10	11	12

New array A = [RHS]

A = [70, 74, 81, 85, 93, 88]

A =	70	74	81	85	93	88
	0	1	2	3	4	5

- Iteration 2

Find middle index (m)

m = n / 2

= 6 / 2

= 3

Get middle value (V)

V = A [m]

= A [3]

= 85

# Check if, k > V. We pick RHS
# Check if, k < V. We pick LHS
# Check if, k = V. We stop
k = 85
V = 85

Since k = V. We stop
Therefore it took 2 iterations to search for k = 85

It took 2 iterations to find k = 85 //

iii) Search for k = 31

A = [	3,	14,	27,	31,	39,	42,	55,	70,	74,	81,	85,	93,	88	]
	0	1	2	3	4	5	6	7	8	9	10	11	12	

- Iteration 1

Find the middle index (m)

m = n / 2 # n is the number of elements in A
= 13 / 2
= 6.5 # Discard .5
= 6

Get middle value (V)

V = A[m]
= A[6]
= 55

Check if k > V. We pick RHS
Check if k < V. We pick LHS
Check if k = V. We stop

k = 31

V = 55

Since k < V. We pick LHS

A = [3, 14, 27, 31, 39, 42, 55, 70, 74, 81, 85, 93, 88]

0 1 2 3 4 5 6 7 8 9 10 11 12

New array A = [LHS]

A = [3, 14, 27, 31, 39, 42]

0 1 2 3 4 5

- Iteration 2

Find the middle index (m)

m = n / 2

~~= 13 / 2~~ = 6 / 2

~~= 6.5~~ = 3

~~= 6~~

Get the middle value (v)

V = A [m]

= A [3]

= 31

Check if k > V. We pick RHS
Check if k < V. We pick LHS
Check if k = V. We stop

$k = 31$

$V = 31$

Since $k = V$. We stop

Therefore it took 2 iterations to search for $k = 31$

It took 2 iterations to find $k = 31$ //

Question (Five) 5: Divide and Conquer

a) Given the general condition of divide-and-conquer recurrence relationship as $T(n) = aT(\frac{n}{b}) + f(n)$, such that $a \geq 1$, $b > 1$.

State the master theorem

b) Use the Masters Theorem to derive the complexity class of the following functions

i. $T(n) = 8T(\frac{n}{2}) + 1$

ii. $T(n) = 2T(\frac{n}{6}) + n^3$

c) Sort the array below using merge sort algorithm. Make sure you show each steps of divide and conquer

9	4	3	10	8	2	6	4
0	1	2	3	4	5	6	7

a) Master Theorem states that the Time Complexity $T(n)$ can be determined based on the below

- $T(n) = aT(\frac{n}{b}) + f(n), \quad a \geq 1, \quad b > 1$
- If $f(n) \in \Theta(n^d)$ where $d \geq 0$ then

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \operatorname{Log} n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Note

$\boxed{T(n) = aT(\frac{n}{b}) + n^d}$

b) i) $T(n) = 8T(\frac{n}{2}) + 1$

Find a, b, d

$a = 8$

$b = 2$

$d = 1^{0} = 0$

Check if $a < b^d$. Answer will be $\Theta(n^d)$

Check if $a = b^d$. Answer will be $\Theta(n^d \operatorname{Log} n)$

Check if $a > b^d$. Answer will be $\Theta(n^{\log_b a})$

$a = 8$
$b^d = 2^0 = 1$

\# Therefore $a > b^d$, so our answer is $\Theta(n^{\log_b a})$

$T(n) \in \Theta(n^{\log_b a})$ \# $\in$ means = or is

\# Further solve $n^{\log_b a}$ for mor marks

$= n^{\log_b a}$

\# Where $a = 8$ and $b = 2$

$= n^{\log_2 8}$
$= n^{(\log 8)/(\log 2)}$
$= n^3$

\# Final Answer

$T(n) \in \Theta(n^3)$ //

ii) $T(n) = 2T(\frac{n}{6}) + n^3$

Find a, b, d;

$a = 2$

$b = 6$

$d = 3$

Check if $a < b^d$. Answer will be $\Theta(n^d)$

Check if $a = b^d$. Answer will be $\Theta(n^d \log n)$

Check if $a > b^d$. Answer will be $\Theta(n^{\log_b a})$

$a = 2$

$b^d = 6^3 = 216$

Therefore $a < b^d$, so our answer is $\Theta(n^d)$

$T(n) \in \Theta(n^d)$

Further solve n^d for more marks

$= n^d$

Where $d = 3$

$= n^3$

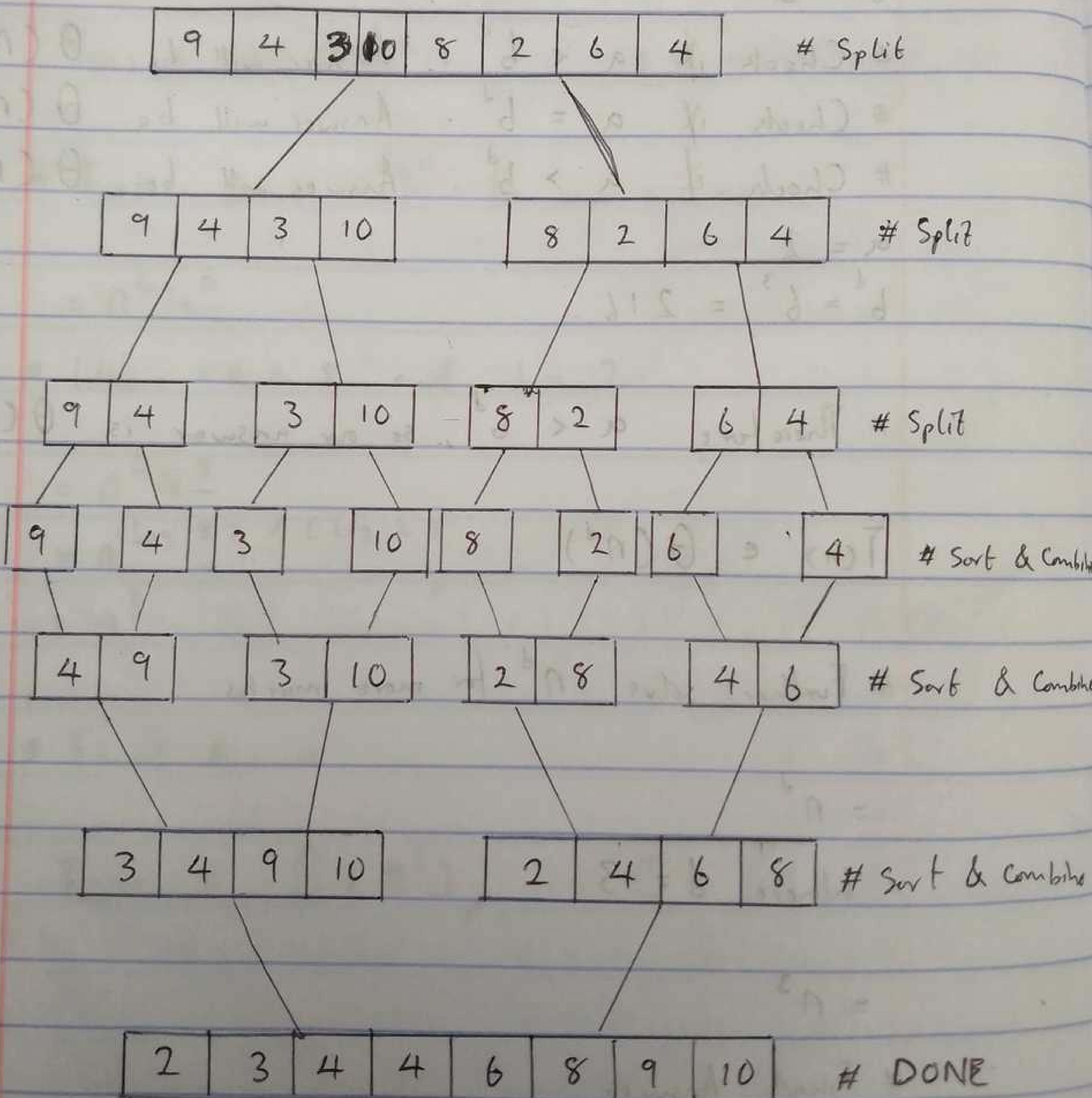
Final Answer

$T(n) \in \Theta(n^3)$

c) Sort the below array using merge sort

A = [9 | 4 | 3 | 10 | 8 | 2 | 6 | 4]

Start sorting array A



Our array A is sorted