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
```

Description | Editorial | Solutions | Accepted | Submissions

All Submissions

Accepted 147 / 147 testcases passed

XoITACHIXo submitted at Apr 29, 2025 12:39

Editorial | Solution

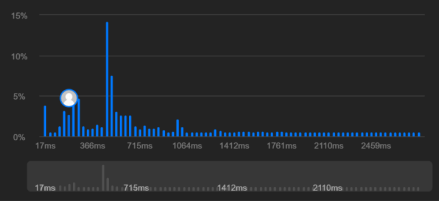
Runtime

210 ms | Beats 88.47%

Analyze Complexity

Memory

19.21 MB | Beats 51.45%



Code | Python3

```python
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])
```

Code

Python3 | Auto

```python
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)
```

Saved | Ln 2, Col 1

Testcase | Test Result

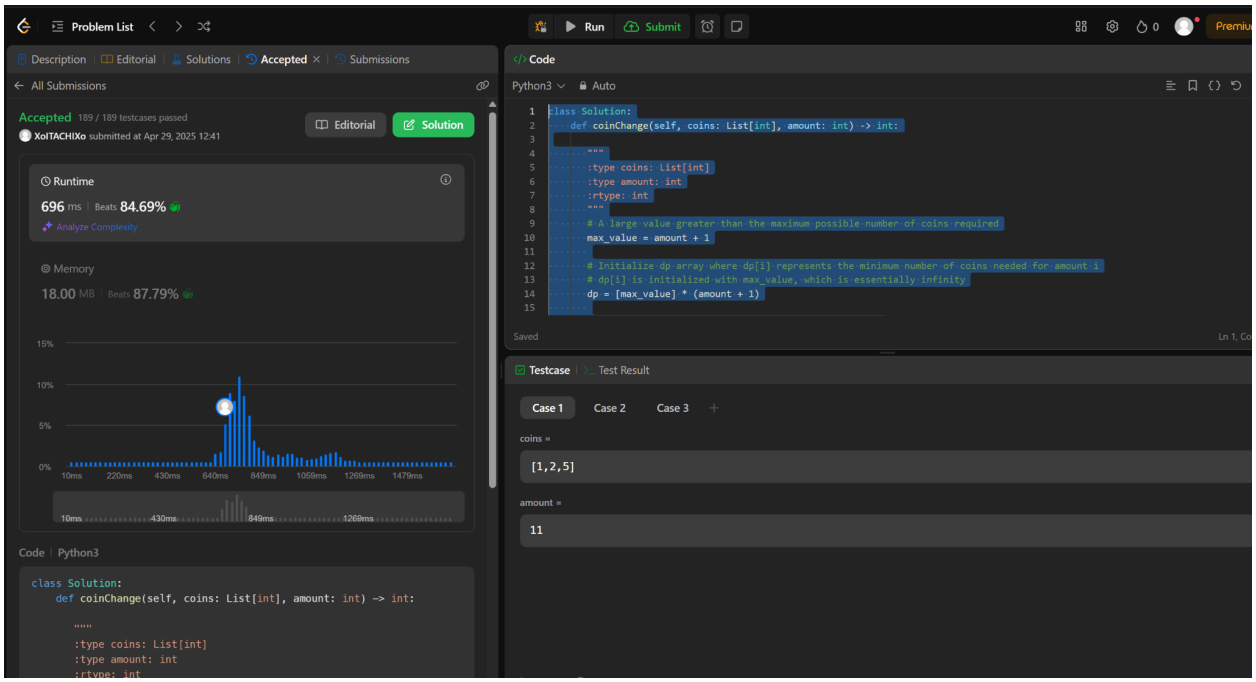
Case 1 | Case 2 | +

nums =

[1,5,11,5]

Source

```python
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
```




```python
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

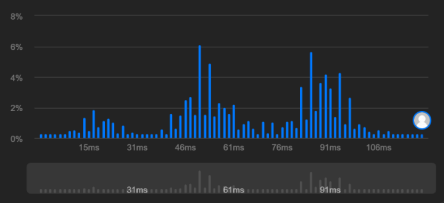
Runtime

575 ms | Beats 5.06%

Analyze Complexity

Memory

289.48 MB | Beats 7.00%



Code | Python3

```python
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
        return solve(0, False)
```

Run | Submit

Code

Python3 | Auto

```python
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)
```

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