

LeetCode Bootcamp Week 8

```
# def canPartition(self, nums: List[int]) -> bool:
    total = sum(nums)
    if total % 2 != 0:
        return False

    target = total // 2
    possible_sums = set([0])

    for num in nums:
        next_sums = set()
        for s in possible_sums:
            if s + num == target:
                return True
            next_sums.add(s + num)
            next_sums.add(s)
        possible_sums = next_sums

    return target in possible_sums
```

DescriptionEditorialSolutionsAcceptedSubmissions

All Submissions

Accepted147 / 147 testcases passed

XoTACHiXo submitted at Apr 29, 2025 12:39

EditorialSolution

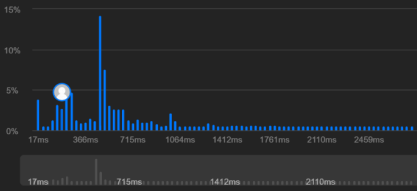
Runtime

210 msBeats 88.47%

Analyze Complexity

Memory

19.21 MBBeats 51.45%



CodePython3

```
class Solution:
    def canPartition(self, nums: List[int]) -> bool:
        total = sum(nums)
        if total % 2 != 0:
            return False

        target = total // 2
        possible_sums = set([0])

        for num in nums:
            next_sums = set()
            for s in possible_sums:
                if s + num == target:
                    return True
                next_sums.add(s + num)
```

Code

Python3Auto

```
1 class Solution:
2     def canPartition(self, nums: List[int]) -> bool:
3         total = sum(nums)
4         if total % 2 != 0:
5             return False
6
7         target = total // 2
8         possible_sums = set([0])
9
10        for num in nums:
11            next_sums = set()
12            for s in possible_sums:
13                if s + num == target:
14                    return True
15                next_sums.add(s + num)
```

SavedLn 2, Col 1

TestcaseTest Result

Case 1Case 2+

nums =

[1,5,11,5]

</> Source

```

class Solution:
    def coinChange(self, coins: List[int], amount: int) -> int:

        """
        :type coins: List[int]
        :type amount: int
        :rtype: int
        """

        # A large value greater than the maximum possible number of coins
        # required
        max_value = amount + 1

        # Initialize dp array where dp[i] represents the minimum number of coins
        # needed for amount i
        # dp[i] is initialized with max_value, which is essentially infinity
        dp = [max_value] * (amount + 1)

        # Base case: No coins are needed to make the amount 0
        dp[0] = 0

        # Iterate through each coin denomination
        for coin in coins:
            # Iterate through all amounts from the current coin value to the
            # target amount
            for x in range(coin, amount + 1):

                dp[x] = min(dp[x], dp[x - coin] + 1)

        return dp[amount] if dp[amount] != max_value else -1

```

Problem List

< > 🔍

Description

Editorial

Solutions

Accepted ×

Submissions

← All Submissions

Accepted 189 / 189 testcases passed

XoITACHIXo submitted at Apr 29, 2025 12:41

Editorial

Solution

Runtime

696 ms | Beats: 84.69%

Analyze Complexity

Memory

18.00 MB | Beats: 87.79%

Code | Python3

```

class Solution:
    def coinChange(self, coins: List[int], amount: int) -> int:
        """
        :type coins: List[int]
        :type amount: int
        :rtype: int
        """
        # A large value greater than the maximum possible number of coins required
        max_value = amount + 1

        # Initialize dp array where dp[i] represents the minimum number of coins needed for amount i
        # dp[i] is initialized with max_value, which is essentially infinity
        dp = [max_value] * (amount + 1)

```

Code

Python3

Auto

Run

Submit

🔍

📄

Testcase

Test Result

Case 1

Case 2

Case 3

+

coins =

[1,2,5]

amount =

11

```

class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        @cache
        def solve(i, must_pick):
            if i >= len(nums): return 0 if must_pick else -inf
            return max(nums[i] + solve(i+1, True), 0 if must_pick else
solve(i+1, False))
        return solve(0, False)

```

Problem List

Description Editorial Solutions Accepted Submissions

All Submissions

Accepted 210 / 210 testcases passed

XoITACHIxo submitted at Apr 29, 2025 12:43

Editorial Solution

Runtime

575 ms | Beats 5.06%

Analyze Complexity

Memory

289.48 MB | Beats 7.00%

The figure shows two performance graphs. The top graph is a bar chart representing runtime usage in milliseconds, with the x-axis ranging from 15ms to 105ms and the y-axis from 0% to 8%. The bars show a distribution of execution times, with a notable peak around 60ms. The bottom graph is a similar bar chart for memory usage, with the x-axis ranging from 31ms to 91ms and the y-axis from 0% to 8%. It shows a more concentrated distribution of memory usage across the time range.

Code | Python3

```
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        @cache
        def solve(i, must_pick):
            if i >= len(nums): return 0 if must_pick else -inf
            return max(nums[i] + solve(i+1, True), 0 if must_pick else solve(i+1, False))
        return solve(0, False)
```

Run Submit

Python3 Auto

```
1 class Solution:
2     def maxSubArray(self, nums: List[int]) -> int:
3         @cache
4         def solve(i, must_pick):
5             if i >= len(nums): return 0 if must_pick else -inf
6             return max(nums[i] + solve(i+1, True), 0 if must_pick else solve(i+1, False))
7         return solve(0, False)
```

Saved

Ln 1, Col

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

nums =
[-2,1,-3,4,-1,2,1,-5,4]

Output

6

Expected

6

Contribute a testcase