Recap : DP → Why DP ? Recursion + Memoization → DP sol^n

↳ Problem 1 : Knapsack → 1. Why greedy doesn't work ?

subset vs. subarray

contiguous

↓ DP. ✓ prefix sum. ✓

2. Recurrence Relation

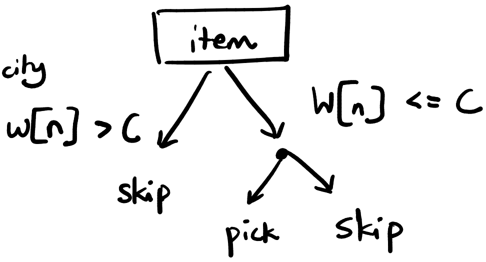
n = 3	23	22	4
C = 12	11	10	2

pe kg → picked

most valuable

n items

C capacity



max price of bag

↓

int memo[n+1][C+1] = {-1}

```
code :  int knapsack ( int w[] int p[], int n, int C) :
            if n == -1 or C <= 0 : return 0
            int ans = 0 ;        →  if memo[n][c] != -1 : return memo[n][c]
            if w[n] > C :
        ans = ~~return~~ knapsack (w, p, n-1, C)
            else :
        ans = ~~return~~ max { knapsack (w, p, n-1, C)
                             { knapsack (w, p, n-1, C-w[n]) + p[n] }
        memo[n][c] = ans
        return memo[n][c]
```

T.C. ~ $O(2^n)$

exponential

↓ Memoization

T.C. ~ $O(n.C)$

polynomial

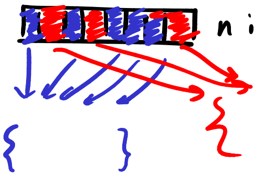
How TC ↓ ? → Because I am just finding the max price possible but not the actual items that make up that price.

Any question → item for each one.

↙ ↘

pick skip

KNAPSACK TEMPLATE.



n items

{ } Form a subset | pick.

{ } excluded set | skip

Exhaustive search.

TC $O(2^n)$ ↓ $O(n.sum)$

↓

Variations :

① Subset sum problem arr = {1, 2, 4, 5, 6, 10, 11} target sum = 9

(↑ & sum) ⇒ Is there a subset that sums to the target sum?

o/p : Yes

∵ {4, 5}

{1, 2, 6}

int memo[n][sum] = {-1}

bool ans = F

DP.

```
sol^n :  bool subset_sum ( int arr[], int n, int sum) :
           if sum == 0 : return True       if memo[n][sum] != -1 :
                                              return [n][sum]
         * if n == -1 : Return False ←
           if arr[n] > sum : ans = ~~return~~ ss (arr, n-1, sum)
           else : ans = ~~Return~~ ss (arr, n-1, sum) || ss (arr, n-1, sum - arr[n])
           mem [n][sum] = ans      return ans ;
```

② Equal sum partition [leetcode] 416.

Q. Can you split the array into 2 sets such that their sum is equal?

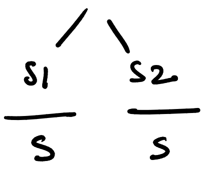
Eg. arr = $\{1, 5, \overset{3,2}{\cancel{5}}, 11\}$ $\xrightarrow[\text{subsets}]{}$ $\{1, 5, \overset{3,2}{\cancel{5}}\}$ $\{11\}$ o/p : Yes.

sum(arr) shuffled. 11 S1 11 S2

⇒ Subset sum problem → target sum → x | I need to come up.

arr → sum(arr) = 2s

↳ even.



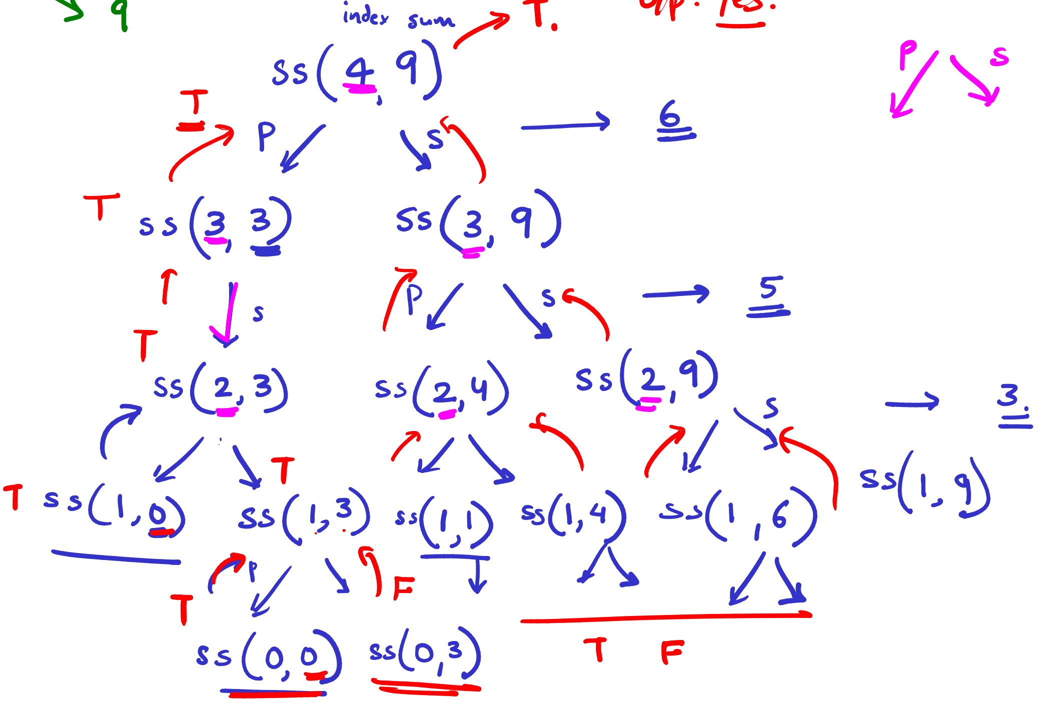
if sum(arr) % 2 != 0 : ans = No.

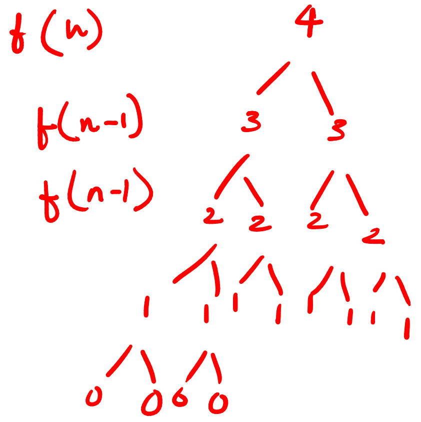
else ans = ss(arr, n, sum(arr)/2)

→ arr = $\{\overset{0}{1}, \overset{1}{3}, \overset{2}{3}, \overset{3}{5}, \overset{4}{6}\}$ ⇒ Can I split into 2 subsets?

n = 5

1. sum(arr) = 18 arr → 9, 9 target_sum = 9 o/p: Yes.





Short circuiting

↳ OR → one true → I will not recurse further! } handled by compiler!!

↳ AND → one false → not recurse further!

arr = $\{\underset{1}{1}, \underset{2}{3}, \underset{3}{4}\}$ n = 3 ss(arr, n, 4)

{1, 3} {4}.

How memo table looks like?

memo[n+1][sum+1] sum.

index n	0	1	2	3	4
0	T	F	F	F	F
1	T	T	F	F	F
2	T	T	F	T	T
3	T	T	F	T	T

1-based indexing

final row →

sum=4

V.V.Imp.

empty cell ⇒ -1.

n sum
n-1 sum

↓ skip

[1][1]

[1][4-3]

else.

↓

(n-1)[sum - arr[n]]

[1][3-3] 0

arr

Recurrence : $ss(n, sum) =$ if skip : $ss(n-1, sum) \Rightarrow$ * Go one row up in the same colⁿ & copy that value

CODE. $= $ else: $ss(n-1, sum) \;||\; ss(n-1, sum - arr[n])$

↱ represents sum

$\boxed{3}$

n: 1 2 3

$arr = [1, 3, 4]$

Go one row up & col ⇒ sum - arr[n] .

$ss(2, 4)$ → Is sum possible of 4 with initial 2 elements ?

memo[1][4-3]

↓

memo[1][1] ⇒ Tell me if it is possible to make sum 1 with the 1st element ?

↓

Yes

$arr = \{1, 3, 5, 6\}$. 4

$sum(arr) = 15$

6,1

5,3

5,3,1 6,3,1 6,5

6,5,1

Memo :

n \ sum	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
1																
2																
3																
→ 4	T	T	F	T	T	T	T	T	T	T	T	T	T	F	T	T

0 to sum(arr) → all possible sums that you can make .

③ Minimum subset sum difference .

arr → s_1 → s ; s_2 → s Equal sum partition ⇒ ?

$arr = \{1, 3, 5, 4\}$ $sum(arr) = 13$.

Equal sum partition → ×

$sum(s_1) = sum(s_2)$

$\Rightarrow sum(s_1) - sum(s_2) = 0$ ↙ may not be zero.

Q. Min.diff partition ?

$$\left| \underbrace{\{5, 1\}}_{6} - \underbrace{\{3, 4\}}_{7} \right| = 1$$

minimize → $diff = | sum(s_1) - sum(s_2) |$

$s \rightarrow sum(arr)$ $s_1 \rightarrow sum(s_1)$ $s_2 \rightarrow sum(s_2)$

$s = s_1 + s_2$

$diff = | s_1 - (s - s_1) | = | 2s_1 - s |$

minimize ↓ $diff = | s - 2s_1 |$ maximize s_1

↑ constant .

$s_1 \Leftarrow \frac{s}{2}$

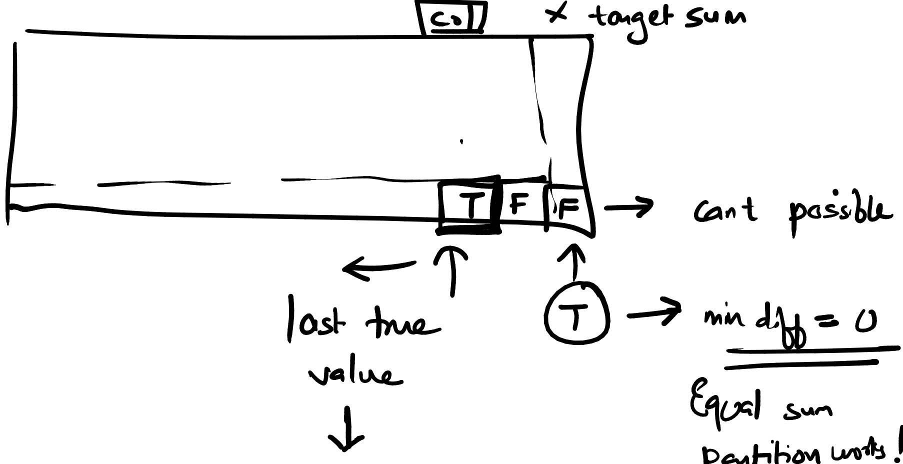
$ss(arr, n, s/2)$

$diff = |s - 2.s_1|$ ↑ max value.

diff ↓ Minimum

```
for (int i= sum ; i>=0 ; i--)
    if memo[n][i]
        max_s1 = i
        break;
```

V.Imp.



↓ The max sum I can make in set s1.

(s₁ +ve) (s₂ -ve) sum(s₁) - sum(s₂) = target

arr : [1, 1, 1, 1, 1] target = 3.

↑ give each number a sign +ve −ve

→ eventual result = 3.

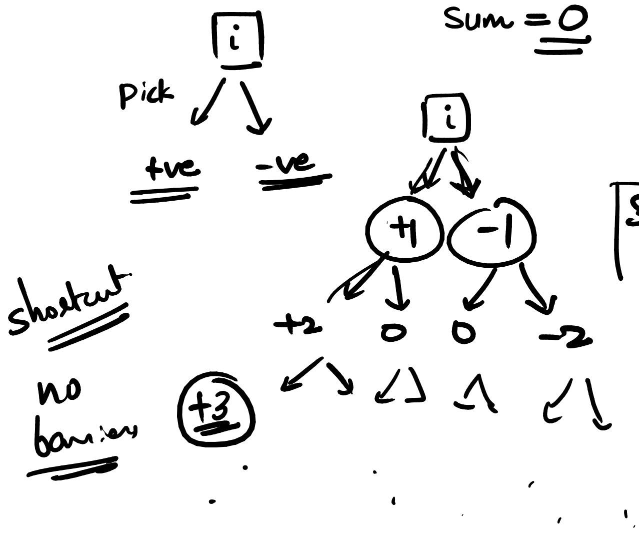
④ Target sum (leetcode) 494.

+1 +1 +1 +1 −1 = 3.

Q. Is this possible ?

Easier !! $diff = |sum(s_1) - sum(s_2)|$

No need to rely on memo table !!



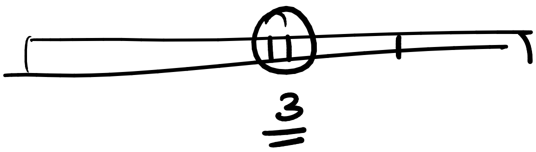
Yes/No.

$$\boxed{s_1 = \frac{s-t}{2}}$$

$$target = \Big|\underbrace{sum(arr)}_{S} - 2*s_1\Big|$$

$t = s - 2.s_1$

$\frac{5-3}{2} = 1$



ways → Combinatorial.

1+1+1+1−1
1+1+1−1+1
1+1−1+1+1
⋮
} 5

o/p : 5