Advance Interview Problems

~~~~~~~~

Agenda

~~~~~~~~

1. Partition to K Equal Sum Subsets
2. Flatten Binary Tree to Linked List
3. Insert Delete Get Random
4. Best Time to Buy and Sell Stocks III

Hello Everyone
Very Special Good Evening
to all of you 😊😊😊
We will start session
from 9:06 PM

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Partition to K Equal Sum Subsets

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Given an integer array nums and an integer k, return true if it is possible to divide this array into k non-empty subsets whose sums are all equal.

* All elements are +ve

nums:

4	3	2	3	5	2	1
0	1	2	3	4	5	6

, K = 4

#1. sum of entire array would be divisible by K.

Sum = 20

sum % K ⇒ 20 % 4 = 0

o/p: [4,1], [5], [2,3], [2,3] o/p: true

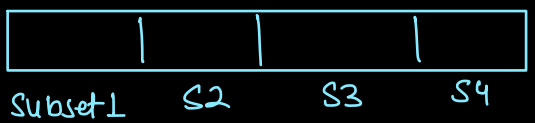
NOTE: Generation of all subset will not give you correct answer.

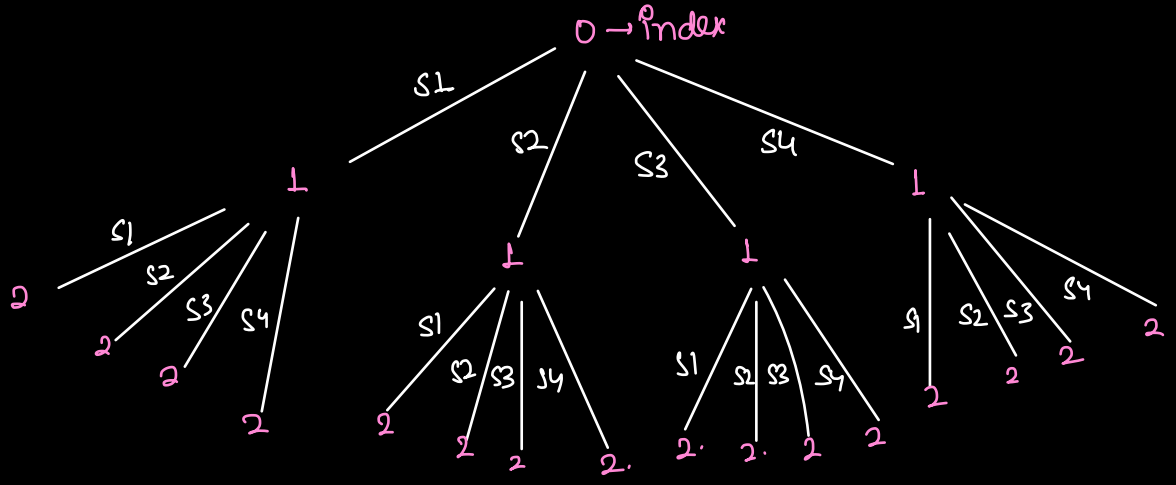
Why? → Subset → (4,1), (2,2,1)

here 1 is part of both the subset which is not correct.

Brute force!

K=4 , make 4 subset





T.C. $O(K^n)$

Optimise Approach

nums: | 4 | 3 | 2 | 3 | 5 | 2 | 1 | , K=4

0	1	2	3	4	5	6

sum of array ⇒ "S"

Check the divisibility of sum from 'K'

→ if(sum % K != 0) return false

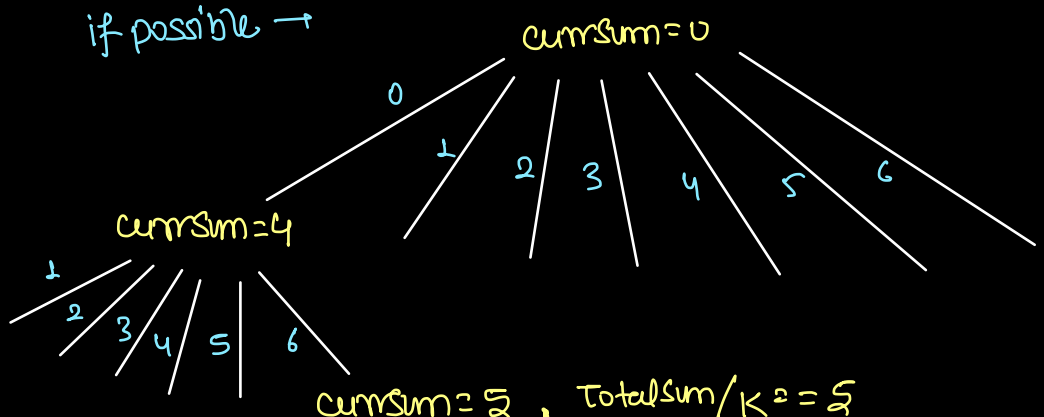
Given:

* K
* totalSum
* array
* vis

vis | T | F | F | F | F | F | T |

0	1	2	3	4	5	6

if possible →



currSum=5 , TotalSum/K==5

vis | T | F | F | F | F | F | T |

0	1	2	3	4	5	6

* one subset is prepared
* Now from given array with some visited array we have generate "K-1"

pseudocode:

```
sum=0;
for(int ele : nums){
    sum += ele;
}
if( sum % k != 0) return false;
boolean[] vis = new boolean[n];        for all vis[i] = false,
return canPartition( nums[], k, 0, 0, sum / k, vis);
                                  index   currSum
```

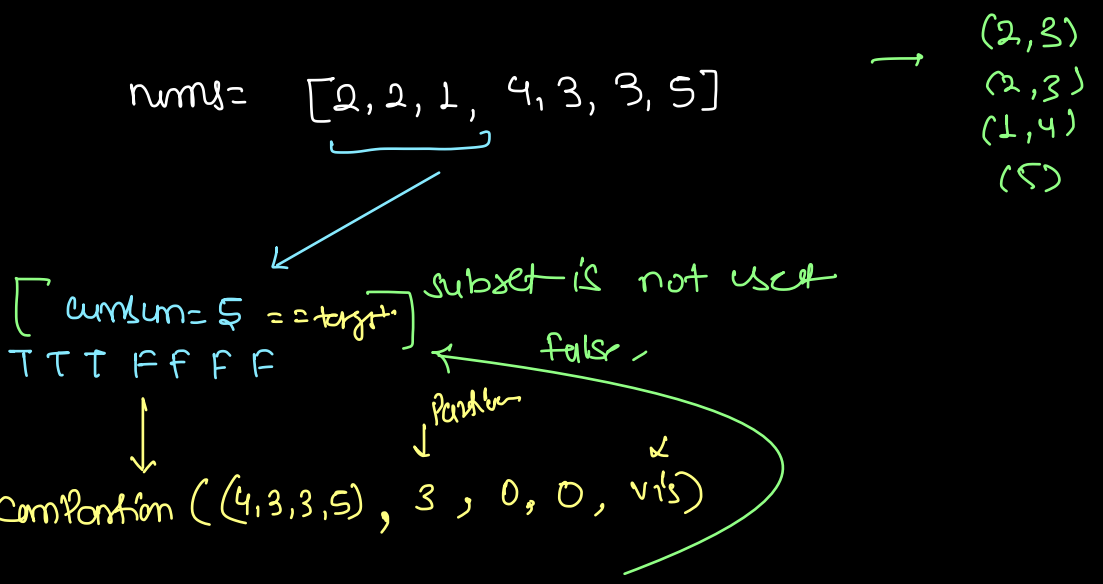
```
boolean canPartition ( nums[], k, indx, currSum, targetSum, vis[]) {
    if(k==0) { return true; }

    if( currSum == targetSum) {
        return canPartition (nums, k-1, 0, 0, targetSum, vis);
    }
    for(int i= indx; i<n; i++){
        if(vis[i] == false && currSum + nums[i] <= targetSum) {
            vis[i]= true;
            boolean res= canPartition(nums, k, i+1,
                                      currSum+ nums[i], targetSum, vis);
            if(res==true) {
                return true;
            }
            vis[i] = false;
        }
    }
    return false;
}
```

T.C: $O(2^n * k)$

S.C: (Todo)

Dry Run:

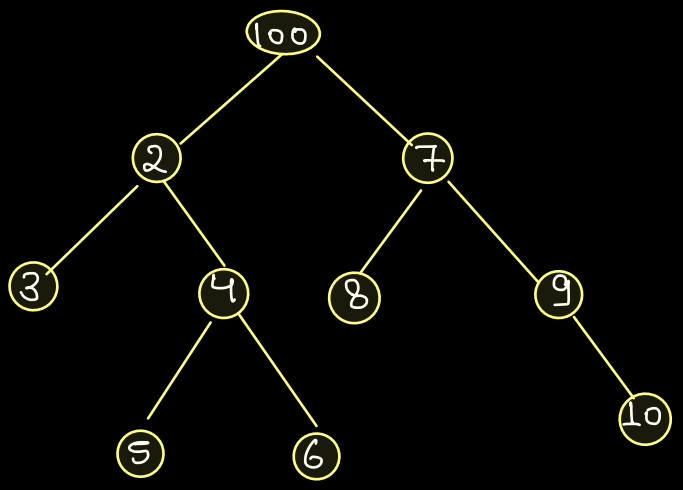


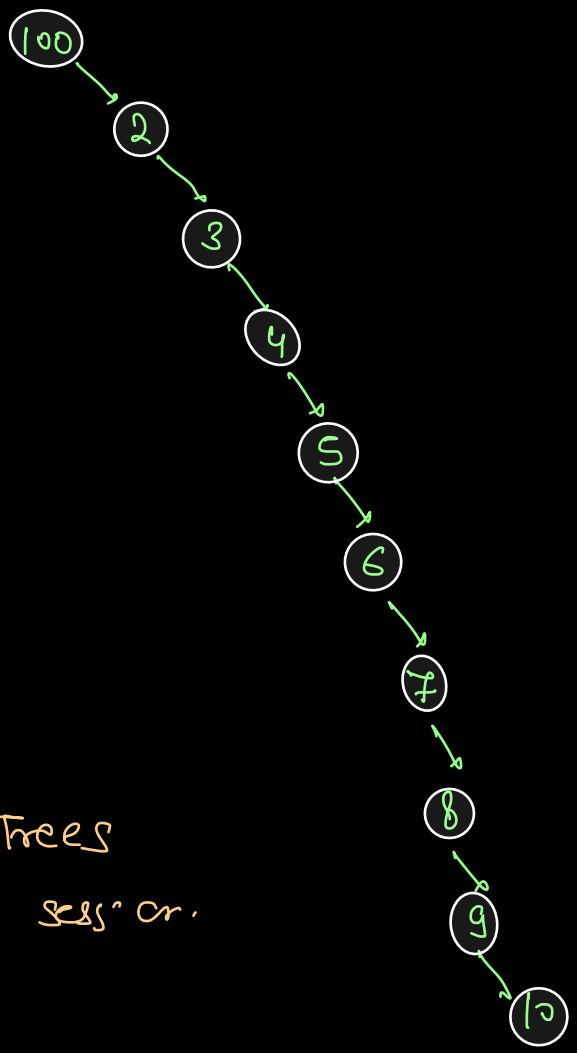
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Flatten Binary Tree to Linked List

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Given the root of a binary tree, the task is to flatten the tree into a "linked list" using the same TreeNode class. The flattened "linked list" should follow the same order as a pre-order traversal of the binary tree.





Already Discussed:

for more Reference
visit : Trie and Trees
DSA 4.2 sessions.

Insert Delete Get Random

Implement the RandomizedSet class:

`RandomizedSet()` : Initializes the RandomizedSet object. → Constructor
`bool insert(int val)` : Inserts an item val into the set if not present. Returns true if the item was not present, false otherwise.
`bool remove(int val)` : Removes an item val from the set if present. Returns true if the item was present, false otherwise.
`int getRandom()` : Returns a random element from the current set of elements (it's guaranteed that at least one element exists when this method is called). Each element must have the same probability of being returned.

You must implement the functions of the class such that each function works in average O(1) time complexity.

operations:

insert(5) ✓ → True
insert(9) ✓ → True
insert(2) ✓ → True
insert(15) ✓ → True
remove(8) → false
insert(2) ✓ → false
insert(3) ✓ → True
getRandom() →
remove(9) ✓ → True

Randomised Set

5, ~~9~~, 2, 15, 3

How to solve:

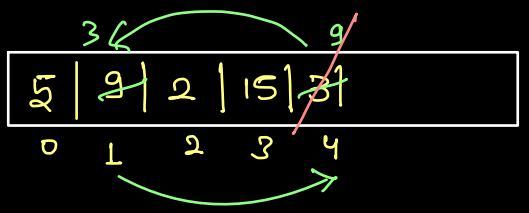
operations:

insert(5) ✓ true
insert(9) ✓ "
insert(2) ✓ "
insert(15) ✓ "
remove(8) ✓ false
insert(2) ✓ false
insert(3) ✓ true
getRandom() → indexing (0,4)
remove(9) ✓ ↳ swap it with last element of list.

Randomised Set

5 → 0
~~9 → 1~~ 4
2 → 2
15 → 3
3 → ~~4~~ 1

Hashmap:
ele : index in list



: list

Random random = new Random
list.get(random.nextInt(list.size()));

```
class RandomisedSet {
    HashMap<Int, Int> map;
    ArrayList<Int> list;

    public RandomisedSet() {
        map = new HashMap<>();
        List = new ArrayList<>();
    }

    public boolean insert(int ele) {
        if(map.containsKey(ele) == true){
            return false;
        }
        list.add(ele); ⟶ addLast
        map.put(ele, list.size -1);
        return true;
    }

    int getRandom() {
        Random random = new Random
        list.get(random.nextInt(list.size()));
    }

    boolean remove(ele) {
        if(map.containsKey(ele) == false){
            return false;
        }
        indx = map.get(ele);
        swap(list('indx') with list(list.size -1))
             val1                  val2
        map.put(list['indx'], 'indx];
        list.remove(list.size() -1);
        map.remove(ele);
        return true;
    }
}
```

T.C: O(1) for every function

10:12 – 10:25 PM Break.

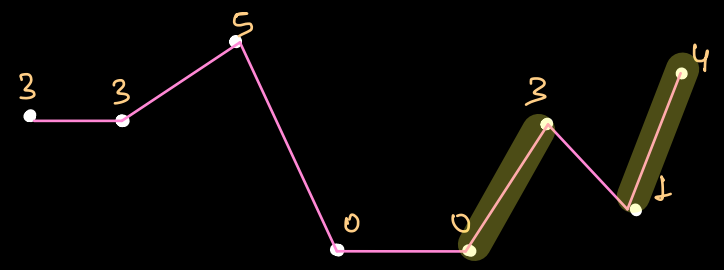
Best Time to Buy and Sell Stocks III

You are given an array A, where the ith element is the price of a given stock on day i. Design an algorithm to find the maximum profit you can achieve by completing at most 2 transactions. Note that you cannot engage in multiple transactions at the same time, eaning you must sell the stock before buying again.

1 transaction → (Buy − Sell) , 2 transaction → [Buy - Sell] [Buy - Sell]

arr:

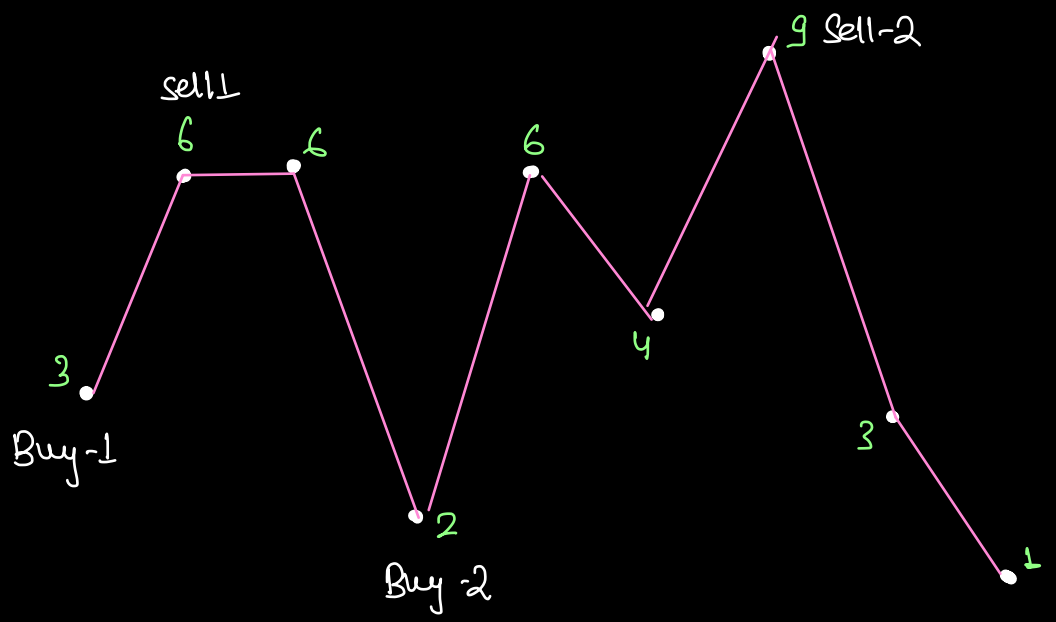
3	3	5	0	0	3	1	4
0	1	2	3	4	5	6	7



profit = (3-0) + (4-1)
= 3 + 3 = 6

arr:

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



total profit = (Sell1 - Buy1) + (Sell2 - Buy2)
= (6-3) + (9-2)
= 3 + 7 = 10

Bruteforce:

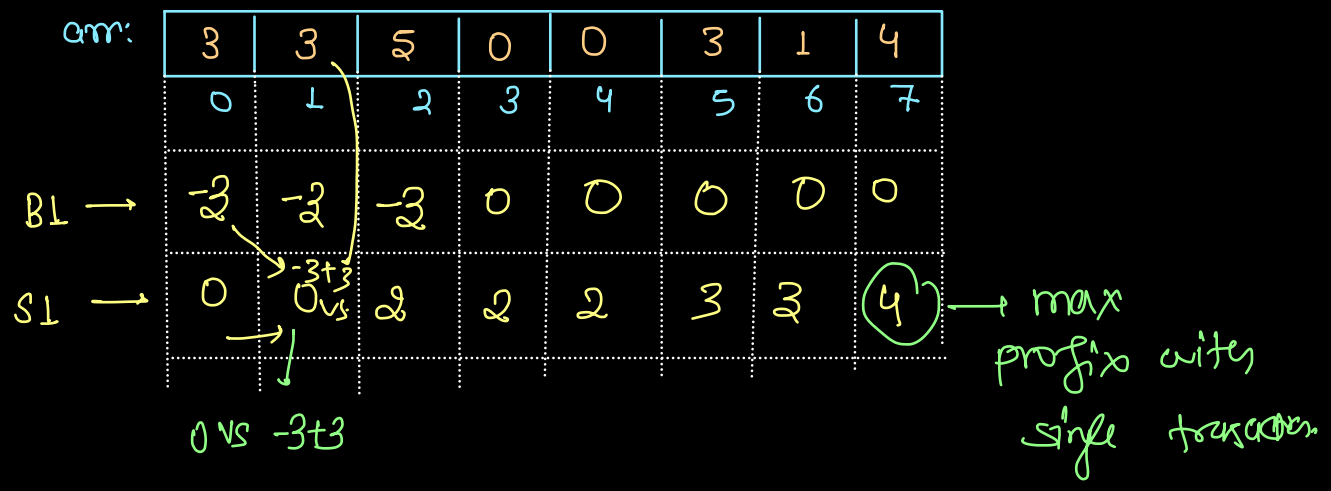
Consider all possible Quadruplets

T.C. $O(n^4)$

Optimise Approach:

```
b1 = max(b1, -arr[i]);
s1 = max(s1, arr[i] + b1);
```

first transaction.




```
firstbuy = -arr[0];
firstsell = 0;
for(int i=1; i<n; i++){
        b1 = firstbuy;
        s1 = firstsell;
        first_buy = max(b1, -arr[i]);
        firstsell = max(s1, arr[i] + b1);
}
return firstsell;
```

Two transaction:

arr:	3	3	5	0	0	3	1	4
	0	1	2	3	4	5	6	7
B1 →	-3	-2	-	-	-	-	-	-
S1 →	0	0	-	-	-	-	-	-
B2 →	-∞	-2	-	-	-	-	-	-
S2 →	0	0	-	-	-	-	-	-

B1 = -arr[0]
S1 = 0
B2 = -∞
S2 = 0

firstBuy = -arr[0], firstSell = 0, secondBuy = -∞, secondSell = 0

```
for(int i=1; i<n; i++){
    b1 = firstBuy;
    s1 = firstSell;
    b2 = secondBuy;
    s2 = secondSell;

    firstBuy = max(b1, -arr[i]);
    firstSell = max(s1, arr[i]+b1);
    secondBuy = max(b2, s1-arr[i]);
    secondSell = max(s2, b2+arr[i]);
}
return max(firstSell, secondSell);
```

T.C : O(n)
S.C : O(1)