Q → Given an integer array with non-negative integers & a target K. Check if there exist a subset sum with target K.

$$A = [\overset{0}{3} \quad \overset{1}{34} \quad \overset{2}{4} \quad \overset{3}{12} \quad \overset{4}{5} \quad \overset{5}{2}] \qquad K = \underline{11}$$

<u>Bruteforce</u> → $\forall$ subsets check if sum = K. $TC = \underline{O(2^N)}$

```
function isSubsetSum(set[], n, sum) {
    if (sum == 0) return true;
    if (n == 0 and sum != 0) return false;

    // Explore two possibilities: include the current element or exclude it
    return isSubsetSum(set, n - 1, sum) || isSubsetSum(set, n - 1, sum - set[n - 1]);
}
```

<u>Sol</u> → [0-1 Knapsack]

dp[i][j] → true, if there exist a sum j considering first 'i' elements.
→ false, else

$\forall i,j$ dp[i][j] = false
$\forall i$, dp[i][0] = true
for i → 1 to N {
    for j → 1 to K {
        dp[i][j] = dp[i-1][j] //reject i^{th} element
        if (j >= A[i-1])
            dp[i][j] |= dp[i-1][j - A[i-1]] //select
    }
} return dp[N][K]

$TC = \underline{O(N * K)}$ $\quad SC = \underline{O(N * K)}$ → $\underline{O(2K)} \to \underline{O(K)}$

Problem :

Flipkart is simplifying the shopping experience by introducing a feature that tailors product suggestions based on customers' needs and budget. Customers can specify what they're looking for and how much they can spend, and Flipkart will recommend the best options to fit their criteria. This streamlined approach helps customers find what they need quickly and within their budget, enhancing their overall satisfaction.

Problem Statement :

Given the **Budget** of the user and **cost and happiness value** of **N** items of the desired product. Compute the maximum happiness value you can get if you buy some products optimally.

N = 5 cost → [110 180 50 120 100]

h → [39 57 13 44 24]

B = 300 Ans = 101

```
function isSubsetSum( cost[], happiness[], n, sum) {
    dp = 2D array of integers of size (n + 1) x (sum + 1);

    // Initialize the first column as true, as negative infinity
    for (i -> 0 to n) dp[i][0] = -negativeinf;
    dp[0][0] = 0;

    // Fill the dp array using a bottom-up approach
    for (i -> 1 to n) {
        for (j -> 1 to sum) {
            dp[i][j] = dp[i-1][j];
            if (j >= cost[i - 1]) {
                dp[i][j] = max(dp[i][j] , dp[i - 1][j - cost[i - 1]] + happiness[i - 1]);
            }
        }
    }

    return dp[n][sum];
}
```

TC = O(N * K) SC = O(N * K)

→ O(2K) → O(K)

Q → Given an integer array.

We are initially at position A[0].

A[i] → Max length of forward jump from index 'i'.

i ⟶ upto i + A[i]

Find min # jumps to reach (N-1) index.

$$\begin{array}{cccccc} & 0 & 1 & 2 & 3 & 4 \\ A = [& 2 & 3 & 1 & 1 & 4 \;] \end{array}$$

Ans = 2

$$\begin{array}{ccccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ A = [& 2 & 3 & 0 & 2 & 0 & 5 \;] \end{array}$$

Ans = 3

Hint → $\forall i$, find min # jumps to reach 'i'.

```
inf = INT_MAX
if (A[0] == 0) return inf
jumps[0] = 0
for i → 1 to (N-1) {
    jumps[i] = inf - 1
    for j → 0 to (i-1) {
        if (j + A[j] >= i)
            jumps[i] = min(jumps[i], jumps[j] + 1)
    }
} return jumps[N-1]
```

$\boxed{j \rightarrow i}$

TC = $O(N^2)$ SC = $O(N)$

Hint → $\forall i$, find farthest index we can jump from index 0 to i.

$i \longrightarrow A[i] + i$

$$\begin{array}{cc} i-1 & i \\ 5 & 2 \rightarrow \underline{i+2} \end{array}$$

$i - 1 + 5 = \underline{i + 4}$ ✓

$$\begin{array}{cccccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ A = [& 3 & 1 & 5 & 3 & 2 & 0 & 2 & 3 & 2 \;] \\ \hookrightarrow & 3 & 3 & 7 & 7 & 7 & 7 & 8 & 10 & 10 \end{array}$$

$A[i] = \max(A[i-1], i + A[i])$

Ans = 3

```
for i → 1 to (N-1) {
    A[i] = max(A[i-1], i + A[i])
}

i = 0        jumps = 0
while (i < N-1) {
    jumps++
    if (i == A[i]) return INT_MAX
    i = A[i]
}
return jumps
```

TC = O(N)   SC = O(1)

Q → Find # of N digit numbers with digit sum = K.
(leading 0's not allowed)

N = 2   K = 4   40, 31, 22, 13   Ans = 4

min N digit number → 10^{N-1} N = 3 → 100
max N digit number → $10^{N} - 1$ → 999

```
int digitSum(N) {
    sum = 0
    while (N > 0) {
        sum += N % 10
        N = N/10
    }
    return sum
```

}

N = 3 K = 5 __ __ __ (3, 5)

(2, 4) 1 _ _ 2 _ _ 3 _ _ 4 _ _ 5 _ _

10 _ 11 _ 12 _ 13 _ 14 _

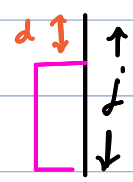
104 113 122 131 140

optimal substructure
overlapping subproblems } DP

$dp[i][j] \rightarrow$ # i digit numbers with digit sum j.

```
∀i,j  dp[i][j] = 0
for j → 1 to 9 {
    dp[1][j] = 1
}

for i → 2 to N {
    for j → 1 to K {
        for d → 0 to 9 {
            if (j >= d) {
                dp[i][j] += dp[i-1][j-d]
                dp[i][j] %= 10^9 + 7   // if mentioned in que.
            }
        }
    }
} return dp[N][K]
```



TC = O(N * K) SC = O(N * K)

$\rightarrow O(2K) \rightarrow O(K)$

Q → Given an integer array where

$A[i]$ → price of stock on i^{th} day.

We are allowed to do multiple trades (buy → sell)

but not multiple trades simultaneously.

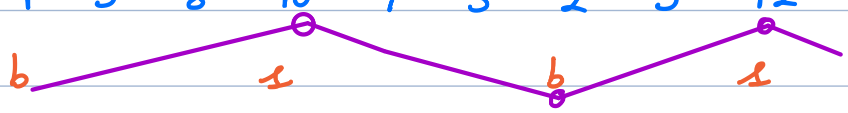
Find max profit we can earn.

$$\begin{array}{cccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ A = [& 5 & 2 & 10 & 20 & 5 & 15 & 10 \;] \\ & & b & & s & b & s & \end{array}$$

Ans $= (20-2) + (15-5)$

$= 18 + 10 = \underline{28}$

$$\begin{array}{cccccccccccccccccccc} 0 & \underline{4} & 1 & \underline{3} & 2 & \underline{2} & 3 & \underline{x} & 4 & \underline{x} & 5 & \underline{x} & 6 & \underline{3} & 7 & \underline{7} & 8 & \underline{x} & 9 \\ A = [\;1 & & 5 & & 8 & & 10 & & 7 & & 3 & & 2 & & 5 & & 12 & & 10\;] \end{array}$$



Ans $= (10-1) + (12-2) = 9 + 10 = \underline{19}$

```
ans = 0
for i → 1 to (N-1) {
    if (A[i] > A[i-1])
        ans += A[i] - A[i-1]
}
return ans
```

TC = $\underline{O(N)}$ SC = $\underline{O(1)}$