









Subset Sum

We'll cover the following



- Problem Statement
 - Example 1:
 - Example 2:
 - Example 3:
- Try it yourself
- Basic Solution
 - Bottom-up Dynamic Programming
 - Code
- Challenge
 - Try it yourself

Problem Statement#

Given a set of positive numbers, determine if there exists a subset whose sum is equal to a given number 'S'.

Example 1:#

Input: {1, 2, 3, 7}, S=6

Output: True

The given set has a subset whose sum is '6': {1, 2, 3}

Example 2:#







```
Input: {1, 2, 7, 1, 5}, S=10
Output: True
The given set has a subset whose sum is '10': {1, 2, 7}
```

Example 3:#

```
Input: {1, 3, 4, 8}, S=6
Output: False
The given set does not have any subset whose sum is equal to '6'.
```

Try it yourself#

Try solving this question here:



Basic Solution#

This problem follows the **0/1 Knapsack pattern** and is quite similar to Equal Subset Sum Partition. A basic brute-force solution could be to try all subsets of the given numbers to see if any set has a sum equal to 'S'.

So our brute-force algorithm will look like:



```
for each number 'i'

create a new set which INCLUDES number 'i' if it does not

process the remaining numbers

create a new set WITHOUT number 'i', and recursively process the remaining

return true if any of the above two sets has a sum equal to 'S', otherwise r
```

Since this problem is quite similar to Equal Subset Sum Partition, let's jump directly to the bottom-up dynamic programming solution.

Bottom-up Dynamic Programming#

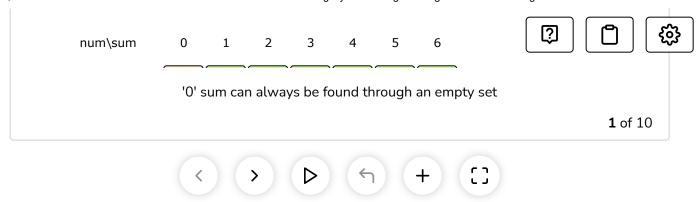
We'll try to find if we can make all possible sums with every subset to populate the array dp[TotalNumbers][S+1].

For every possible sum 's' (where $0 \le s \le S$), we have two options:

- 1. Exclude the number. In this case, we will see if we can get the sum 's' from the subset excluding this number => dp[index-1][s]
- 2. Include the number if its value is not more than 's'. In this case, we will see if we can find a subset to get the remaining sum => dp[index-1][snum[index]]

If either of the above two scenarios returns true, we can find a subset with a sum equal to 's'.

Let's draw this visually, with the example input {1, 2, 3, 7}, and start with our base case of size zero:



Code#

Here is the code for our bottom-up dynamic programming approach:

```
🤁 Python3
     def can partition(num, sum):
       n = len(num)
       dp = [[False for x in range(sum+1)] for y in range(n)]
       # populate the sum = 0 columns, as we can always form '0' sum with an empt
       for i in range(0, n):
         dp[i][0] = True
       # with only one number, we can form a subset only when the required sum is
       # equal to its value
       for s in range(1, sum+1):
         dp[0][s] = True if num[0] == s else False
       # process all subsets for all sums
       for i in range(1, n):
         for s in range(1, sum+1):
           # if we can get the sum 's' without the number at index 'i'
           if dp[i - 1][s]:
             dp[i][s] = dp[i - 1][s]
           elif s >= num[i]:
             # else include the number and see if we can find a subset to get the
             dp[i][s] = dp[i - 1][s - num[i]]
       # the bottom-right corner will have our answer.
       return dp[n - 1][sum]
```



The above solution has time and space complexity of O(N * S), where 'N' represents total numbers and 'S' is the required sum.

Challenge#

Can we further improve our bottom-up DP solution? Can you find an algorithm that has O(S) space complexity?



Try it yourself#



Interviewing soon? We've partnered with Hired so that \times companies apply to you instead of you applying to them. See how ①



Equal Subset Sum Partition



! Report an Issue

