

Algorithms and Coding Patterns

Guru Sarath

TEXAS A&M
UNIVERSITY

BINARY SEARCH

↑ start ↑ mid ↑ end

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

key: 2

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

$$\text{mid} = \frac{0+2}{2} = 1$$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

key: 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

$$\text{mid} = \frac{0+2}{2} = 1$$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

$$\text{mid} = \frac{0+0}{2} = 0$$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

key: 5

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

$$\frac{4+7}{2} = \frac{11}{2} = 5.5$$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

$$\frac{4+6}{2} = 5$$

$$\frac{4+5}{2} = 4.5$$

key 6.5

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

key : 1.5

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

↑ ↑ ↑

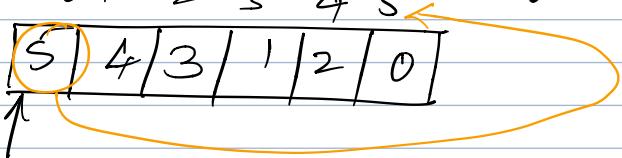
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

↑↑

Cyclic Sort

Unsorted array in range $0 - (n-1)$

Ex: 0 1 2 3 4 5



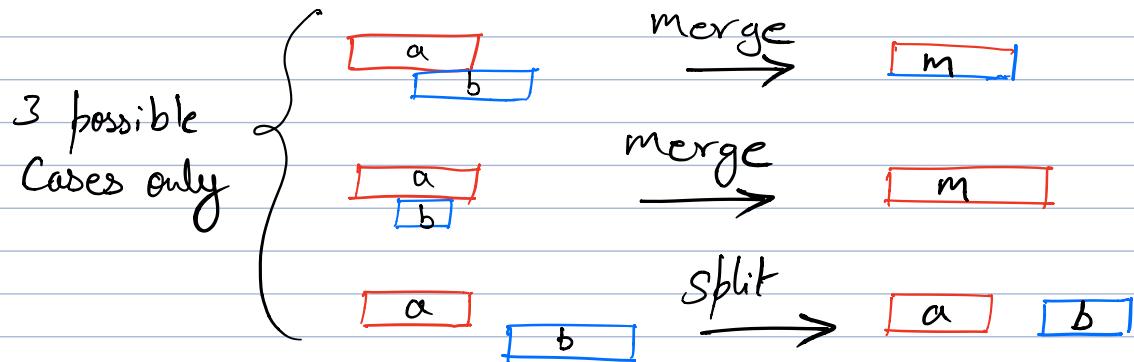
- ① check the number at the current index (5)
- ② Goto that index (index 5)
- ③ check if current value is present in that index
(At index 5, we see the value 0)
- ④ index 5 does not contain the correct value
- ⑤ so swap index 5 and 0

Dont check if the current number is in the correct position.

Check if the correct number is in the position where the current number should be.

Merge Intervals

Sort the intervals by their starting time.

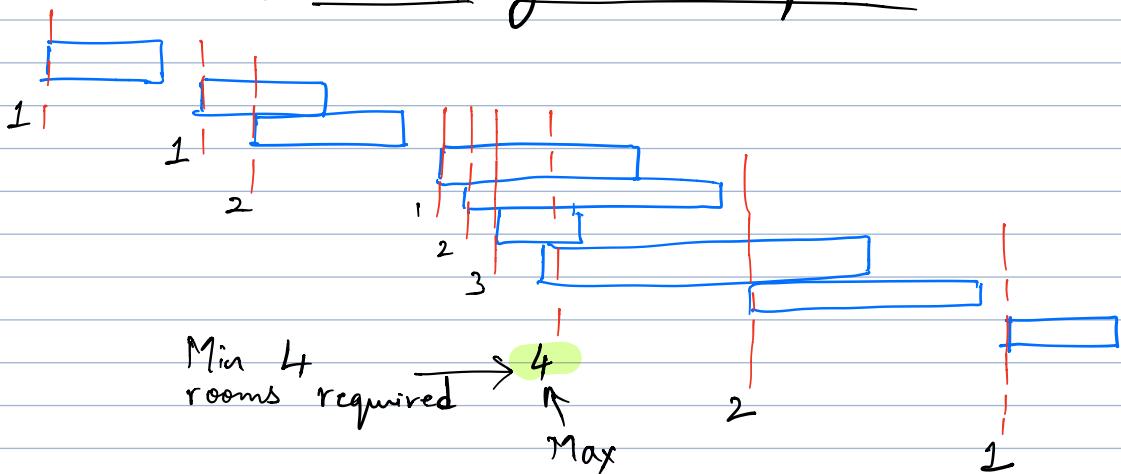


If ($b.start < a.end$) → Intervals are merged

↳ merged interval

↳ start = $a.start$
end = $\max(a.end, b.end)$

Minimum Number of rooms required



When an event starts → check how many events are still running

↳ Max of this number = Min rooms required

LINKED LIST

REVERSE A LINKED LIST



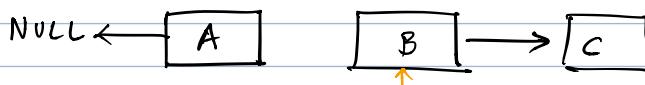
INIT

PREV = NULL
CURRENT = A

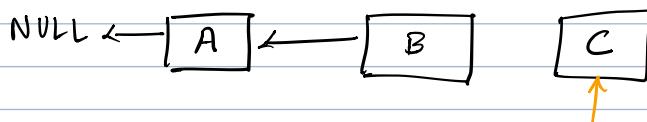
REVERSE

NEXT = CURRENT.NEXT
CURRENT.NEXT = PREV
PREV = CURRENT
CURRENT = NEXT

After iteration 1



After iteration 2



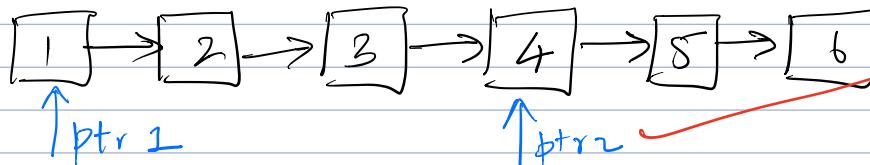
Repeat the reverse function until the current == NULL

Remove n^{th} node from the end (Only one pass)

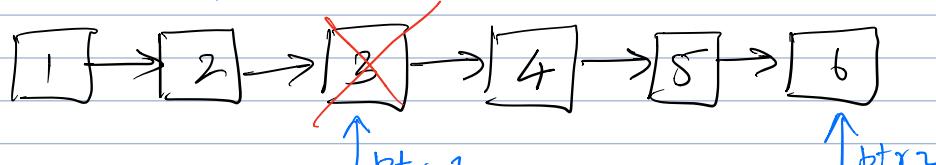
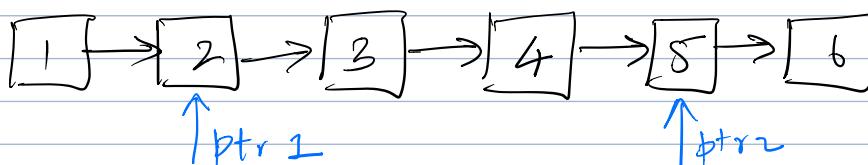


Let $n = 4$

Two pointer approach

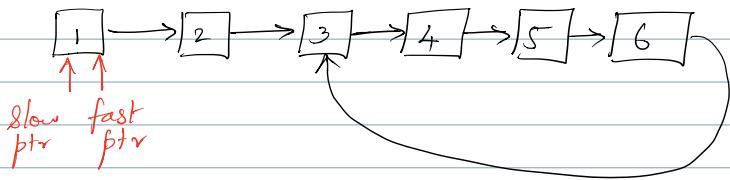


Pointer 2 starts from 4th node

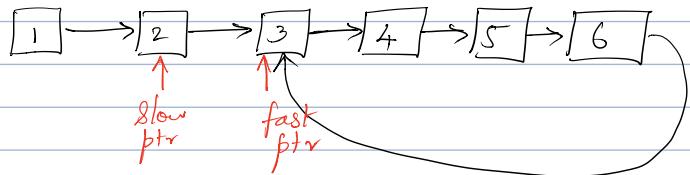


iterate until ptr2 reaches the end of the linked list

Cycle in a linked list

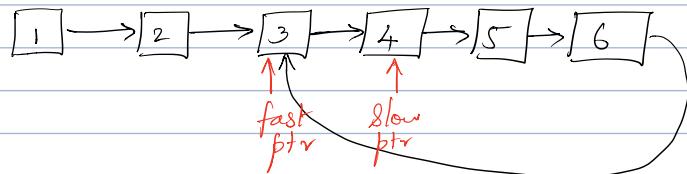
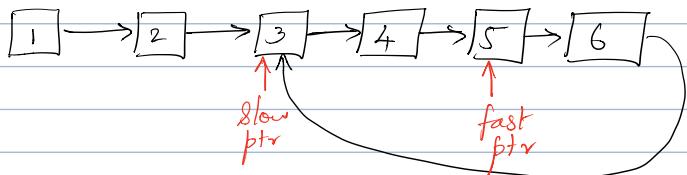


Slow pointer moves one step



fast pointer moves two steps

if there is a loop
the fast pointer will
catch up with the
slow pointer.



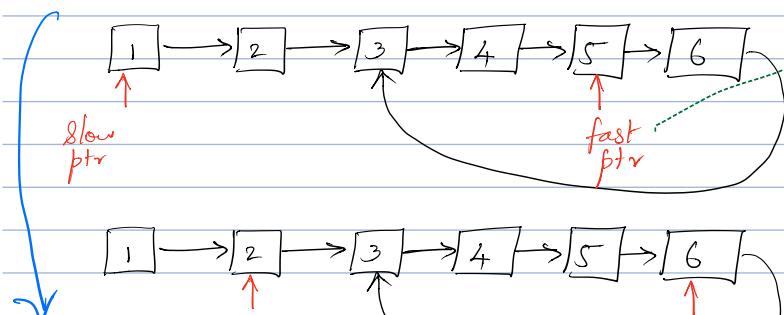
Loop detected !!

Length of cycle

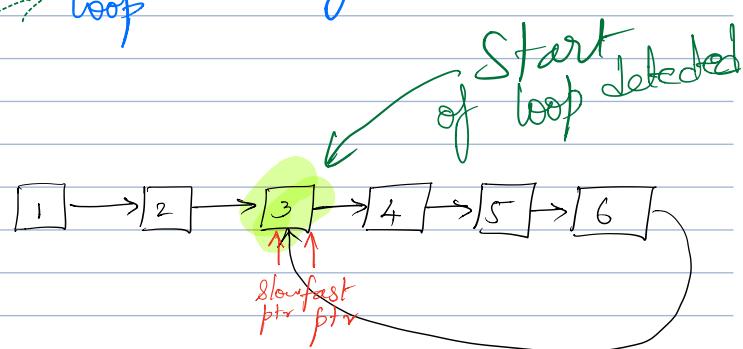
To determine the length of the cycle

- once loop is detected, move only the slow pointer counting the number of steps to reach the fast pointer again.

Start of cycle



Start with fast ptr offset by a number equal to the number of elements in the loop



Move both fast and slow points one step at a time until they meet.

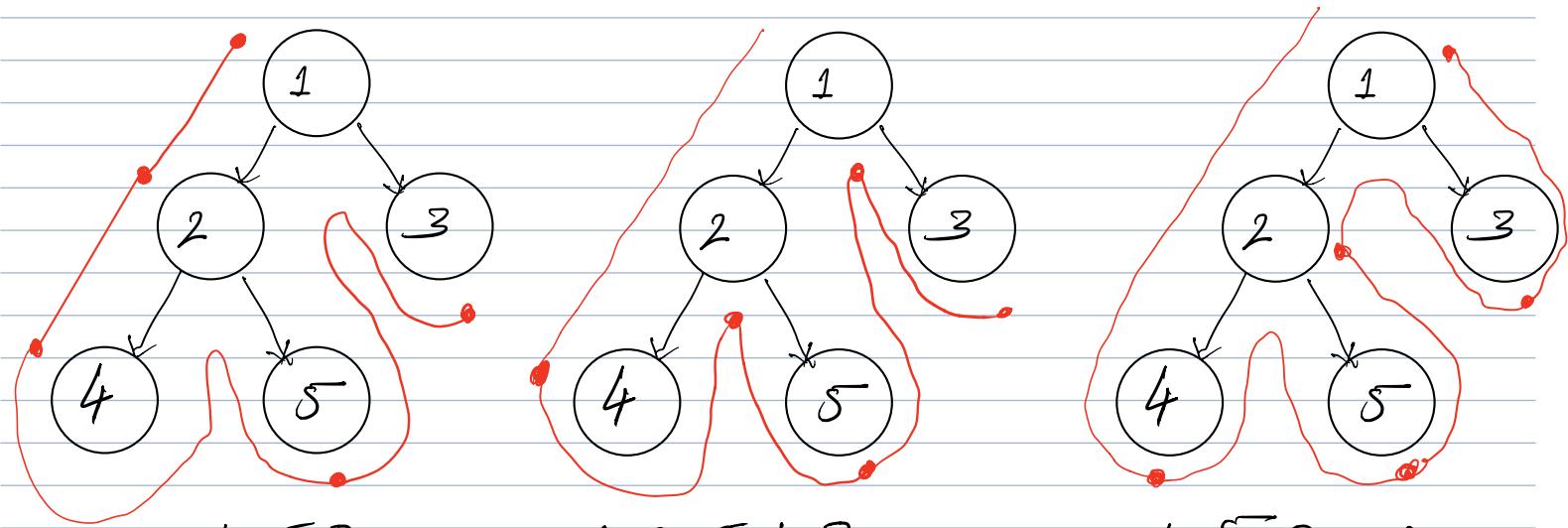
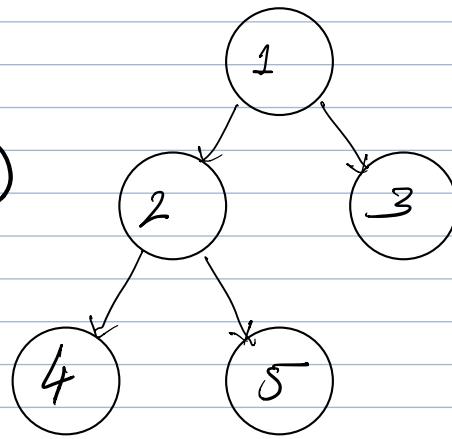
Binary Tree

DFS (Stack for implementation)

Inorder : Left Root Right

Preorder : Root Left Right

Postorder : Left Right Root

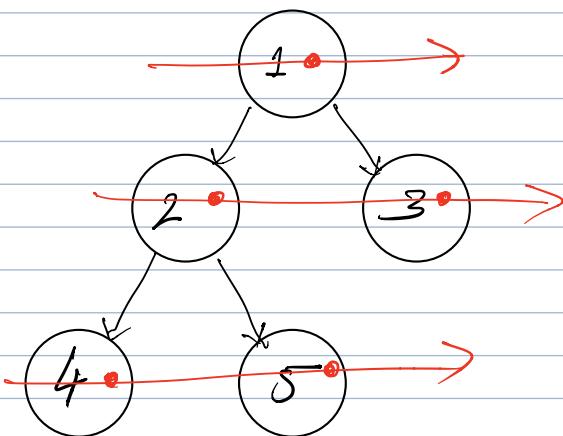


1 2 4 5 3
Preorder (DFS)

4 2 5 1 3
Inorder (DFS)

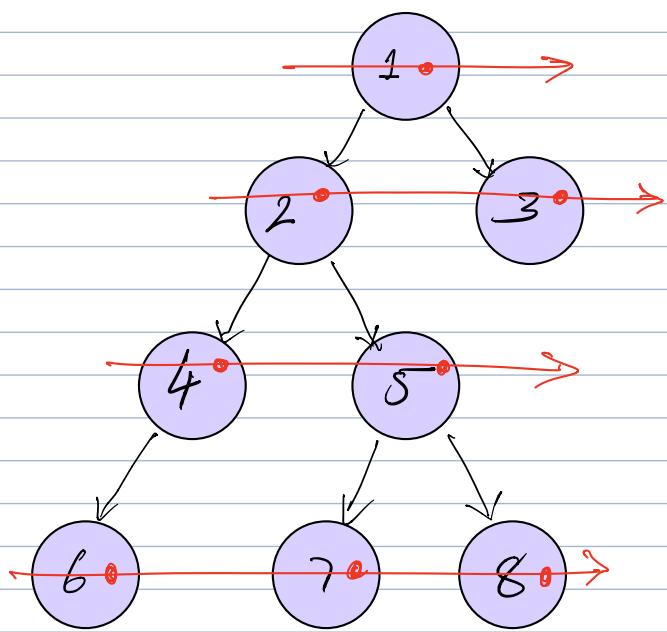
4 5 2 3 1
Postorder (DFS)

BFS (Queue for implementation)

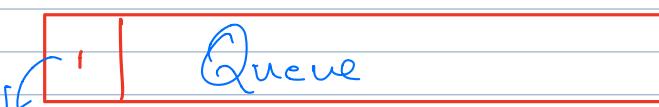


1 2 3 4 5
BFS

BFS - Iterative Solⁿ



Push ① to Queue



Pop 1

result = []

Push ② and ③ (children of 1)



result = [1, 2]



result = [1, 2, 3]



result = [1, 2, 3, 4]



result = [1, 2, 3, 4, 5]



result = [1, 2, 3, 4, 5, 6]



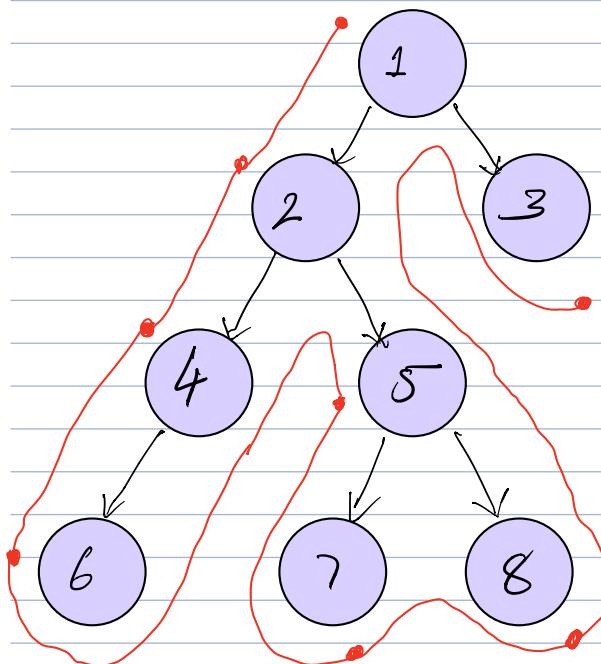
result = [1, 2, 3, 4, 5, 6, 7]



result = [1, 2, 3, 4, 5, 6, 7, 8]



DFS (Preorder Traversal)



Push)
result = [] 1 \rightarrow STACK

Pop) , add to result
and push the children of 1
to the stack

result = [1]

3 | 2 \rightarrow

result = [1, 2]

3 | 5 | 4 \rightarrow

result = [1, 2, 4]

3 | 5 | 6 \rightarrow

result = [1, 2, 4, 6]

3 | 5 \rightarrow

result = [1, 2, 4, 6, 5]

3 | 8 | 7 \rightarrow

result = [1, 2, 4, 6, 5, 7]

3 | 8 \rightarrow

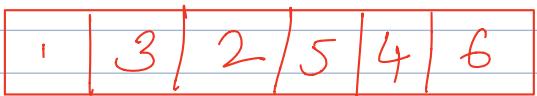
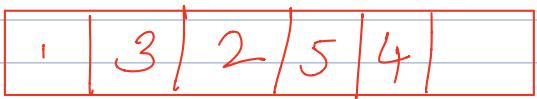
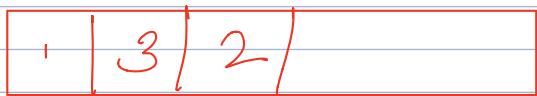
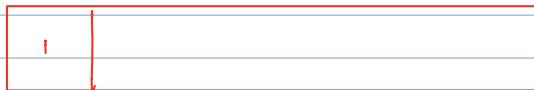
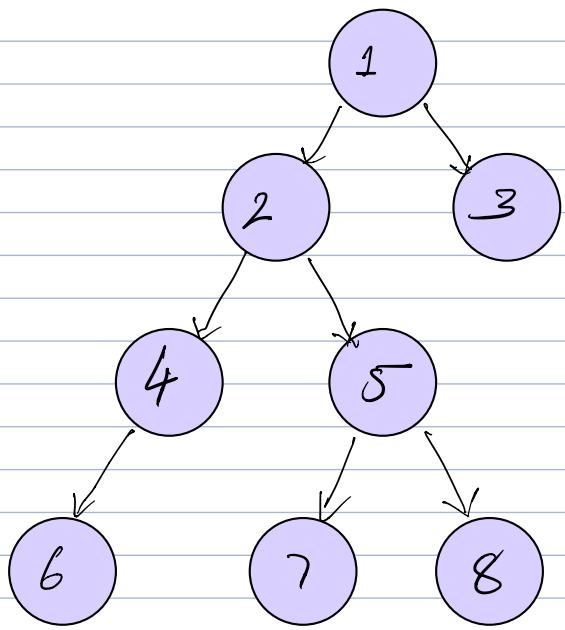
result = [1, 2, 4, 6, 5, 7, 8]

3 \rightarrow

result = [1, 2, 4, 6, 5, 7, 8, 3]

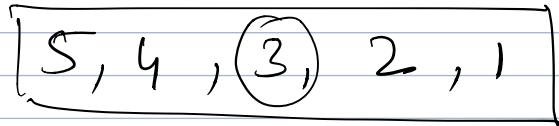
| | |

DFS



Two HEAPS

Find Median



in segt

{ Max heap (first half) Min heap (Second half)
⑤

5

$$\text{Median} = 5$$

Median = top of
larger heap

insert 4

Max heap (first half) Min heap (Second half)

$$\textcircled{5} \longrightarrow \textcircled{4}$$

Median
= Sum of tops/2

insect?

Max heap (first half) Min heap (Second half)

A diagram illustrating a directed edge between two nodes. On the left is a blue-outlined circle containing the number '4'. On the right is another blue-outlined circle containing the number '3'. A curved arrow points from the center of the '4' circle to the center of the '3' circle, indicating a directed relationship or flow from node 4 to node 3.

$$\text{Median} = 4$$

insect 2

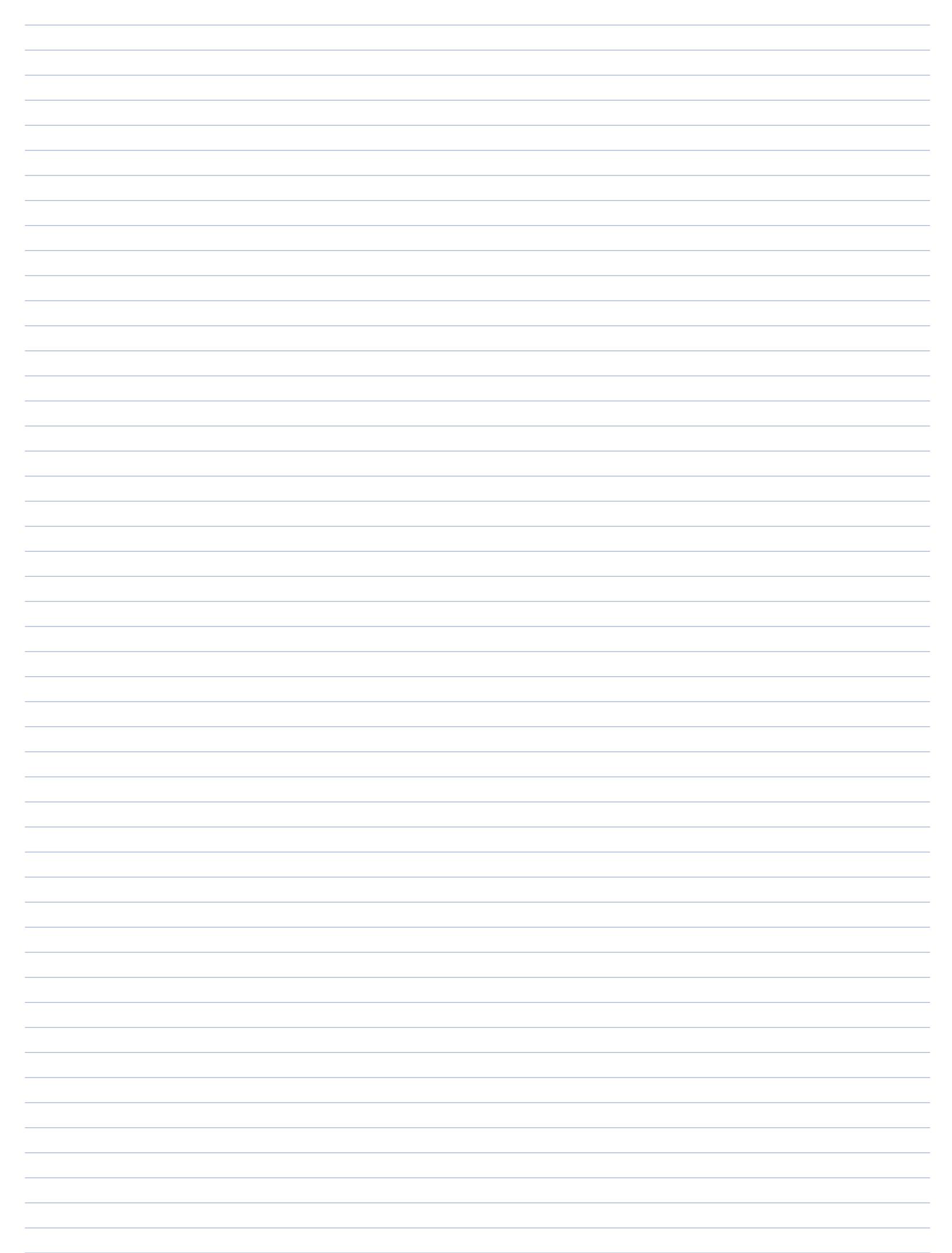
Max heap (first half) Min heap (Second half)

A hand-drawn diagram consisting of two circles, one on the left labeled '4' and one on the right labeled '3'. A curved arrow points from circle '4' to circle '3'.

Max heap (first half) Min heap (Second half)

A diagram consisting of two circles, one on the left labeled '4' and one on the right labeled '5'. A horizontal arrow points from circle '4' to circle '5'.

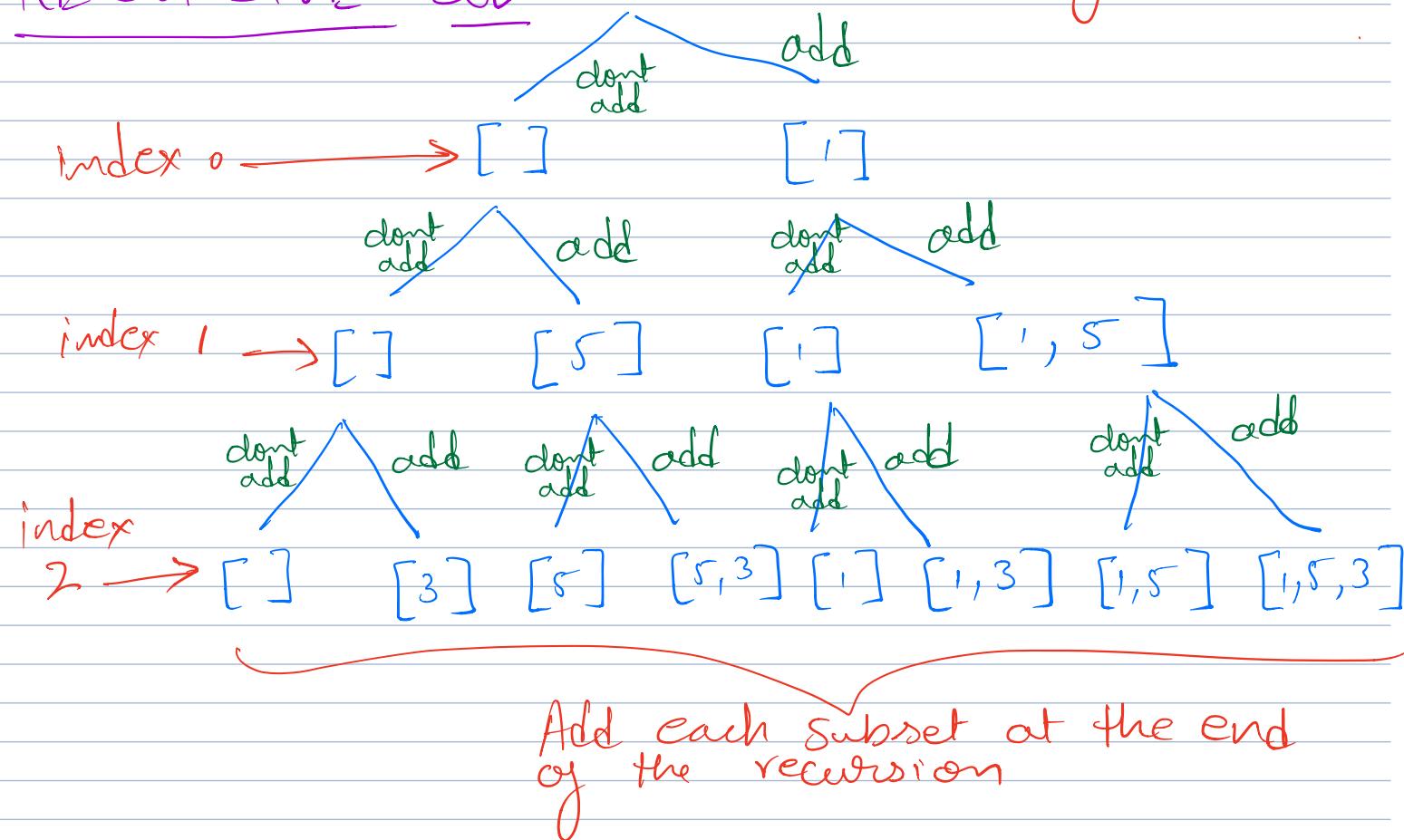
$$\text{Median} = (3+4)/2$$



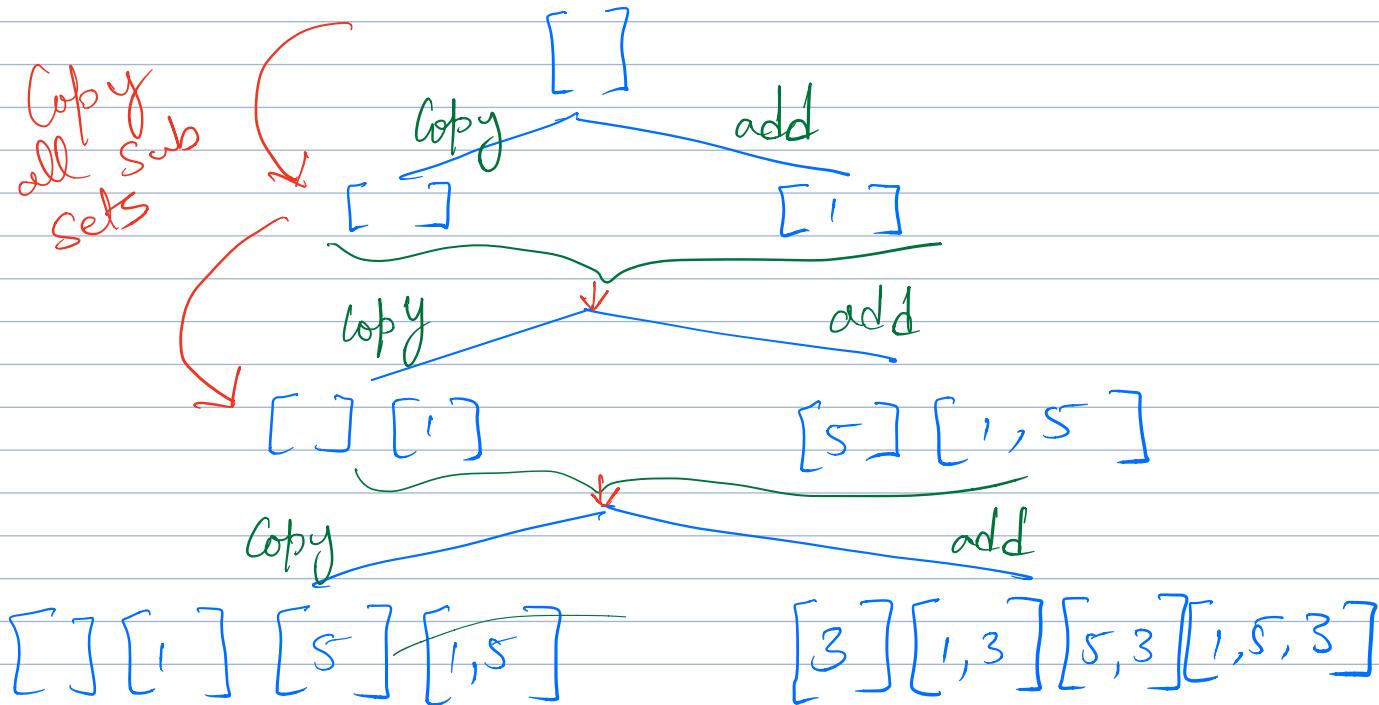
SUBSETS

Ex: $[1, 5, 3]$

RECURSIVE SOLⁿ [] ← Start of Recursion



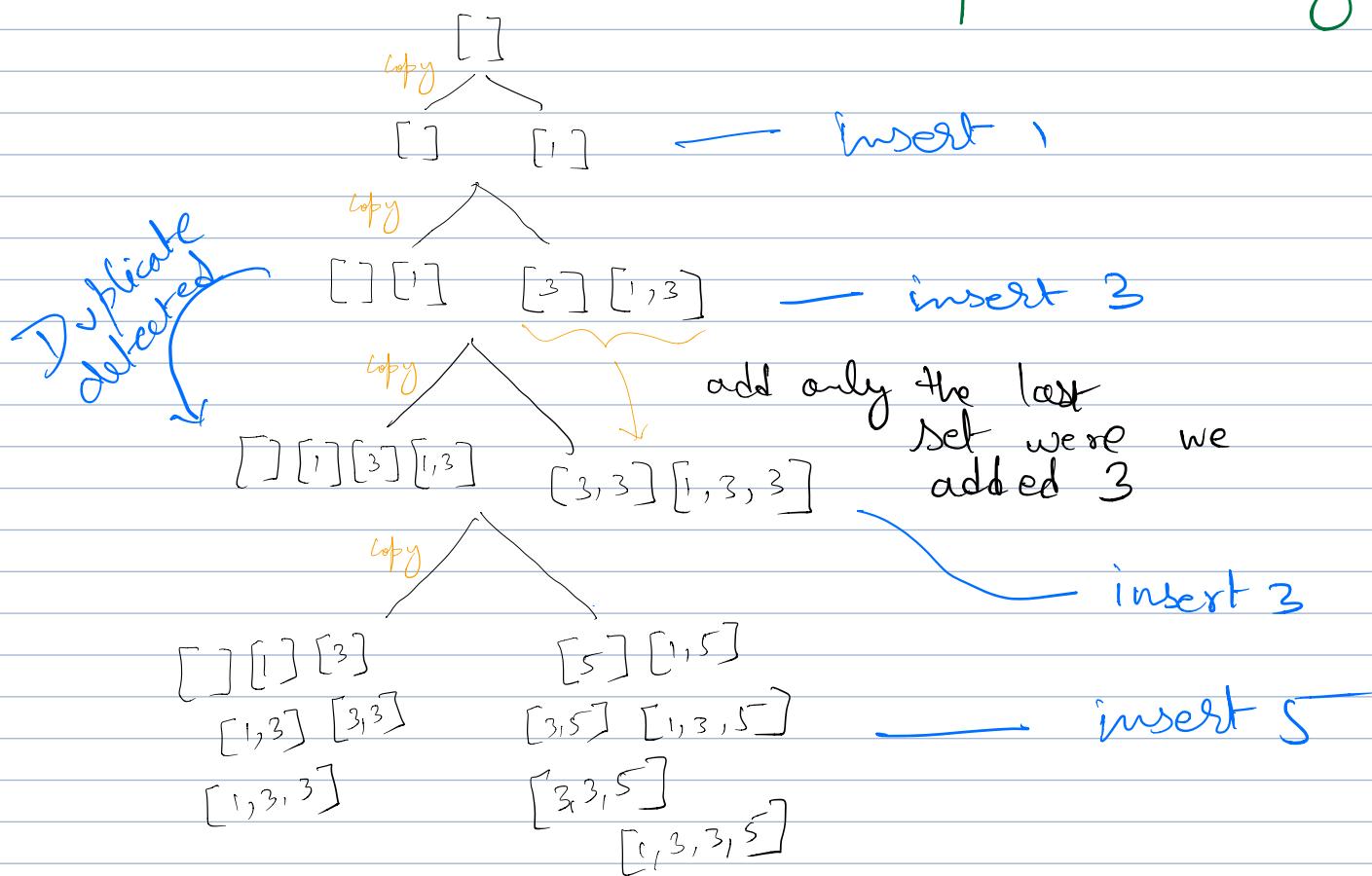
ITERATIVE SOLN



SUBSETS (with Duplicates)

Ex: $[1, 3, 3, 5]$ ←
iterative approach

Sort the input array
so that all the
duplicates are together.



PERMUTATIONS

Algorithm

Ex. $[1, 2, 3]$

① - $[]$

Start with an empty set

② - $[1]$

insert 1 (first element)

③ - $[2, 1] [1, 2]$ insert 2 in all positions

④ - $[3, 2, 1] [2, 3, 1] [2, 1, 3] [3, 1, 2] [1, 3, 2] [1, 2, 3]$

final Result

insert 3 in all positions in each of the sets

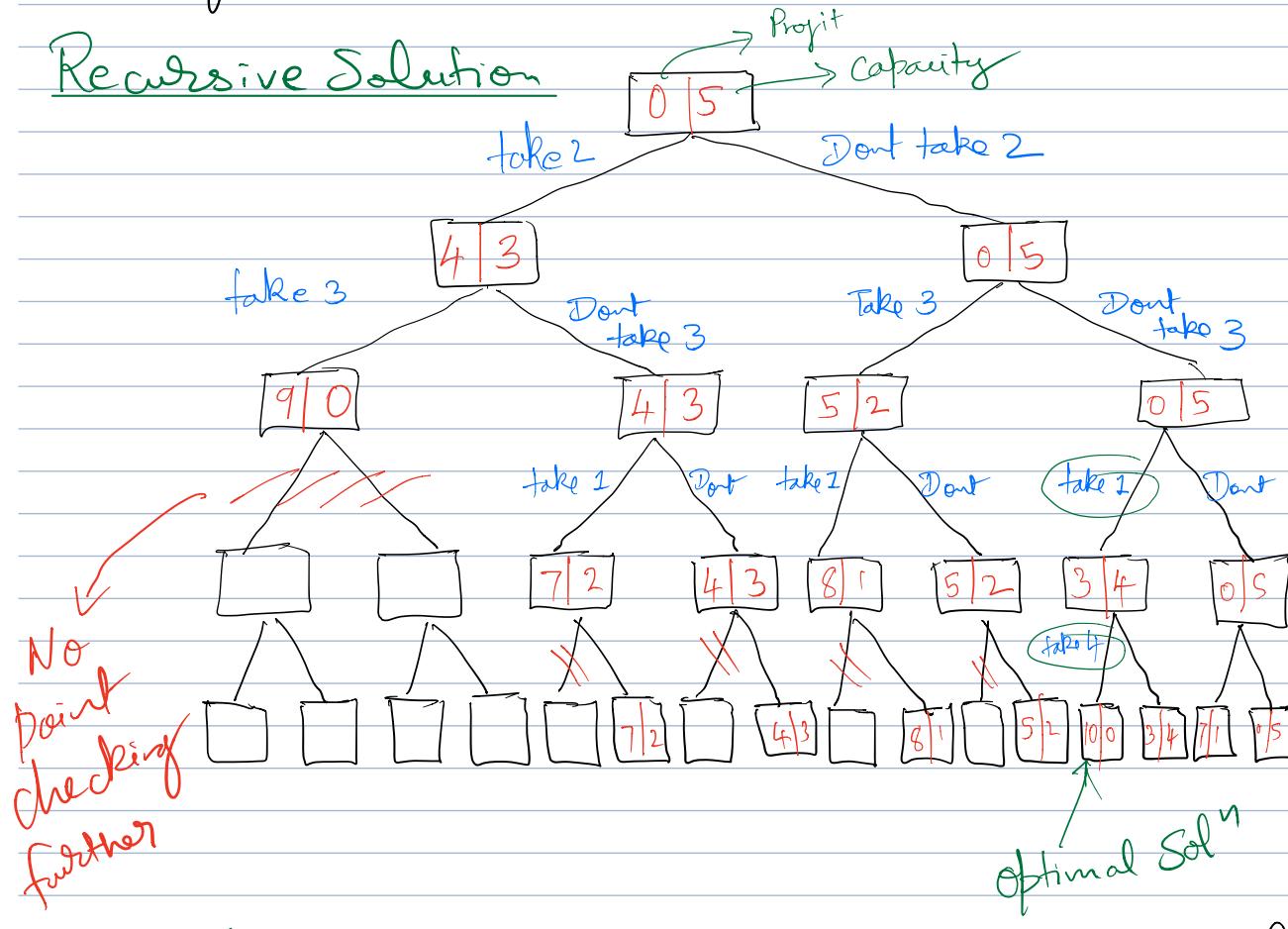
0/1 Knapsack

Weights : { 2, 3, 1, 4 }

Capacity: 5

Profits : { 4, 5, 3, 7 }

Recursive Solution



Tabulation Method

| Profit | weights | 0 | 1 | 2 | 3 | 4 | 5 | Capacity |
|--------|--------------|---|---|---|---|---|----|----------|
| 4 | {2} | 0 | 0 | 4 | 4 | 4 | 4 | |
| 5 | {2, 3} | 0 | 0 | 0 | 5 | 5 | 9 | |
| 3 | {2, 3, 1} | 0 | 3 | 3 | 5 | 8 | 9 | |
| 7 | {2, 3, 1, 4} | 0 | 3 | 3 | 5 | 8 | 10 | |

Annotations on the table:

- Row 1: Profit 4, weights {2}. A red arrow points from "weights" to the cell "4", with the text "fill in 0's" below it.
- Row 2: Profit 5, weights {2, 3}. A red arrow points from "weights" to the cell "5", with the text "fill this first" next to it.
- Row 3: Profit 3, weights {2, 3, 1}. A red arrow points from "weights" to the cell "9", with the text "Max(9, 3+7)" below it.
- Row 4: Profit 7, weights {2, 3, 1, 4}. A red arrow points from "weights" to the cell "10", with the text "Max(4, 0+5)" below it.
- Bottom left: "Not including 3" with an arrow pointing to the cell "3".
- Bottom right: "Profit of 3" with an arrow pointing to the cell "3".

Unbounded knapsack

| Profit | Weights | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|--------|------------|---|----|----|----|----|----|-----|-----|-----|
| 15 | 1 | 0 | 15 | 30 | 45 | 60 | 75 | 90 | 105 | 120 |
| 50 | 1, 3 | 0 | 15 | 30 | 50 | 65 | 80 | 100 | 115 | 130 |
| 60 | 1, 3, 4 | 0 | 15 | 30 | 50 | 65 | 80 | 100 | 115 | 130 |
| 90 | 1, 3, 4, 5 | 0 | 15 | 30 | 50 | 65 | 90 | 105 | 120 | 140 |

The difference here is that we look at the same row compared to 0/1 knap sack \rightarrow Simulates multiple takes.

$$\text{Max}(120, 80 + 50)$$

SPECIFIC LEETCODE PROBLEMS

→ Product of array elements except self

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
|---|---|---|---|

① generate left product array

| | | | | |
|------------|---|---|---|---|
| Start left | 1 | 1 | 2 | 6 |
|------------|---|---|---|---|

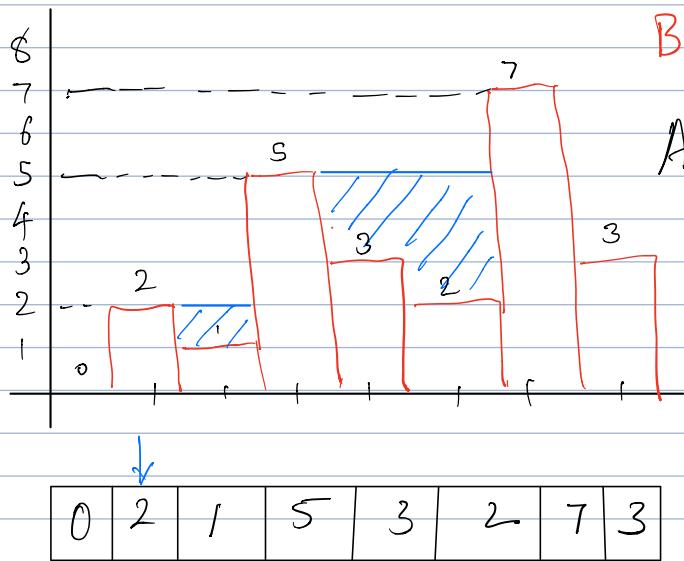
② generate right product array

| | | | |
|----|----|---|---|
| 24 | 12 | 4 | 1 |
|----|----|---|---|

③ Multiply both

| | | | |
|----|----|---|---|
| 24 | 12 | 8 | 6 |
|----|----|---|---|

→ Trapping Water



Building → 0 1 2 3 4 5 6 7
↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
Array = {0, 2, 1, 5, 3, 2, 7, 3}

Water height + building height
Water on building 1
 $W_i = 2 - 2 = 0$
 $\min(2, 7) = 2$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 2 | 1 | 5 | 3 | 2 | 7 | 3 |
|---|---|---|---|---|---|---|---|

$$\min(2, 7) = 2$$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 2 | 1 | 5 | 3 | 2 | 7 | 3 |
|---|---|---|---|---|---|---|---|

$$\min(5, 7) = 5$$

$$W_i = 5 - 5 = 0$$

$W_i = 2 - 1 = 1$ → Subtract building height

building

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 2 | 1 | 5 | 3 | 2 | 7 | 3 |
|---|---|---|---|---|---|---|---|

$$\min(5, 7) = 5$$

$$W_i = 5 - 3 = 2$$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 2 | 1 | 5 | 3 | 2 | 7 | 3 |
|---|---|---|---|---|---|---|---|

$$\min(5, 7) = 5$$

$$W_i = 5 - 2 = 3$$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 2 | 1 | 5 | 3 | 2 | 7 | 3 |
|---|---|---|---|---|---|---|---|

$$\min(7, 7) = 7$$

$$W_i = 7 - 7 = 0$$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 2 | 1 | 5 | 3 | 2 | 7 | 3 |
|---|---|---|---|---|---|---|---|

$$\min(7, 3) = 3$$

$$W_i = 3 - 3 = 0$$

$$\sum_{i=1}^n W_i = \text{Total water} = 13$$

→ Maximal Square

| | | | | |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 |

| | | | | | | | |
|--|--|--|--|--|--|--|--|
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

Reusables

