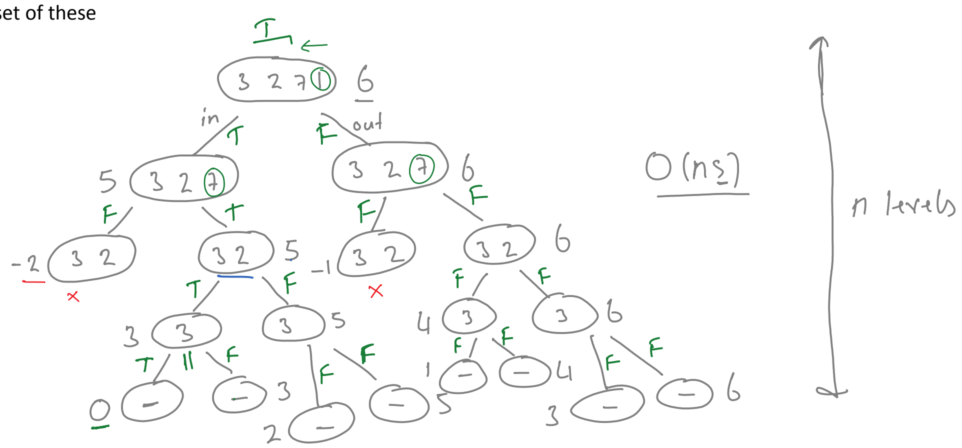
Problem 1: Subset Sum

The Subset Sum Problem is a decision problem. It takes as input a list of integers and is required to determine if a subset of these integers adds up to a given sum S.

Sample Input
4 6
3 2 7 1

Sample Output
YES

Output Details
{3, 2, 1}



overlapping subproblems ? => memoization.

Imlementation with Tabulation

A[] = 3 2 7 1 sum 6

	0	1	2	3	4	5	6 = sum
dp	1	Ø	Ø	Ø	Ø	Ø	Ø
		1	1	1	1	1	1 → Yes.

O(NS)

```
vector<int> dp(s+1, false);
dp[0] = true;

for (int i=0; i<N; i++)
    for (int j = S-A[i]; j>=0; j--)
        dp[j+A[i]] = dp[j+A[i]] || dp[j];
```

S = 10

A[] = 2 8 7 5

	0	1	2	3	4	5	6	7	8	9	10
dp	1	0	Ø	0	0	Ø	0	Ø	Ø	Ø	Ø
			1			1		1	1	1	1

Exercise 1: Number of Subsets Having Sum S
Calculate number of subsets of N elements that are selected from a given set whose sum adds up to a given number S. Your algorithms should run in O(NS) time.

Sample Input
5 7
5 2 7 3 4

5 2 7 3 4

	0	1	2	3	4	5	6	7
	1	0	Ø	Ø	Ø	Ø	Ø	Ø
			1	1	1	~~1~~	1	~~1~~
						2		~~2~~
								3

Sample Output

3

Output Details
{5, 2}
{3, 4}
{7}

```
vector <int> dp (s+1, 0);
dp[0] = 1;

for (int i = 0; i < N; i++)
    for (int j = s - A[i]; j >= 0; j--)
        dp[j + A[i]] += dp[j];

cout << dp[s] << endl;
```

Exercise 2: Printing a Subset Having Sum S

Print one of the subsets of elements that are selected from a given set whose sum adds up to a given number S. Your algorithms should run in O(NS) time.

Sample Input
5 7
5 2 7 3 4

Sample Output
{5, 2}

	0	1	2	3	4	5	6	7
dp	0	-1	~~-1~~	~~-1~~	~~-1~~	~~-1~~	~~-1~~	~~-1~~
parent			0	0	0	0	2	5

5 2 7 3 4

{2, 5}

Problem 2: Subset Sum Problem with a Large S

Given a set of N integers where N <= 40. Determine if there exist a subset having sum S where S <= 10^18. Because S is a very large number we cannot use dynamic programming solution for this problem. On the other hand, we can either not use the standard brute force solution bacuse N is to large for a O(N2 ^N) time solution.

- Brute force $O(N2^N)$ → N is big
- DP. $O(NS)$, $O(S)$ space → S is big.

Solution with "Meet in the Middle"

S = 10

5 2 7 | 3 4 9

left | right

sum	subsets
5	5
7	5, 2
12	5, 7
14	5, 2, 7
2	2
9	2, 7
7	7

subsets	sum
3	3
3, 4	7
3, 4, 9	16
4	4
4, 9	13
9	9
3, 9	12

Test if we have a solution here with brute force. NO ⟹

$(N/2) \cdot 2^{N/2}$

Generated all subsets.

Test if we have a solution in this part NO ⟹

$\frac{N}{2} \cdot 2^{N/2}$

Generated all subsets.

Test if sum of two subsets (one from the left, one from the right) is equal to S..

- sort sort subset sums of one side.
- For each subset sum of the side, make a binary search on the side.

search on this side.

$2^{N/2} \lg 2^{N/2} + 2^{N/2} \lg 2^{N/2}$

Problem 3: Unbounded Subset Sum Problem

In a classical Subset Sum Problem you may use each item only once, whereas in Unbounded Subset Sum problem repetition of items is allowed.

S = 15

3 5 9 7

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
dp	1	0	0	~~0~~ 1	0	~~0~~ 1	~~0~~ 1	~~0~~ 1	~~0~~ 1	~~0~~ 1	~~0~~ 1	~~0~~ 1	~~0~~ 1	0	~~0~~ 1	~~0~~ 1

```
vector<bool> dp[s+1, false];
dp[0] = true;
    for(int i=0; i<=s; i++)
        for(int j=0; j<N; j++)
            dp[i+A[j]] = dp[i+A[j]] || dp[i];
cout << dp[s] << endl;
```

Problem 4: Printing All Subsets Having Sum S

Given an array of integers and a sum S, the task is to print all subsets of given array with sum equal to S.

Sample Input
6 10
2 3 5 6 8 10

Sample Output
5 2 3
2 8
10

2 3 5 6 8 10

	0	1	2	3	4	5	6	7	8	9	10
dp	0	-1	-1	-1	-1	-	-1	-1	-	-1	-1
			(2)	(3)		(2,3)	(6)	(2,5)	(3,5)	(3,6)	(2,3,5)
						(5)			(2,6)		(2,8)
									(8)		(10)

	0	1	2	3	4	5	6	7	8	9	10
-	1	0	0	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0	0
3	1	0	1	1	0	1	0	~~0~~	~~0~~	0	0
5	1	0	1	1	0	2	0	1	1	0	1 → {5, 3, 2}
6	1	0	1	1	0	2	~~0~~ 1	1	~~1~~ 2	~~0~~ 1	1
8	1	0	1	1	0	2	1	1	~~2~~ 3	1	~~1~~ 2 → {8, 2}
10	1	0	1	1	0	2	1	1	3	1	~~2~~ 3 → {10}

$O(NS + NS)$ time

O(NS) space.

S = 10

6 5 2 9 1 3

9, 1

2, 3, 5

6, 1, 3

	0	1	2	3	4	5	6	7	8	9	10
0	1	0	0	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	~~0~~ 1	0	0	0	0
5	1	0	0	0	0	~~0~~ 1	1	0	0	0	0
2	1	0	~~0~~ 1	0	0	1	1	~~0~~ 1	~~0~~ 1	0	0
9	1	0	1	0	0	1	1	1	1	~~0~~ 1	0
1	1	1 ~~0~~	1	1 ~~0~~	0	1	~~1~~ 2	~~1~~ 2	~~1~~ 2	2 ~~1~~	~~0~~ 1
3	1	1	1	2 ~~1~~	1 ~~0~~	~~1~~ ~~2~~ 3	2	2	3 ~~2~~	~~2~~ 4	~~1~~ 3

dp[i][j] is in how many different ways we can make the sum j using the first i numbers.

{3, 7} {1, 9}

{3, 2, 5}

Problem 5: 0-1 Knapsack Problem

We have a set of N items each with an associated weight and value (benefit or profit).
The objective is to fill the knapsack with items such that we have a maximum profit without exceeding the weight limit of the knapsack.
This is a 0-1 Knapsack Problem where we can either take an entire item or reject it completely. We cannot break an item and fill the knapsack.
0-1 Knapsack Problem is a combinatorial optimization problem. Subset Sum problem is a special case 1-0 Knapsack Problem where items have the same weight and value (v_i == w_i).

Sample Input
4 5
3 4 4 1
100 20 60 40

Sample Output
140

Output Details
Item 1 and item 4.

w_i	v_i
3	100
4	20
4	60
1	40

C = 5

	0	1	2	3	4	5
dp	0	~~-1~~ 40	-1	~~-1~~ 100	~~-1~~ ~~20~~ ~~60~~ 140	~~-1~~ 100

dp[i] is the maximum total value for the sum of weighs is i