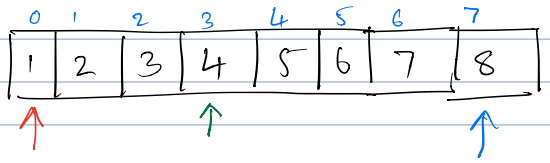
Algorithms and Coding Patterns

Guru Sarath

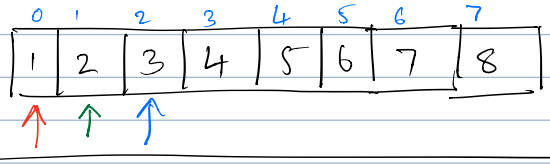
Texas A&M University

BINARY SEARCH

↑start ↑Mid ↑end



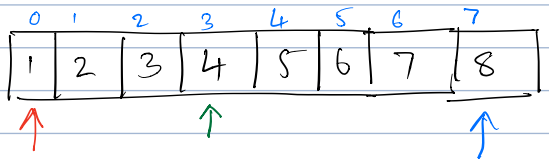
key: 2

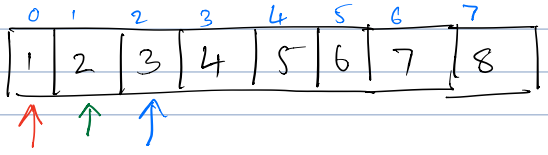


$mid = \frac{0+2}{2} = 1$

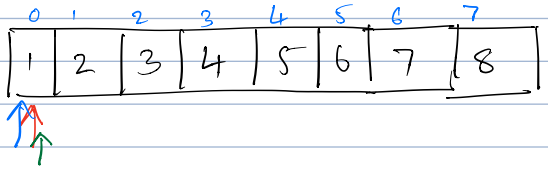
---

key: 1





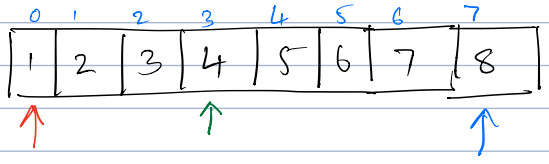
$mid = \frac{0+2}{2} = 1$

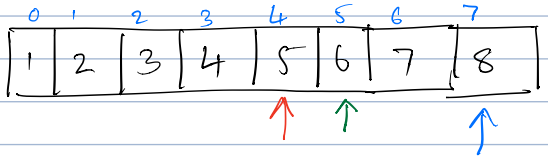


$mid = \frac{0+0}{2} = 0$

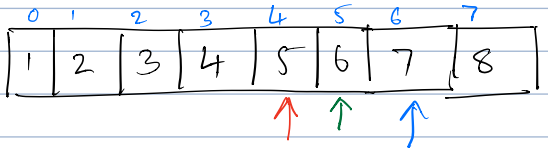
---

key: 5

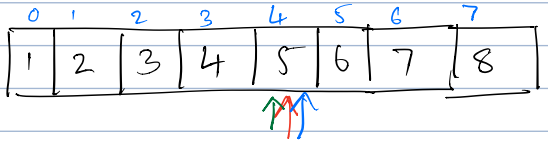




$\frac{4+7}{2} = \frac{11}{2} = 5.5$

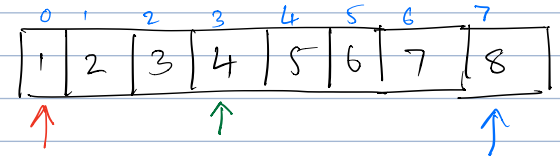


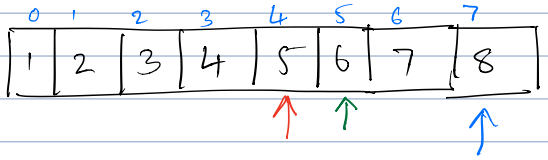
$\frac{4+6}{2} = 5$

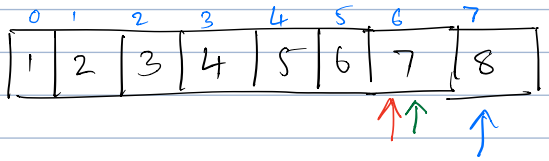


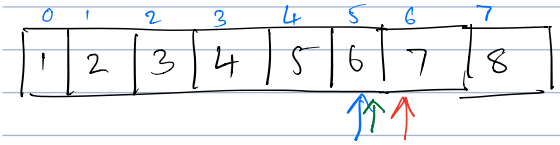
$\frac{4+5}{2} = 4.5$

key 6.5


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

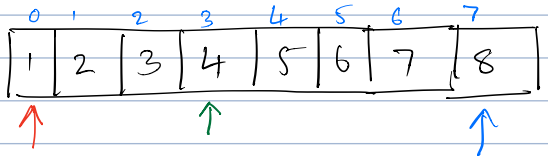

0 1 2 3 4 5 6 7
1 2 3 4 5 6 7 8

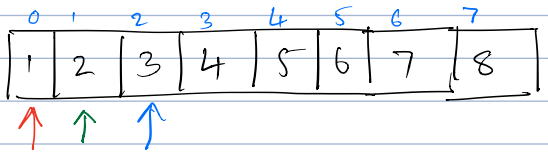

0 1 2 3 4 5 6 7
1 2 3 4 5 6 7 8

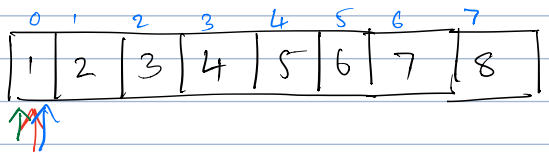

0 1 2 3 4 5 6 7
1 2 3 4 5 6 7 8

---

key : 1.5


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

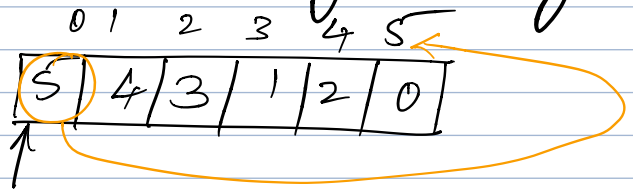

0 1 2 3 4 5 6 7
1 2 3 4 5 6 7 8


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:



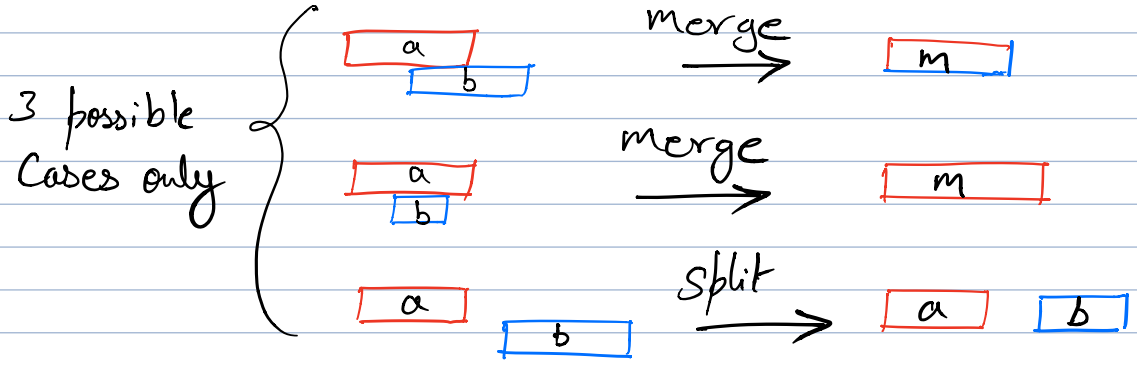
1. Check the number at the current index (5)
2. Goto that index (index 5)
3. Check if current value is present in that index (At index 5, we see the value 0)
4. index value 5 does not contain the correct
5. So swap index 5 and 0

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.



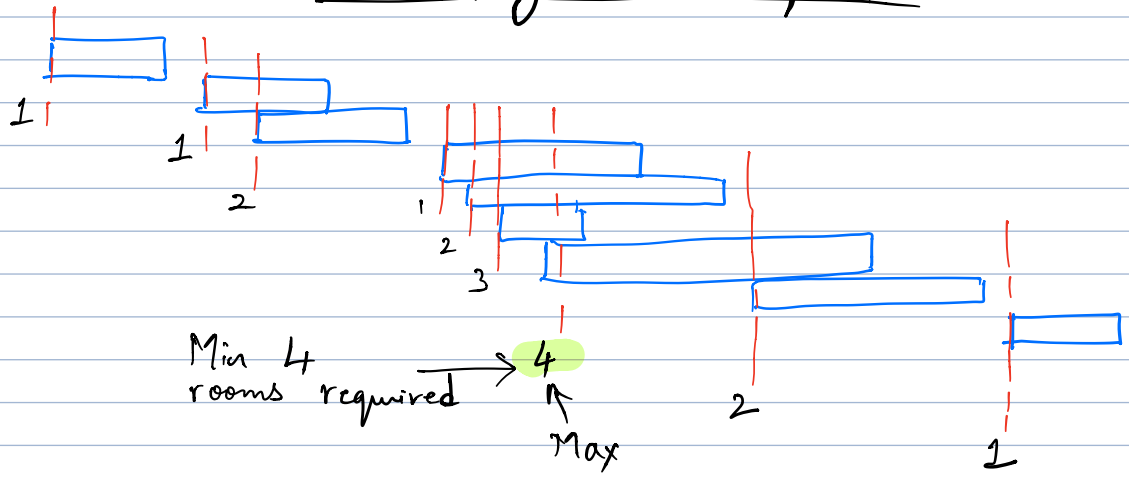
If (b.start < a.end) → Intervals are merged

↳ Merged interval

↳ start = a.start

end = max(a.end, b.end)

Minimum Number of rooms required

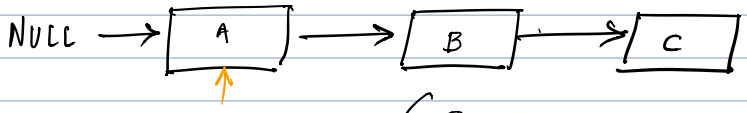


When an events starts → check how many events are still running

↳ Max of this number = Min rooms required

LINKED LIST

REVERSE A LINKED LIST



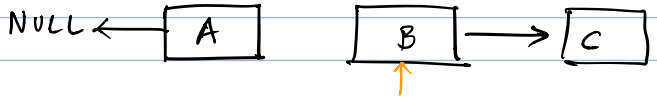
INIT

PREV = NULL
CURRENT = A

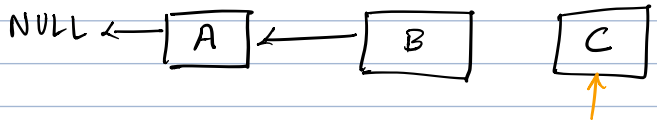
REVERSE

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

After iteration 1

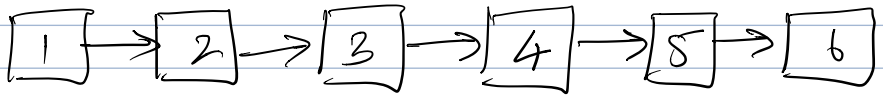


After iteration 2



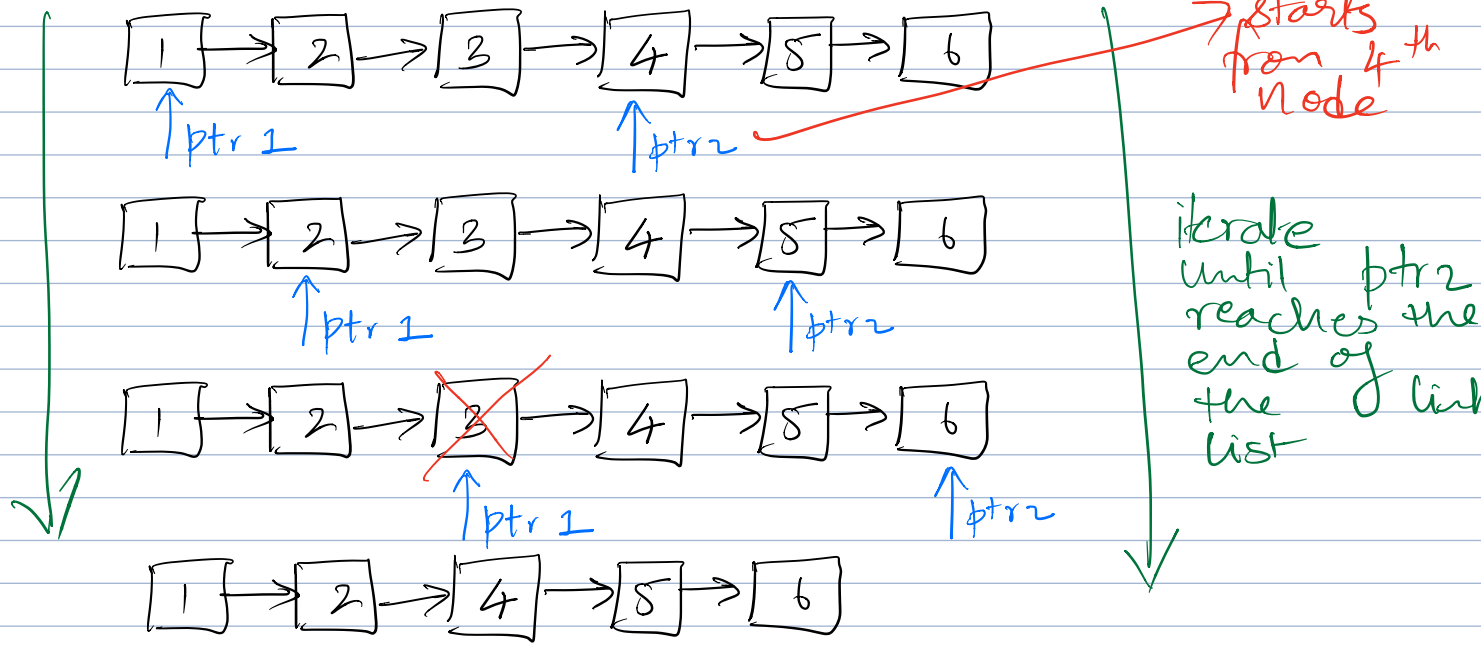
Repeat the reverse function until the current == NULL

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

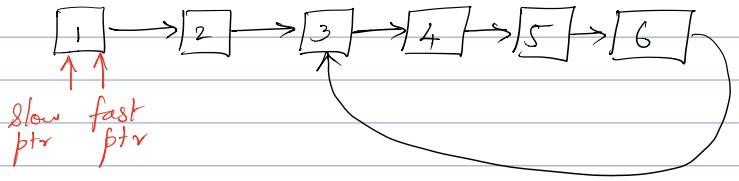


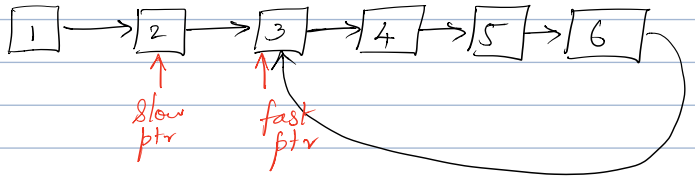
Let n = 4

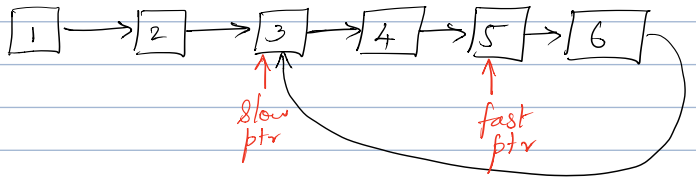
Two pointer approach



Cycle in a Linked list



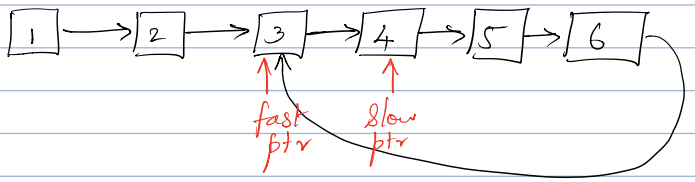


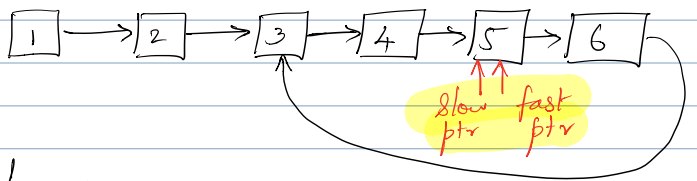


Slow pointer moves one step

fast pointer moves two steps

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





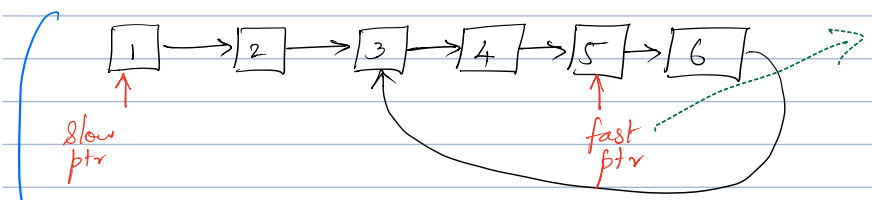
Loop detected !!

Length of cycle

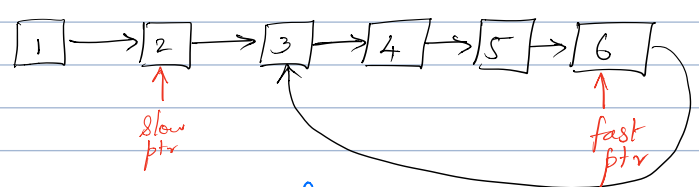
To determine the length of the cycle

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

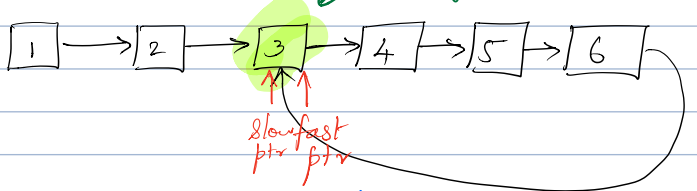
Start of cycle



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



Start of loop detected



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

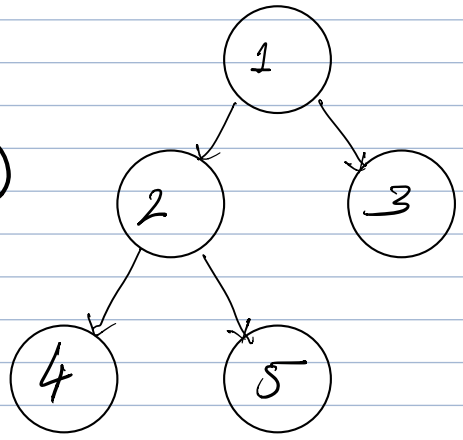
Binary Tree

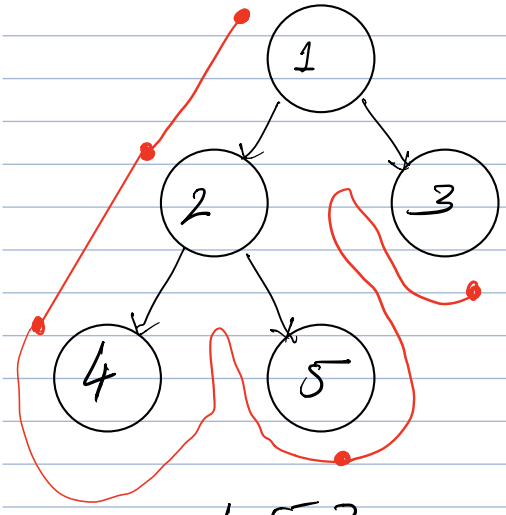
DFS (Stack for implementation)

Inorder : Left Root Right

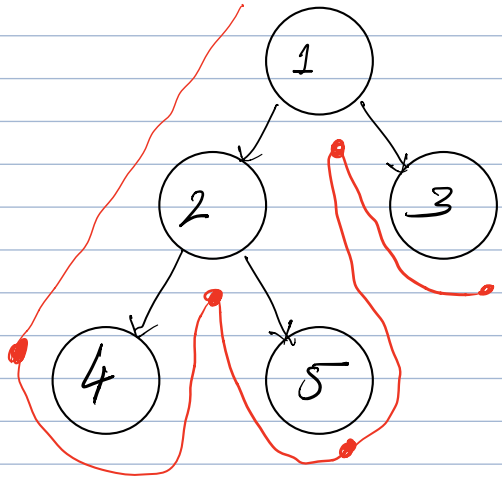
Preorder : Root Left Right

Postorder : Left Right Root

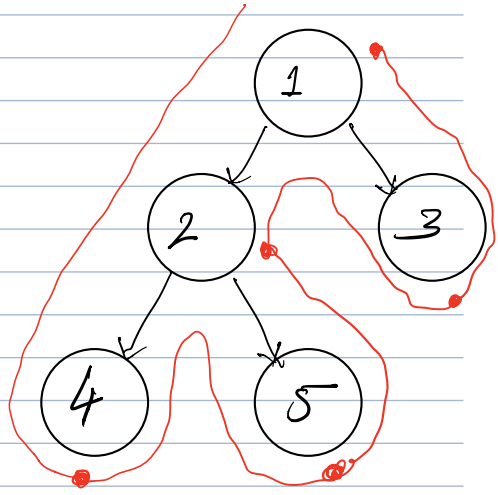




1 2 4 5 3
Preorder (DFS)

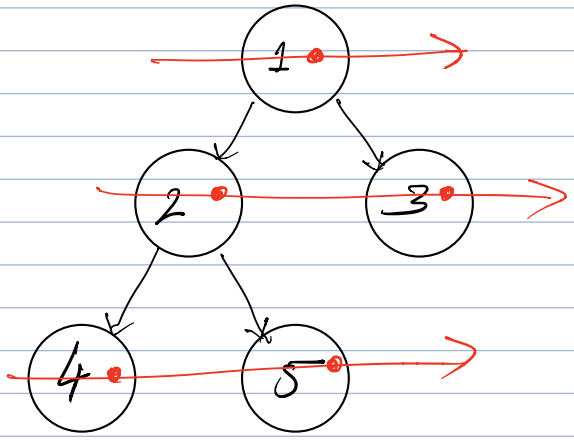


4 2 5 1 3
Inorder (DFS)



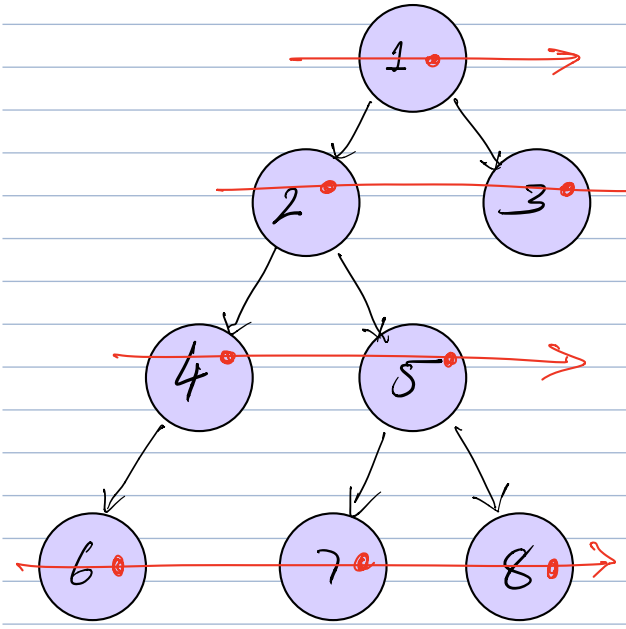
4 5 2 3 1
Post order (DFS)

BFS (Queue for implementation)

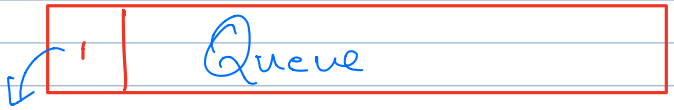


1 2 3 4 5
BFS

BFS - Iterative Soln



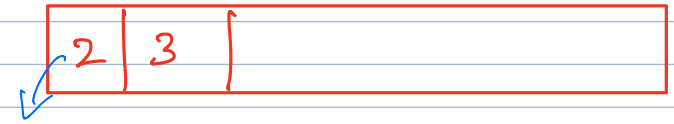
Push (1) to Queue



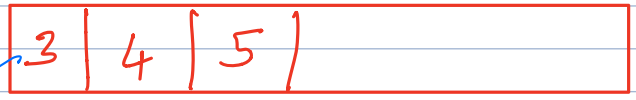
Pop 1

result = [1]

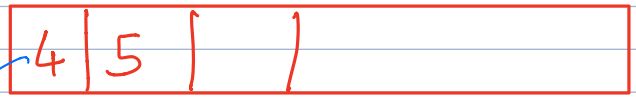
Push (2) and (3) (children of 1)



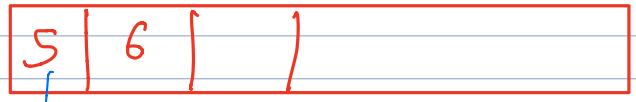
result = [1, 2]



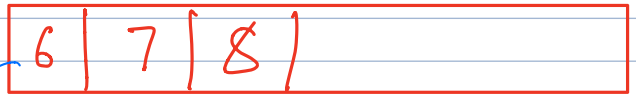
result = [1, 2, 3]



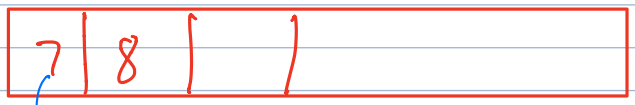
result = [1, 2, 3, 4]



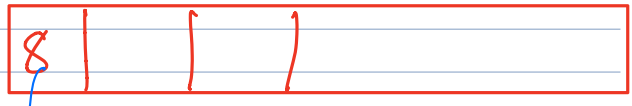
result = [1, 2, 3, 4, 5]



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

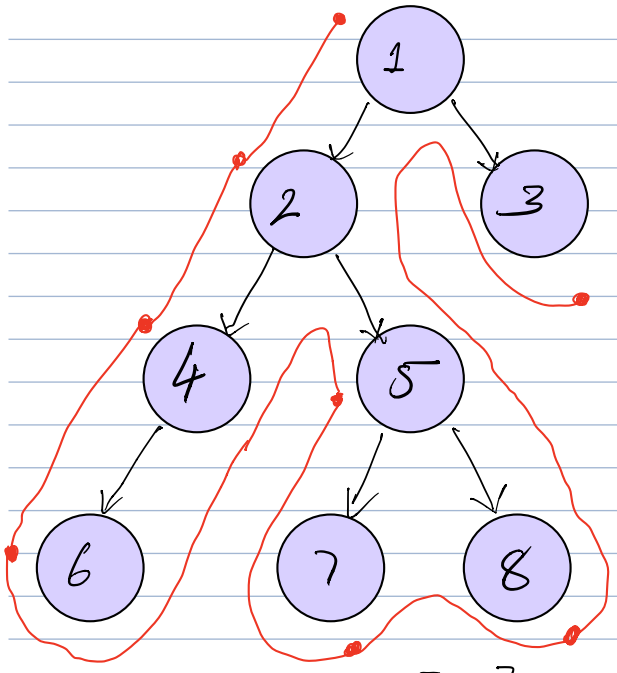


result = [1, 2, 3, 4, 5, 6, 7]



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

DFS (Preorder Traversal)



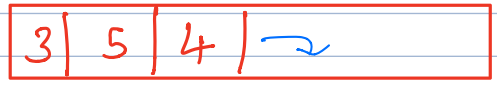
Push 1

result = [ ]

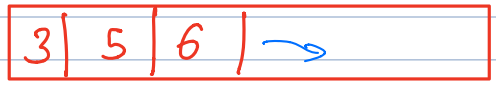
Pop 1, add to result and push the children of 1 to the stack

result = [1]

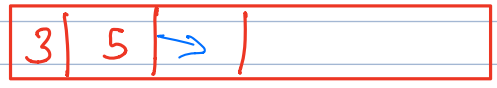
result = [1, 2]



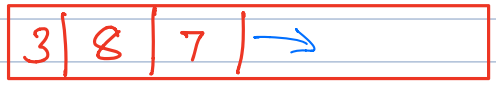
result = [1, 2, 4]



result = [1, 2, 4, 6]



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

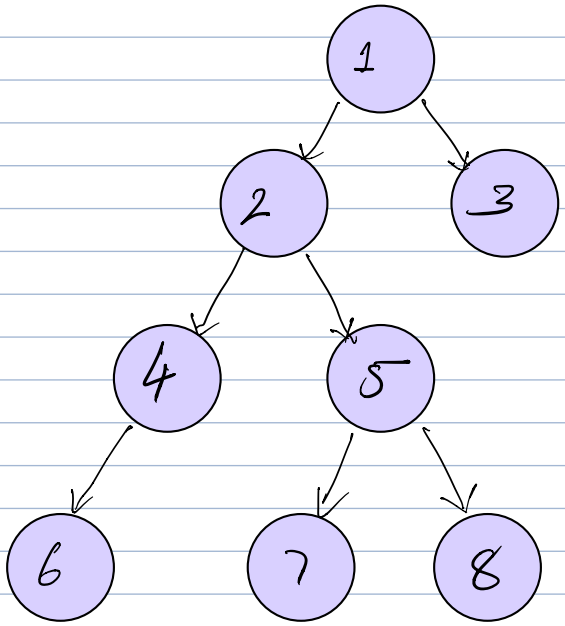


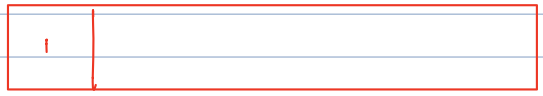
result = [1, 2, 4, 6, 5, 7]

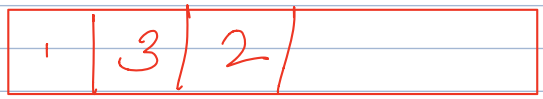
result = [1, 2, 4, 6, 5, 7, 8]

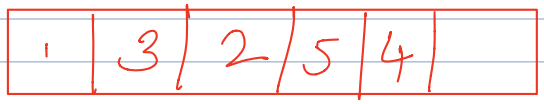
result = [1, 2, 4, 6, 5, 7, 8, 3]

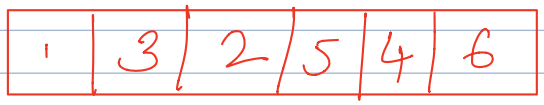
DFS











Two Heaps

Find Median

5, 4, (3), 2, 1

insert 5

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

(5)

Median = 5

Median = top of larger heap

insert 4

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

(5) — (4)

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

(4) | (5)

Median = (4+5)/2

Median = Sum of tops/2

insert 3

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

(4) — (3) | (5)

Median = 4

insert 2

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

(4) — (3) — (2) | (5)

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

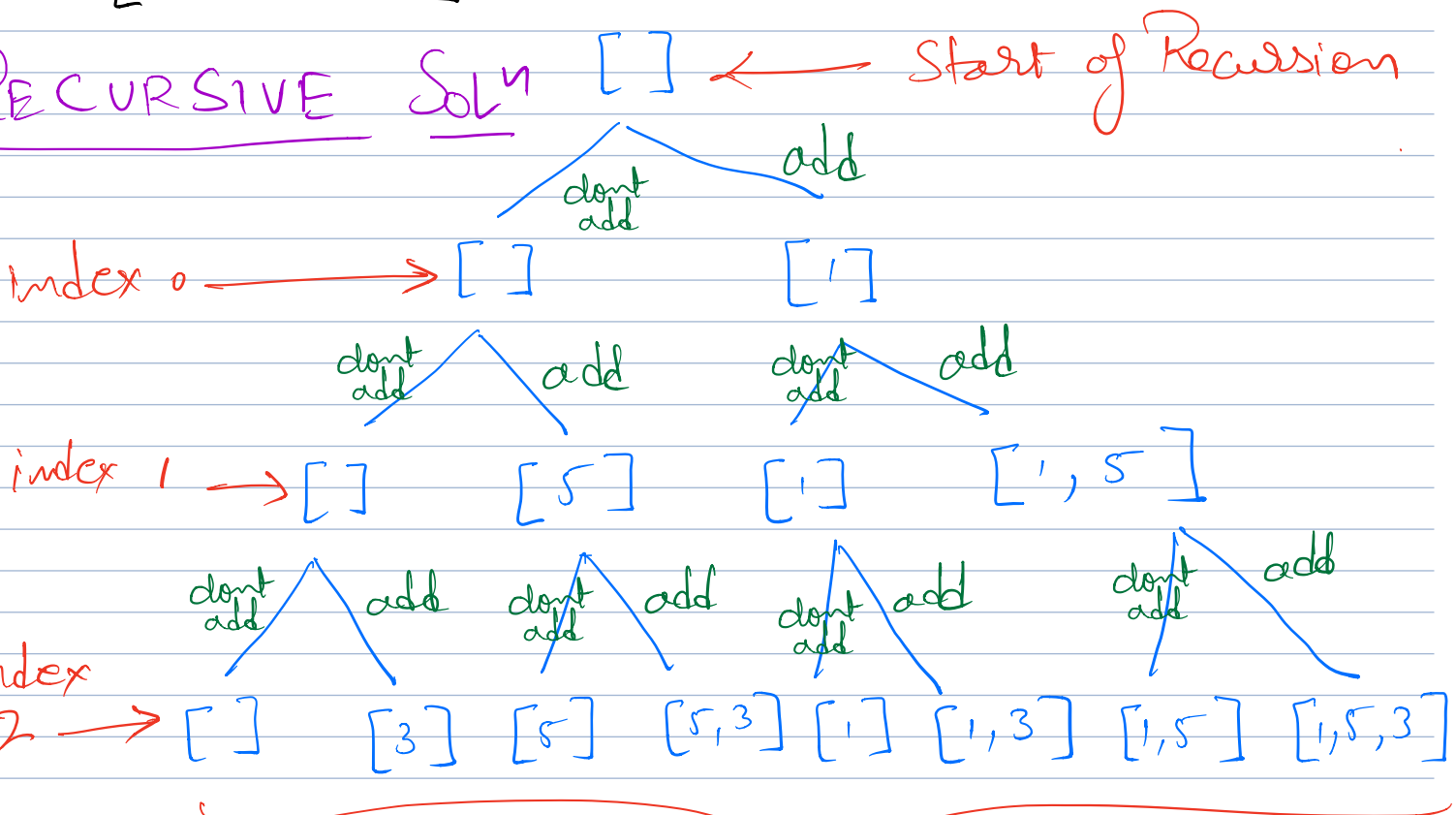
(3) — (2) | (4) — (5)

Median = (3+4)/2

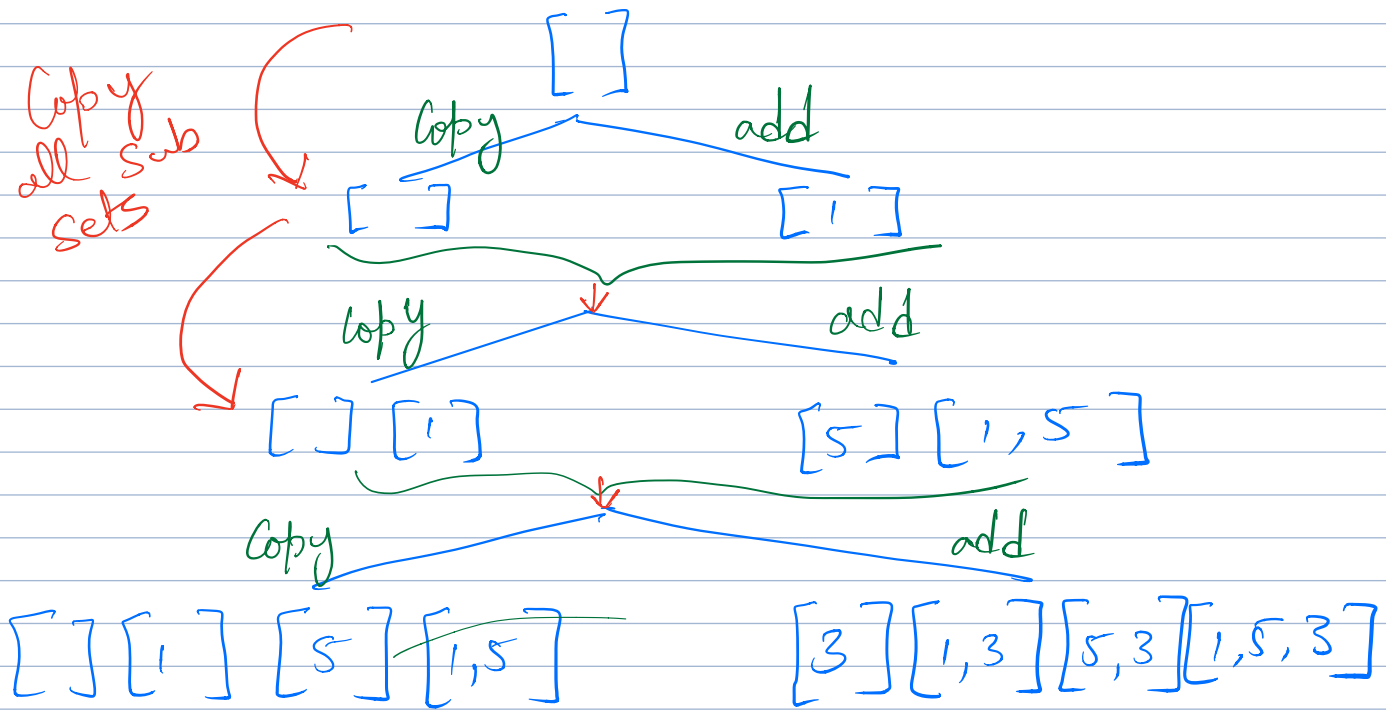
SUBSETS

Ex: [1, 5, 3]

RECURSIVE SOL^{N}



ITERATIVE SOL^{N}

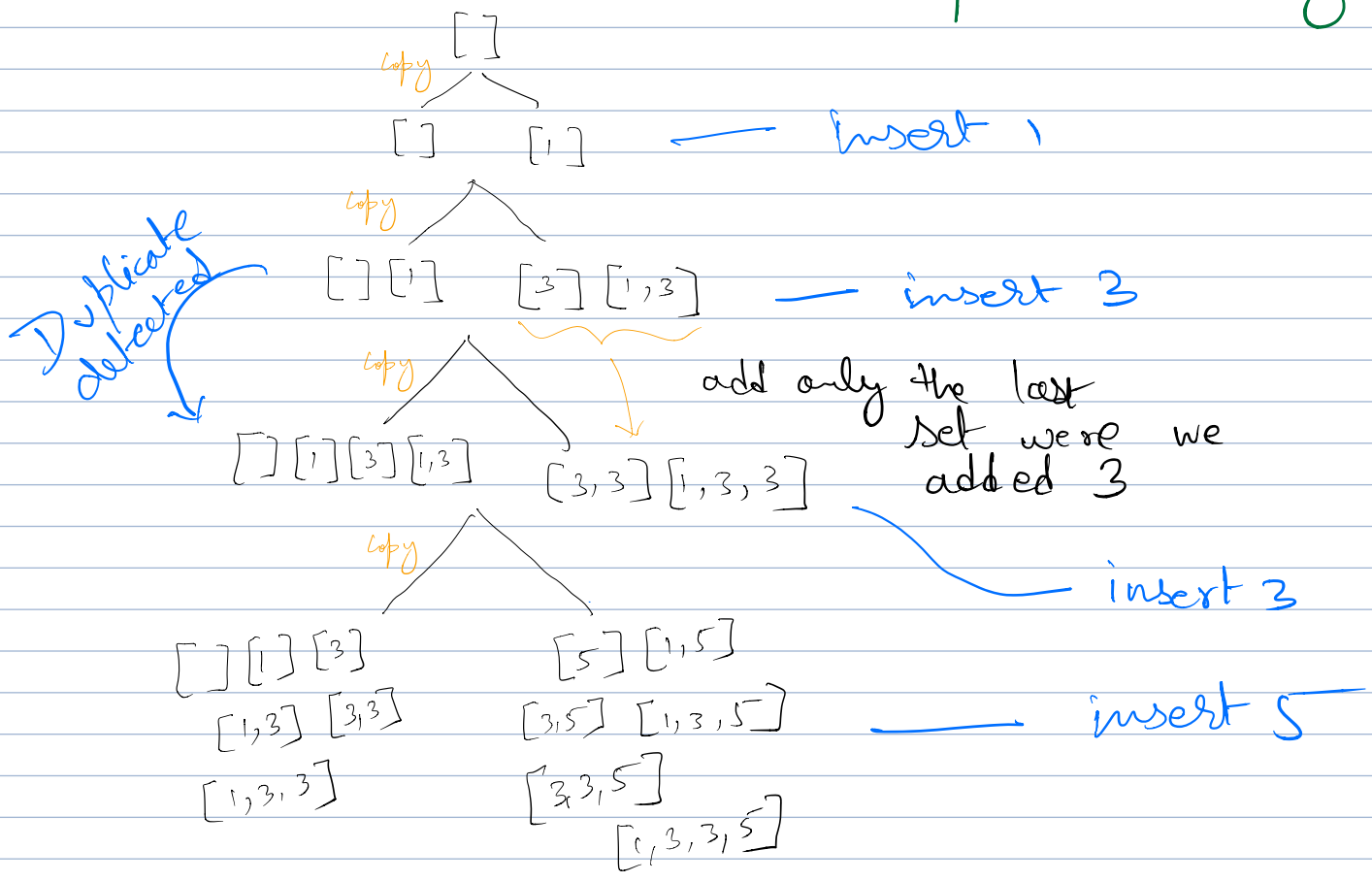


SubSets (with Duplicates)

Ex: [1, 3, 3, 5]

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

iterative approach



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]

insert 3 in all positions in each of the sets

final Result

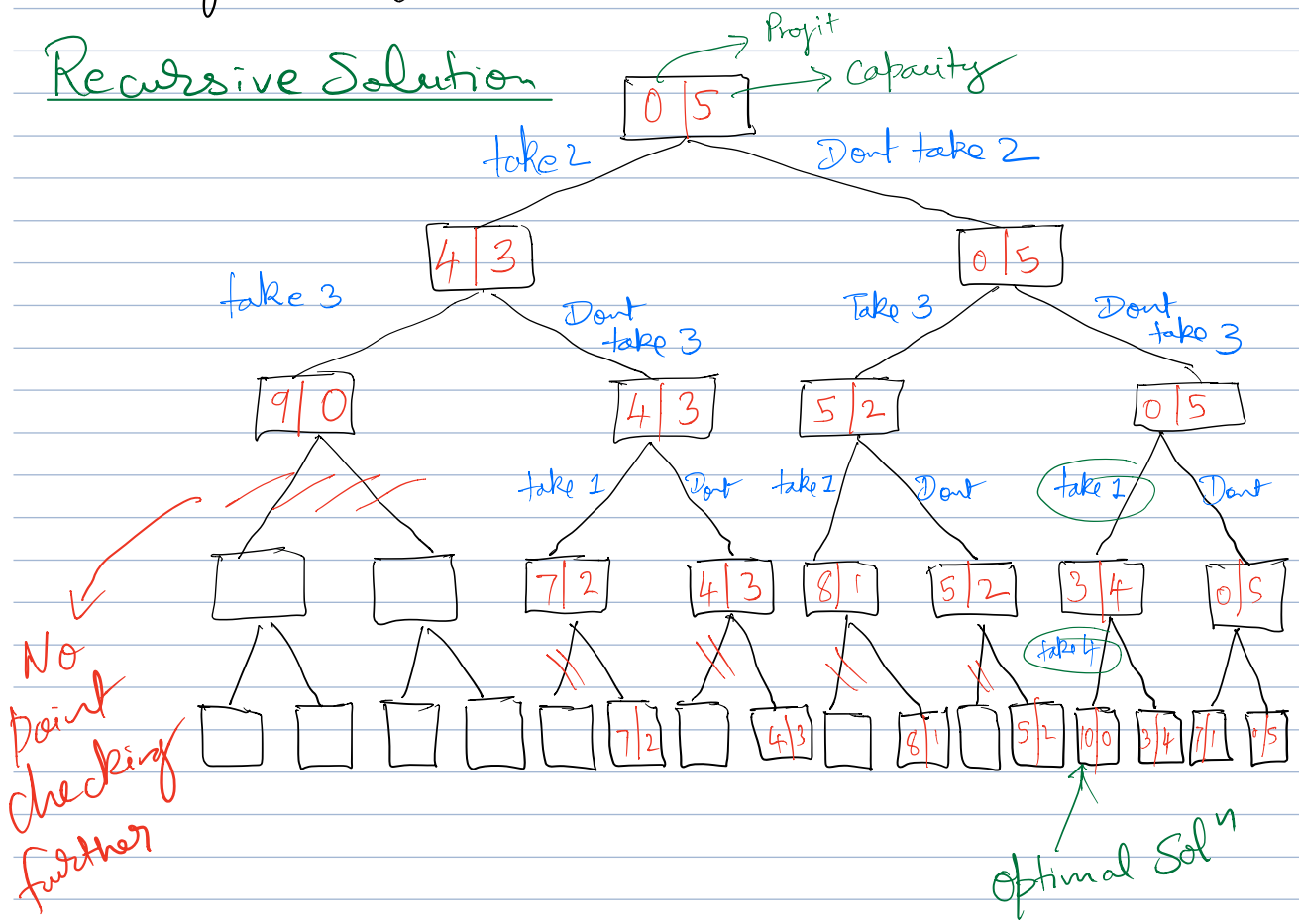
0/1 Knapsack

Weights : {2, 3, 1, 4}

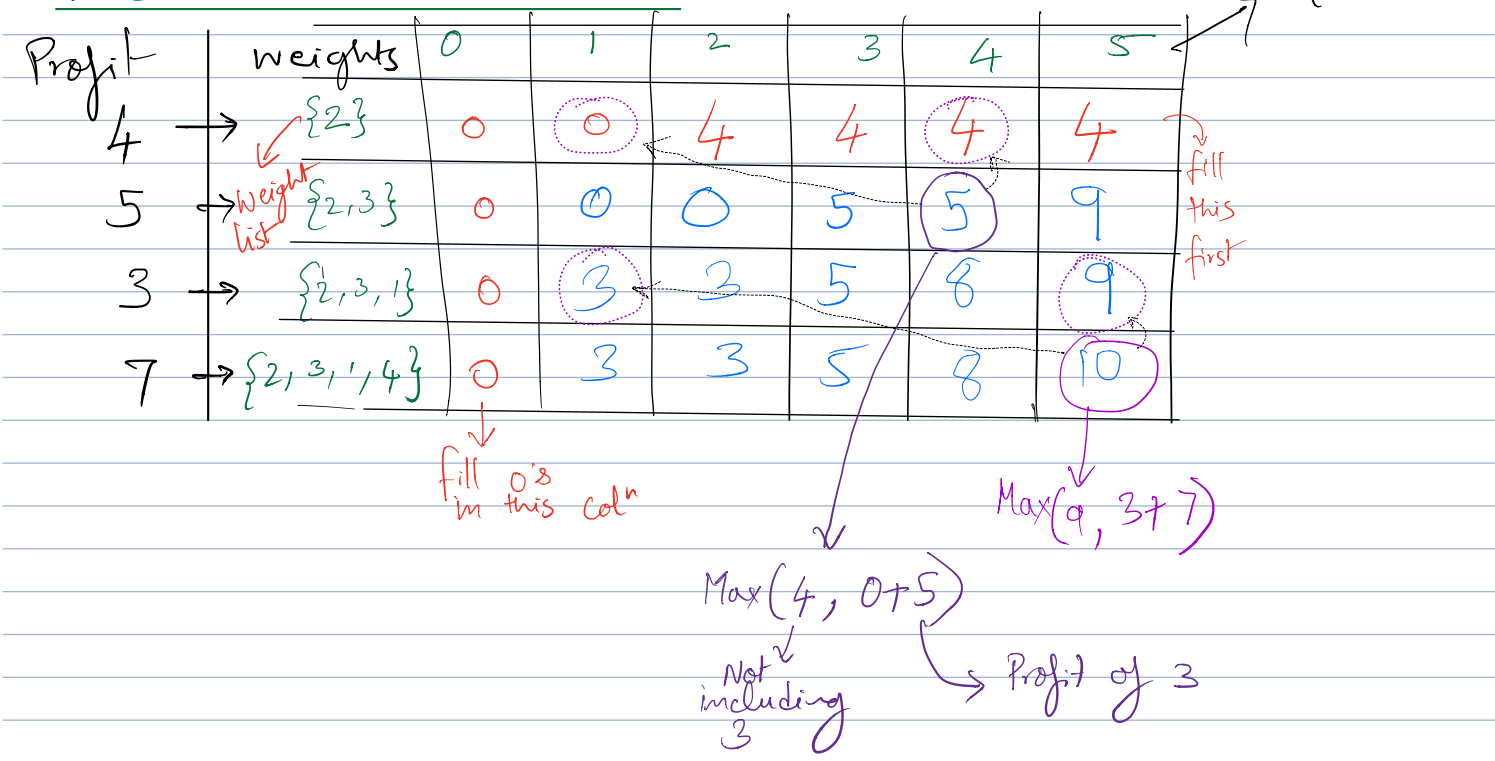
Profits : {4, 5, 3, 7}

Capacity : 5

Recursive Solution



Tabulation Method



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 knapsack ↳ Simulates multiple takes.

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

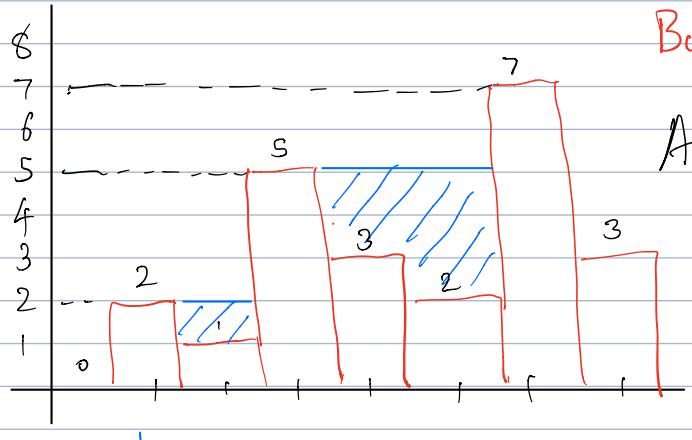
24	12	4	1

← start right

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}

Left Max (including itself)

Right Max (including itself)

Water height + building height

0	2	1	5	3	2	7	3

min (2, 7) = 2

$W_i = 2 - 2 = 0$

Water on building 1

0	2	1	5	3	2	7	3

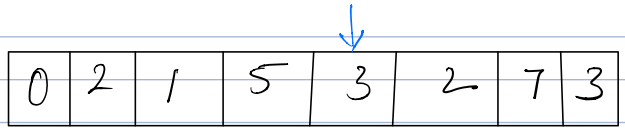
min (2, 7) = 2

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

0	2	1	5	3	2	7	3

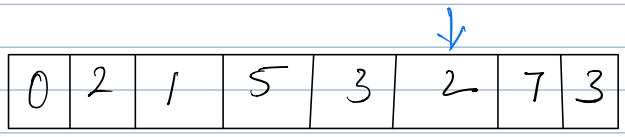
min (5, 7) = 5

$W_i = 5 - 5 = 0$



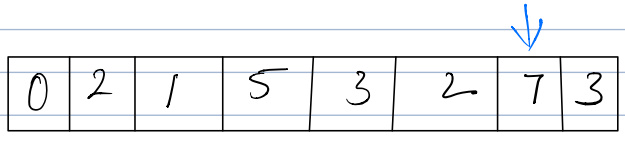
$\min(5,7) = 5$

$W_i = 5 - 3 = 2$



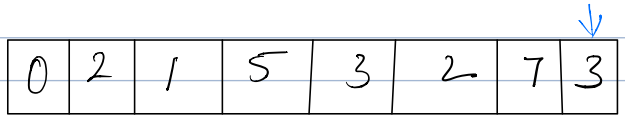
$\min(5,7) = 5$

$W_i = 5 - 2 = 3$



$\min(7,7) = 7$

$W_i = 7 - 7 = 0$



$\min(7,3) = 3$

$W_i = 3 - 3 = 0$

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

→ Maximal Square

1	0	1	0	0
1	0	1	1	1
1	1	1	1	1
1	0	0	1	0

Reusables