# DP - 4

content

- Cutting a rod
- coin sum infinite
- 0-1 knapsack 2

Contest 5 $\longrightarrow$ 26th April 9 pm

Q) Given a rod of length N & an array A of length N.

$A_i \longrightarrow$ price of i length rod

Find max value that can be obtained by cutting the rod into pieces & selling them.

N = 5

| | | | | | | |
|---|---|---|---|---|---|---|
| A = | { 0 | 1 | 4 | 2 | 5 | 6 } |
| length | 0 | 1 | 2 | 3 | 4 | 5 |

length = 5

ans = 9

| cuts | | value |
|---|---|---|
| 1 + 4 | = | 6 |
| 2 + 2 + 1 | = | 4 + 4 + 1 = 9 |
| 2 + 3 | = | 6 |
| 5 | = | 6 |
| 1 + 1 + 1 + 1 + 1 | = | 5 |
| 2 + 1 + 2 | = | 9 |
| 3 + 2 | = | 6 |
| 2 + 1 + 1 + 1 | = | 7 |

N = 5

| | | | | | | |
|---|---|---|---|---|---|---|
| A = | { 0 | 1 | 4 | 2 | 5 | 6 } |
| length | 0 | 1 | 2 | 3 | 4 | 5 |

0–∞ knapsack ∵ same length can be used many times.

Values $\longrightarrow$ A[i]

weight $\longrightarrow$ i or the length

capacity $\longrightarrow$ N

Approach 1

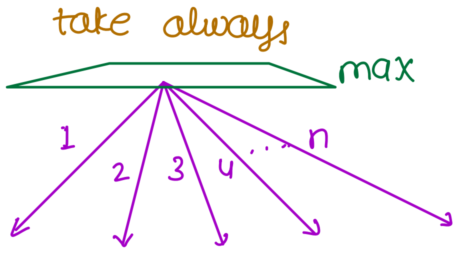
$$\left.\begin{array}{lll} take & = & \overset{dp[index][C-index]}{A[index]} \\ dont & = & dp[index-1][C] \end{array}\right\} \max$$

TC: $O(N^2)$

SC: $O(N^2)$

Approach 2

take always



TC : $O(N^2)$

SC : $O(N)$

Pseudocode $\longrightarrow$

$$\begin{array}{lccc} \text{length} & 1 & 2 & 3 \\ & \uparrow & \uparrow & \uparrow \\ A = & 1 & 2 & 3 \\ \text{index} & \boxed{0} & \boxed{1} & \boxed{2} \end{array}$$

```
int    maxRod ( L ) {        initial length
    if ( L == 0 ) {
            return 0
    }

    profit = 0
    for cut ⟶ 1 to L {
            index = cut - 1
            profit = max ( profit ,
                           A[index] + maxRod(L-cut)
                         )
    }
    return profit
}
```

Q> In how many ways can sum be equal to N by using coins given in the array
1 coin can be used multiple times

{ordered selection} $(x,y) \neq (y,x)$

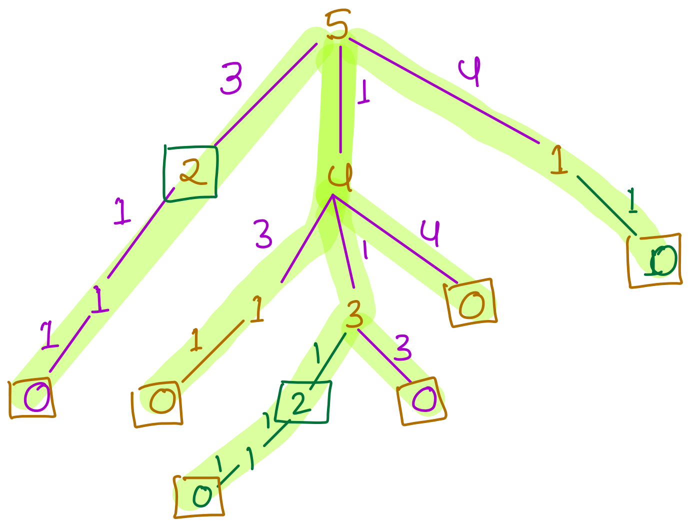
$N = 5$
$A = \{3 \ 1 \ 4\}$

ways = 6

3 1 1
1 3 1
1 1 3

4 + 1
1 + 4

1 1 1 1 1



No. of ways to get sum == 0 → 1 {dont take any of the coins}

$\underbrace{\text{No. of ways}}$ → 0 {impossible}

## Pseudocode

```
int   coinsum∞ (T) {
        if (T==0) {
            return 1
        }
        ways = 0
        for i ⟶ (0 to N-1) {
            if (T-A[i] >=0) {
                ways += coinsum∞(T-A[i])
            }
        }
        return ways
}
```

memoization ⟶ HW

Iterative

T ⟶ total to make

dp [T+1] ⟶ 0

dp[0] = 1

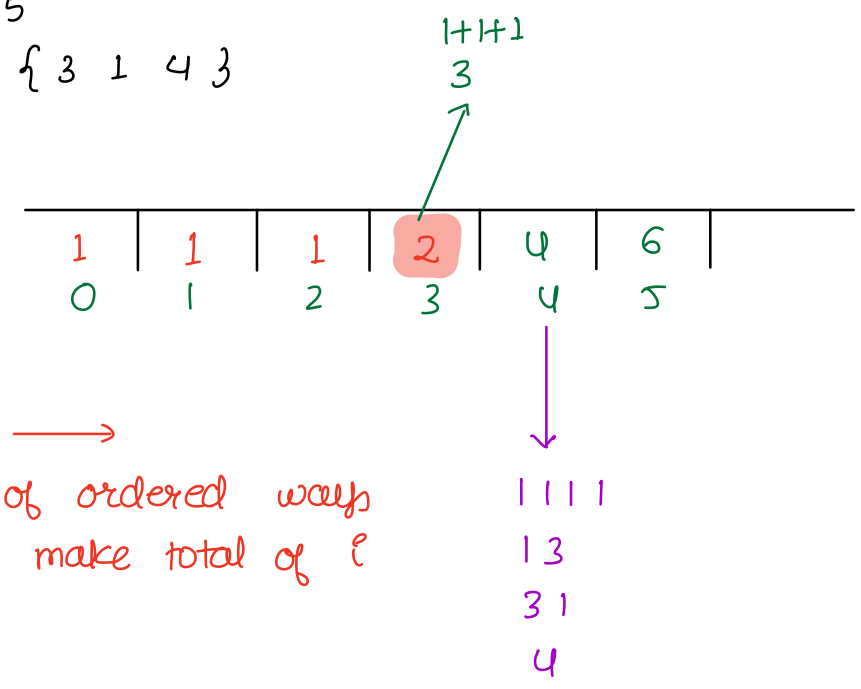
```
for t ⟶ 1 to T {
      for i ⟶ (0 to N-1) {
            if (t-A[i] >=0) {
                  dp[t] += dp[t-A[i]]
            }
      }
}
print(dp[T])
```

TC: O(NT)

SC: O(T)

$N = 5$

$A = \{3 \ 1 \ 4\}$



dp i ⟶

No. of ordered ways to make total of i

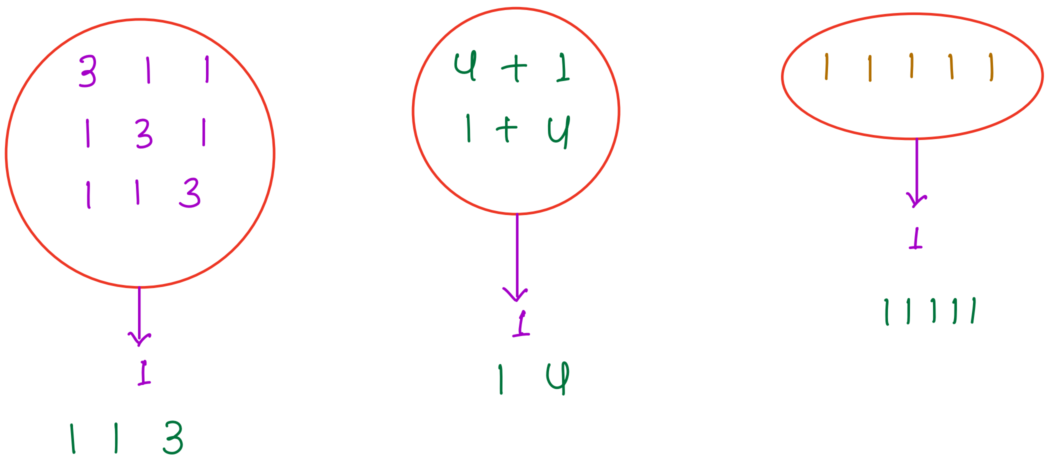
unordered selection $(x, y) = (y, x)$

0−∞ knapsack

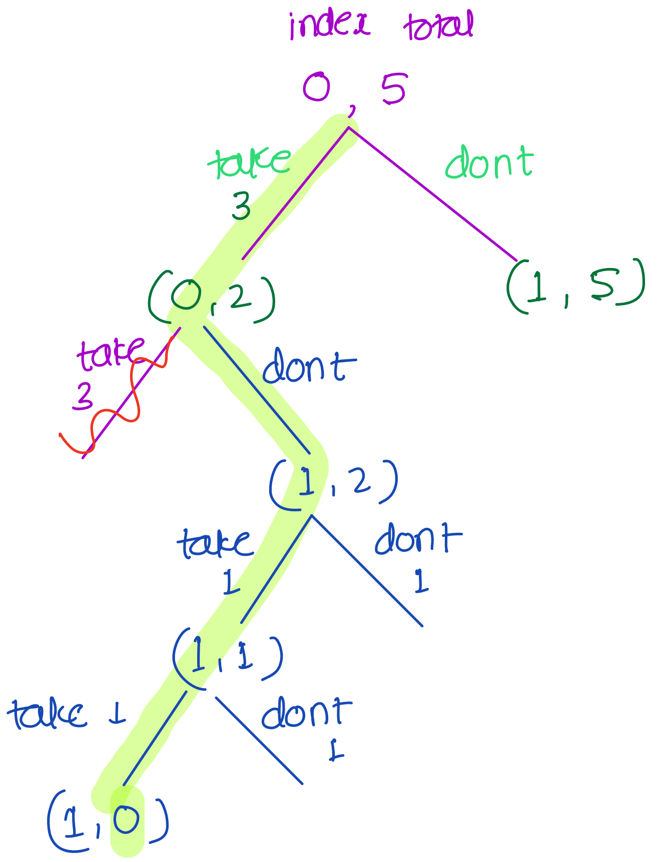
$N = 5$

$A = \{3 \quad 1 \quad 4\}$

ways = 3



coins 3 1 4



## Pseudocode

coinsumoo2 ( i , total) $\longrightarrow$ usings coins from index (0 to i ) coins No. of ways to make the total

map < String, int >

```
                      N-1      T
int  coinsumoo2 ( ~~index~~ , ~~total~~ ) {
        if(total == 0)  return 1
        if(total < 0 )  return 0
        if ( index < 0) {
            return 0
        }
        String key = index + "_" + total
        if ( map.containskey ( key ) return map.get(key)

        dont = coinsumoo2 ( index - 1, total)
        take = coinsumoo2 (index , total - A[index] )
        ways = take + dont

        map.put(key , ways)
        return ways
}
```

TC : O(NT)
SC : O(NT)

22 : 53

write the iterative code for above

3 12     6 20     5 15     2 6     4 10

(weight, value)

C = 8     max value = 27

## 0-1 Knapsack 2 { object cannot be divided }

Given N toys with their hapiness & weight .
Find max total happiness that can be kept in a bag
with capacity = C { toys cannot be divided }

$1 <= N <= 500$

$1 <= H[i] <= 50$     $O(N*C)$

$1 <= Wt[i] <= 10^9$

$1 <= C <= 10^9$     $500 * 10^9$

Normal 0-1 knapsack

$$\begin{tikzcd}
dp[N][C] \arrow[r] \arrow[r, bend left] \arrow[d] & \text{max happiness} \arrow[l, bend left] \\
\text{C cannot be store as an index} &
\end{tikzcd}$$

$\underbrace{dp[N][MH]}$ $\longrightarrow$ min capacity

min capacity needed from index 0 - N-1 toys
such that happiness is exactly MH

| 10 | 10 | 10 | 10 | 10 |
|---|---|---|---|---|
| 10 | 8 | 20 | 30 | 40 |

C = 8

what all capacities can give you a happiness of 10 ?

$\underbrace{10 \quad 8 \quad 20 \quad 30 \quad 40}_{\text{min}}$

Pseudocode

```
                                   H[]    W[]    C
                          N-1      sum(H[])
int   solve ( ~~index~~ , MH ) {        ⟶ min capacity
        if (MH == 0) {
              return 0
        }
        if( index < 0 ) { return 1000000000 }
        take = 1000000000
        dont = solve (index - 1 , MH )
        if (MH - H[index] >=0 )
        take = W[index] + solve ( index -1 ,
                                       MH - H [index])

        mcapacity = min (take, dont)
        return mcapacity
}
```

|  |  |  |
|---|---|---|
| 10 | 10 | 7 |
| 10 | 8 | 6 |

$C = 7$

$$\begin{tikzcd}
MH = & 27 \arrow[d] & 20 \arrow[d] & 17 \arrow[d] & 10 \arrow[d] & 7 \arrow[d] \\
mcap & 24 & 18 & 14 & 8 & 6
\end{tikzcd}$$

---

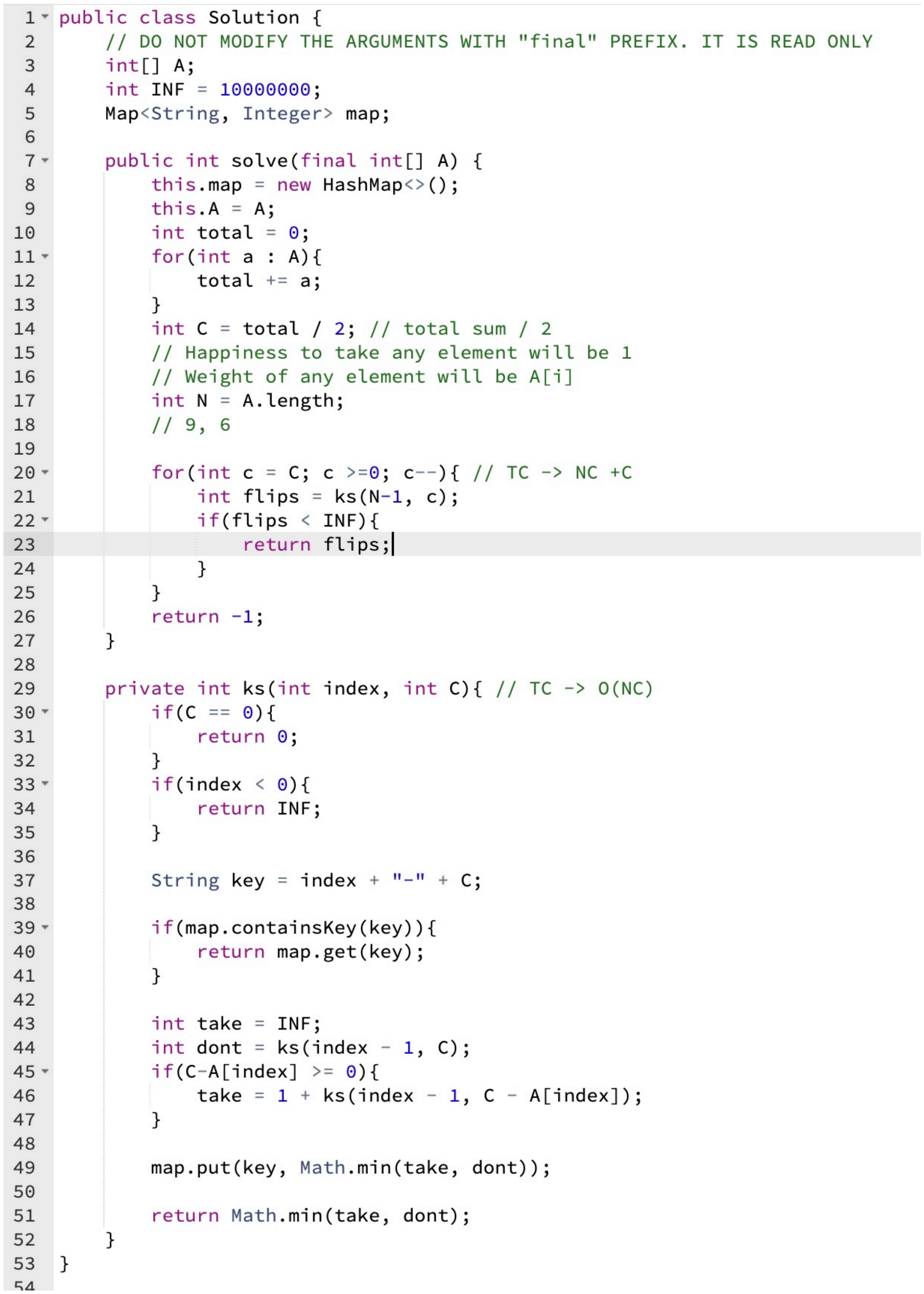
```
Iterate from MH ⟶ 0
        if dp[N-1][MH] <= C
                return MH
```

---

```
MH = sum(H[])                              sum of H
dp[MH+1]    =   ∞                              ↑
dp[0] = 0                                  TC: O(N * MH)
                                           SC: O(MH)

for i ⟶ 0 to N-1 {
    prev = copy of dp
    for h ⟶ 1 to MH {
        take = 1000000000
        dont = prev[h]
        if ( h-H[i]  >=0 )
        take = W[ i ] + prev[h-H[i]]
|     |
|     |   dp[n] = min(dp[n], take, dont)
|     }
}

Iterate from MH ⟶ 0
        if dp[N-1][MH] <= C
              return MH
```



```java
public class Solution {
    // DO NOT MODIFY THE ARGUMENTS WITH "final" PREFIX. IT IS READ ONLY
    int[] A;
    int INF = 10000000;
    Map<String, Integer> map;

    public int solve(final int[] A) {
        this.map = new HashMap<>();
        this.A = A;
        int total = 0;
        for(int a : A){
            total += a;
        }
        int C = total / 2; // total sum / 2
        // Happiness to take any element will be 1
        // Weight of any element will be A[i]
        int N = A.length;
        // 9, 6

        for(int c = C; c >=0; c--){ // TC -> NC +C
            int flips = ks(N-1, c);
            if(flips < INF){
                return flips;
            }
        }
        return -1;
    }

    private int ks(int index, int C){ // TC -> O(NC)
        if(C == 0){
            return 0;
        }
        if(index < 0){
            return INF;
        }

        String key = index + "-" + C;

        if(map.containsKey(key)){
            return map.get(key);
        }

        int take = INF;
        int dont = ks(index - 1, C);
        if(C-A[index] >= 0){
            take = 1 + ks(index - 1, C - A[index]);
        }

        map.put(key, Math.min(take, dont));

        return Math.min(take, dont);
    }
}
```