# 0/1 Knapsack

Consider the knapsack problem with objects $(P_i, W_i)$ where $i = 1, 2, \ldots n$. Weight bound $W$. Now the condition is that we must either choose an object in full or skip.

Let $f_i(y)$ be the maximum profit from items 1, 2, ... i with weight bound $y$. Then

$$f_1(y) = P_1 \text{ if } y >= W, 0 \text{ otherwise for all } y$$

$$f_{i+1}(y) = \min\{P_{i+1} + f_1(y - W_{i+1}), f_i(y)\}$$

This is the DP recurrence.

Once we can compute $f_n(W)$ we have found soln.

Example:

$wt = [3, 4, 5]$ number of elements $n = 3$

$P = [30, 50, 60]$ Capacity of the knapsack $W = 8$

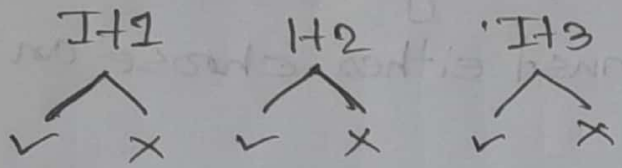
The Goal is, to find maximum profit by picking items so that:

** Item can only be picked 0 or 1 time

** Can't exceed bag limit.

In the scenario, we can picked each item or not picked.

wt = [3, 4, 5]
p = [30, 50, 60]
n = 3 W = 8

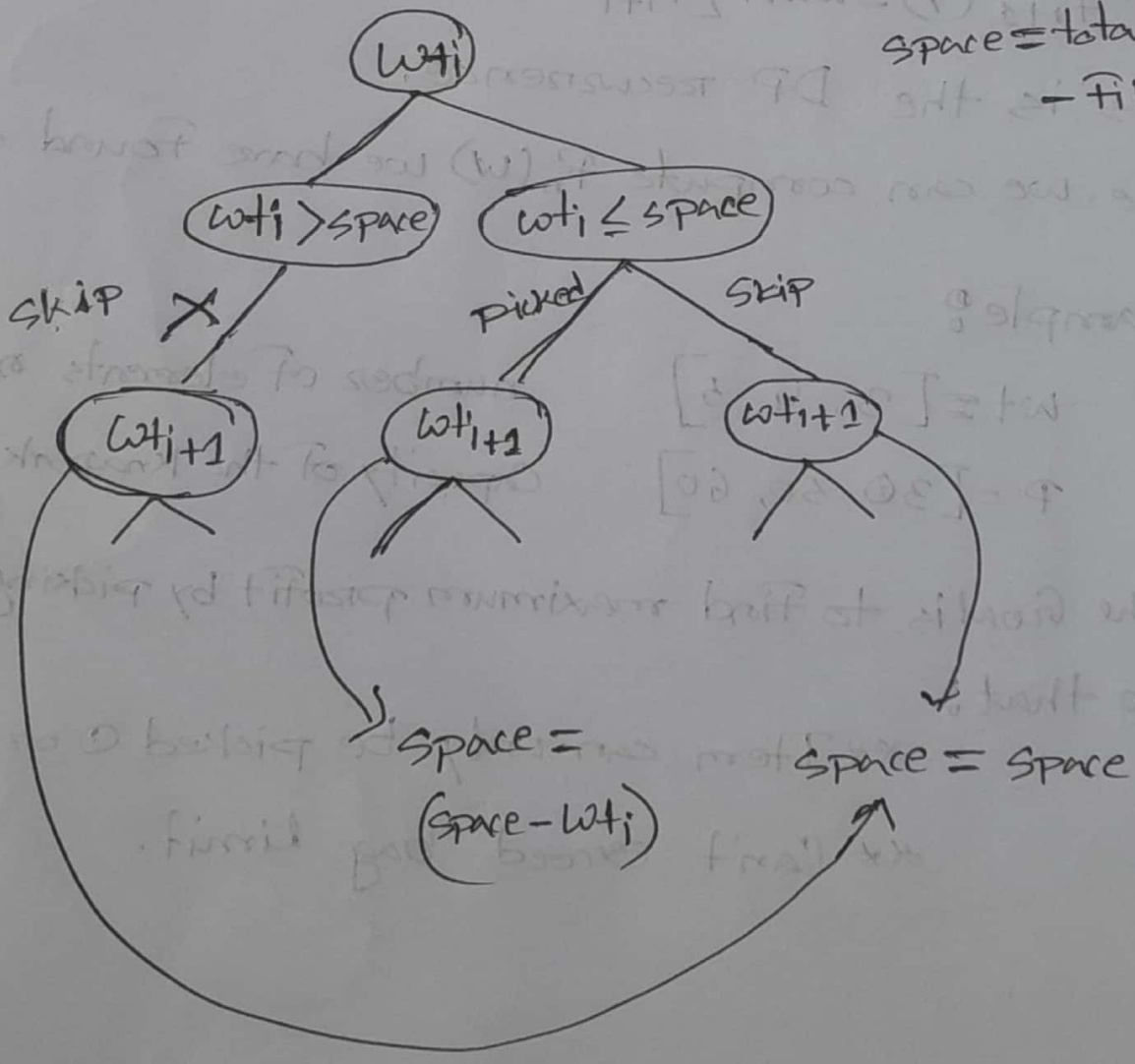



∴ total num. of combination = 2 * 2 * 2 = 8.

For 'n' item, Total combination = $2^n$.

When To Choose or Skip ?

space = total capacity − filled capacity



Recursion Soln:

wt[3,4,5] n=3
P[30,50,60] W=8

I/P: (wt[], P[], W, n)

Nth item: Include (wt[], P[], W-wt[n], n-1)

skip (wt[], P[], W, n-1)

Algorithm:

```
int knapsack(int wt[], int P[], int W, int n){
    if(n==0 || W==0)        // Base case
        return 0;

    if (wt[n-1] > W)        // skip case
        return knapsack(wt, P, W, n-1);

    else
        return max(knapsack(wt, P, W, n-1),
                   profit[n-1] + knapsack(wt, P, W-wt[n-1],
                                          n-1));
}
```

At every item can have 2 choice at max. Max combination of N items = $2^n$.

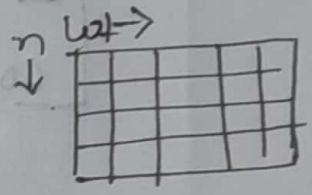
$TC = O(2^n)$

## Memoization Soln:

* For mem... generally array or map is used. But identify on what variable does your answer depend. This variable are your states. Save their values.

Max profit depends on → Bag Capacity
→ Profit on items.

```
int knapsack(wt[], p[], w, n)
    if (w==0 || n==0) return 0;
    if (mem[w][n] != -1)
        return mem[w][n]
    int result;
    if (wt[n] > w)
        result = knapsack(wt, p, w, n-1)
    else
        result = max(knapsack(wt, p, w, n-1),
                     p[n]+knapsack(wt, p, w-wt[n],
                                   n-1))
    return mem[w][n] = result;
```




** TC: O(N*W)

Tabulation Soln:

wt = [3, 4, 5] n = 3

P = [30, 50, 60] W = 8

n ↓ i, w → j

| P | wt. | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 30 | 3 | 1 | 0 | 0 | 0 | 30 | 30 | 30 | 30 | 30 | 30 |
| 50 | 4 | 2 | 0 | 0 | 0 | 30 | 50 | 50 | 50 | 80 | 80 |
| 60 | 5 | 3 | 0 | 0 | 0 | 30 | 50 | 60 | 60 | 80 | 90 |

{ I+1, I+2, I+3 } { 1 0 1 }

| P | wt |
|---|---|
| 90-60=30 | 8-5=3 |
| 30-30=0 | 3-3=0 |

*** max(dp[i-1][j], P[i-1]+dp[i-1][j-wt[i-1]])

Exercise:

P = [1, 2, 5, 6] n = 4

wt = [2, 3, 4, 5] w = 8

```
void knapsack(int wt[], int P, int W, int n)

    for i=0 to n
        for j=0 to n
            if i==0 || j==0
                dp[i][j] = 0
            else if wt[i-1] > j
                dp[i][j] = dp[i-1][j]
            else
                dp[i][j] = max(dp[i-1][j],
                               P[i-1] + dp[i-1][j-wt[i-1]])
    return dp[n][W];
```

TC: $O(n * W)$