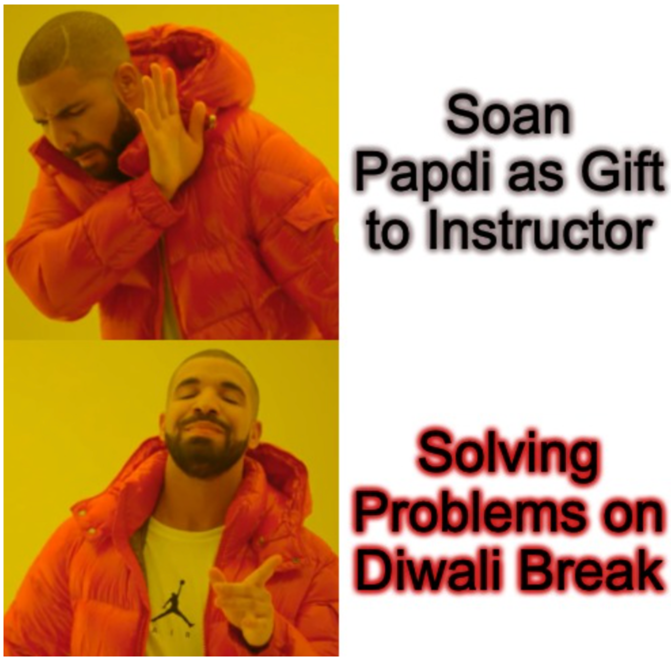


Todays Agenda

→ Rod Cutting

→ Coin Change

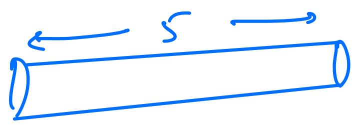
→ 0-1 KnapSack.

Rod Cutting

Given a rod of length N & an array of length N.

arr[i] → price of i-length rod.

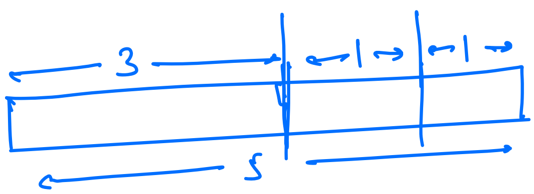
Find the **max value** that can be obtained by cutting the rod into 1 or more pieces and selling them.



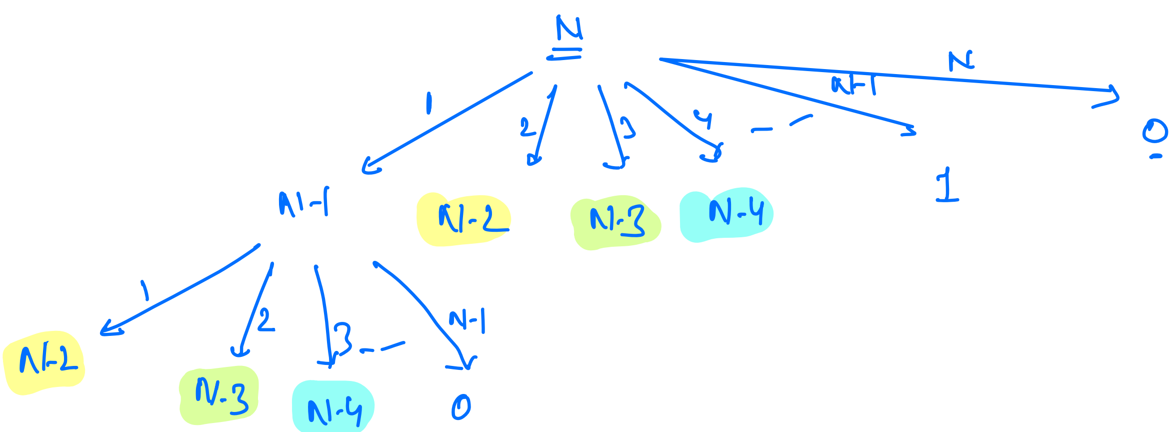
N = 5.

arr →

1	2	3	4	5
1	4	2	5	6



sold length	value
5	6
4+1	6
3+2	6
3+1+1	4
2+2+1	4+4+1 = 9
2+1+1+1	7
1+1+1+1+1	5



⇒ Similar to unbounded Knapsack.

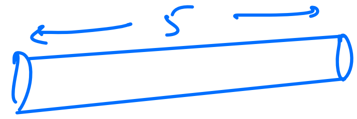
Optimal sub-structure ✓

overlapping sub-problems ✓

$N = 5.$

arr →

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

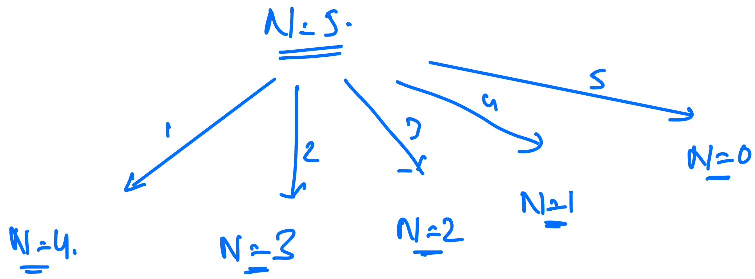


dp →

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

6,8,3,5   9,9,6,6,6

9 → ans.



dp[i] → Max value of i-length rod.

code →

```
dp(N+1) ,   ∀i, dp[i] = 0;

for( i = 1 ; i <= N; i++){
    for( cut = 1 ; cut <= i ; cut++){
        dp[i] = Max( dp[i], arr[cut] + dp[i-cut]) ;
    }
}
return dp[N];
```

T.C → $O(N^2)$
S.C → $O(N)$

Coin change

N different denominations

Total no. of ways to pay a given amount.

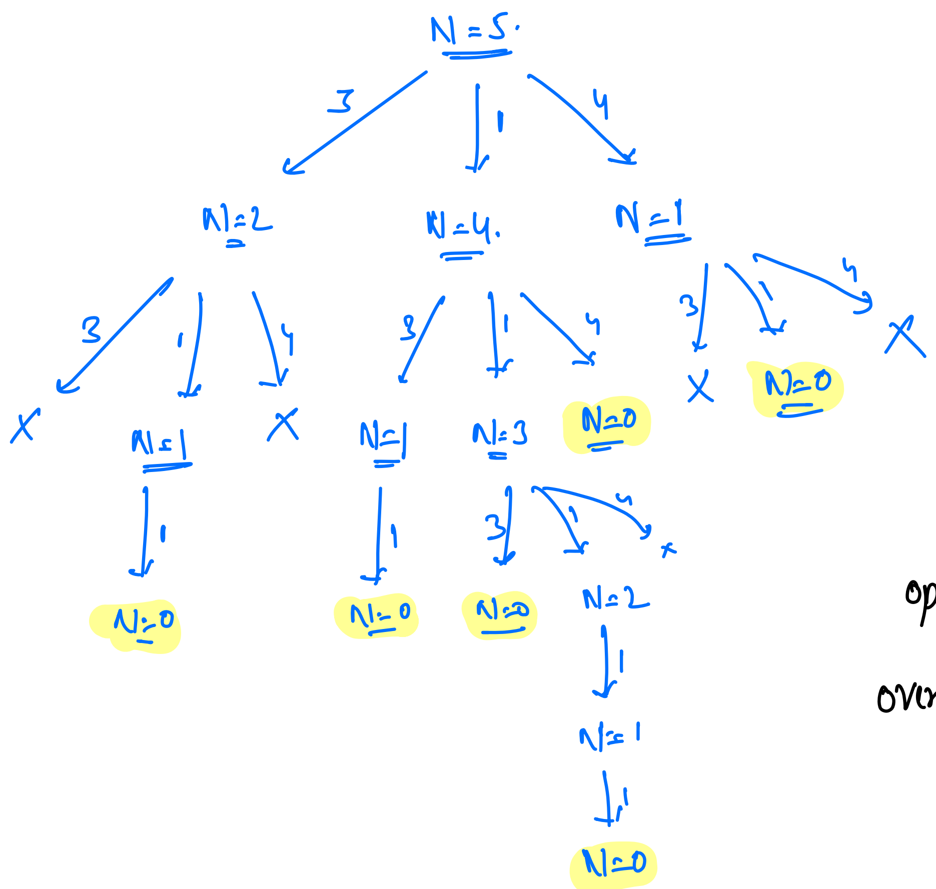
Any denomination any no. of times.

Amount = 5   denoms → [3 1 4] $(x,y) \neq (y,x)$

(1,4)   (3,1,1)   (1,1,1,1,1)

(4,1)   (1,1,3)   ans = 6.

(1,3,1)



optimal substructure ✓

overlapping sub-problems ✓

→ unbounded Knapsack.

No. of ways to pay amount 0 → 0 ✗ 1 ✓

⇓ Amount 0 can't be paid ⇓ [pay nothing]

Eg → denoms → [3 1 4] Amount = 5

dp =

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

dp[i] → Total no. of ways to pay i^{th} amount.

code →

```
dp[amount+1] , ∀i, dp[i] = 0;
dp[0] = 1;
for( i = 1; i ≤ amount; i++){
        for ( j = 0; j < denoms.length; j++){
                if( i - denoms[j] ≥ 0){
                        dp[i] += dp[i - denoms[j]];
                }
        }
}
return dp[amount];
```

T.C → O(N * amount)
S.C → O(amount)

Coin change.

N different denominations

Total no. of ways to pay a given amount.

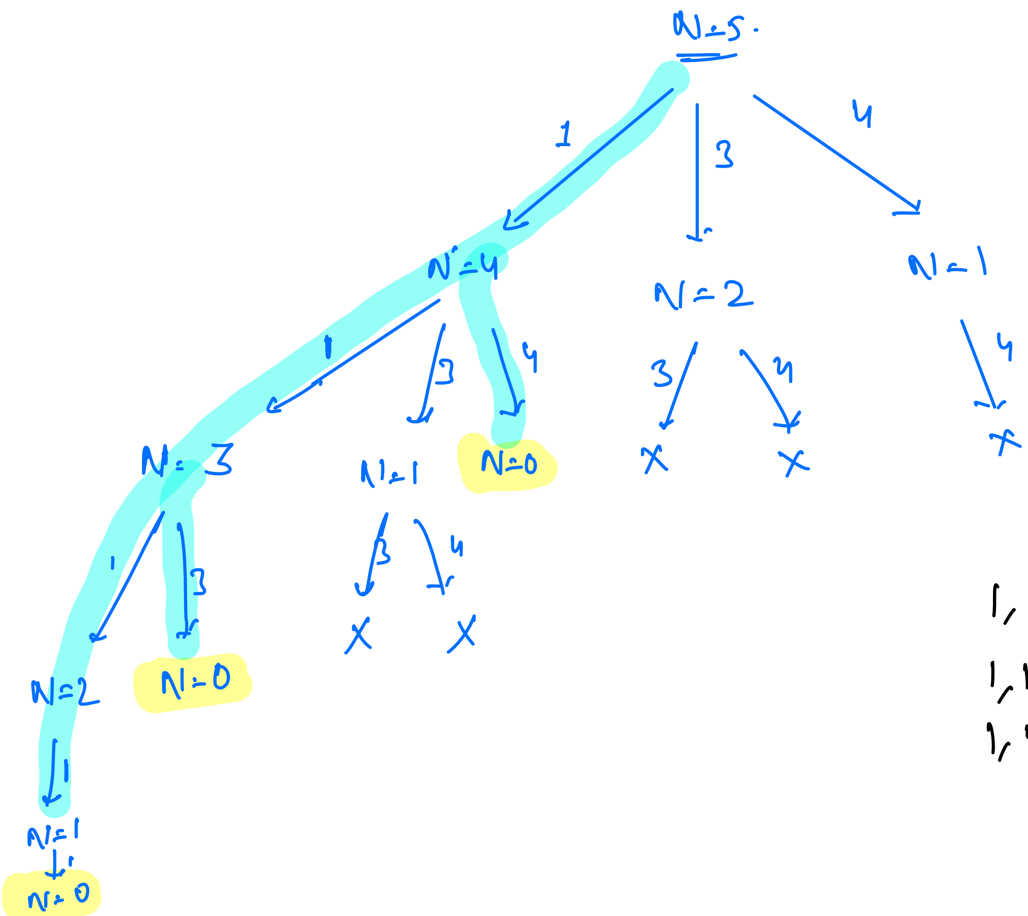
Any denomination any no. of times.

Amount = 5 denoms → [3 1 4] (x,y) = (y,x)

[1,4] [1,1,3] [1,1,1,1,1] ans = 3.

[4,1] [1,3,1]
[3,1,1]

Decide an order which will make repetitions as invalid.



1, 1, 1, 1, 1
1, 1, 3
1, 4

bottom-up.

denoms → [1, 3, 4]   Amount = 5

dp[] →

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

→ ans.

[1]	[1,1]	[1,1,1]	[1,1,1,1]	[1,1,1,1,1]
		[3]	[1,3]	[1,1,3]
			[4]	[1,4]

code →

```
dp[amount + 1],    ∀i, dp(i) = 0
dp[0] = 1;

for( j = 0;  j < denoms.length; j++){
    for( i = denoms[j]; i ≤ amount ; i++){
        dp[i] += dp[i - denoms[j]];
    }
}
return dp[amount];
```

T.C → O(N * amount)
S.C → O(amount)

0-1 KnapSack (2)

We are given N toys with their happiness and weight. Find max total happiness that can be kept in a bag with the capacity W. Here, we cannot divide the toys.

constraints.

$1 \le N \le 500$
$1 \le W \le 10^9$
$1 \le wt[i] \le 10^9$
$1 \le value[i] \le 50$

dp[N+1][W+1];
S.C → O(W)

X

type → 1.

20 lakhs.

type → 2.

L1	L2	L3	L4	L5
↓	↓	↓	↓	↓
15L	25L	19L	50L	1 Cr.

⇒ Max value with first i elements & j capacity.

⇒ Minimum weight required to get value j with first i elements.

value →	2	3	4	5	6	7
	↓	↓	↓	↓	↓	↓
min wt. required →	4	6	7	8	11	12

cap → 9.

dp[i][j] ⇒ Minimum wt. required to get value j with first i- elements.

value[] → [2 1 3]

wt[] → [3 2 4]

<u>W = 7.</u>

$\sum val(i) = 6.$

j (value) → ans.

wt	val
3	2
2	1
4	3

i \ j	0	1	2	3	4	5	6
0	0	∞	∞	∞	∞	∞	∞
1	0	∞	3	∞	∞	∞	∞
2	0	2	3	5	∞	∞	∞
3	0	2	3	4	6	7	9

Can we get value = 1,2,3,4,5,6 if there are 0 elements → <u>No</u>.

$$\text{Min}\begin{cases} f1 \rightarrow dp[i-1][j] \\ \text{if } (j - val[i-1] \geq 0) \quad f2 \rightarrow wt[i-1] + dp[i-1][j - val[i-1]] \end{cases}$$

code.-

```
sum = 0;
for( i = 0; i < N; i++){
        sum += val[i];
}

dp [N+1][sum+1];

// initialise row → 0 with ∞
// intialise col → 0 with 0.

for( i = 1; i ≤ N ; i++){
    for( j = 1; j ≤ sum; j++){
            dp[i][j] = dp[i-1][j];
            if( j - val[i-1] ≥ 0 && dp[i][j-val[i-1]] != ∞){
                dp[i][j] = Min( dp[i][j], wt[i-1] + dp[i][j - val[i-1]]);
            }
    }
}

ans = 0;
for( j = sum; j ≥ 0; j--){
    if( dp[N][j] ≤ W) { ans = j, break }
}

return ans;
```

T.C → $O(N * \Sigma val[i])$
S.C → $O(N * \Sigma val[i])$

Revise it