# Dynamic Programming / Recursion Problems

## 279. Perfect Squares (https://leetcode.com/problems/perfect-squares/)

### Medium

Given a positive integer n, find the least number of perfect square numbers (for example, 1, 4, 9, 16, ...) which sum to n.

Example 1:

```
Input: n = 12
Output: 3
```

Explanation: 12 = 4 + 4 + 4.

Example 2:

```
Input: n = 13
Output: 2
```

Explanation: 13 = 4 + 9.

### Performance

- Runtime: 3080 ms, faster than 27.68% of Python3 online submissions for Perfect Squares.
- Memory Usage: 13.2 MB, less than 36.88% of Python3 online submissions for Perfect Squares.

### Complexity Analysis

```
O(n^2) in time.
O(n) in space.
```

In [ ]:

```
import math

class Solution:
    def numSquares(self, n: 'int') -> 'int':
        if n < 4:
            return n

        max_root = int(math.sqrt(n))
        if max_root == math.sqrt(n):
            return 1
        #sq_nums = list(map(lambda x: x**2, list(range(max_root+1))))

        sq_nums = []
        n_sq = {}
        for i in range(1, n + 1):
            max_root = int(math.sqrt(i))
            if (i == max_root*max_root):
                n_sq[i] = 1
                sq_nums.append(i)
            else:
                n_sq[i] = n_sq[i-1] + 1
                if i > 4:
                    for sq_num in sq_nums:
                        if sq_num > 1:
                            temp = n_sq[i-sq_num] + n_sq[sq_num]
                            if temp < n_sq[i]:
                                n_sq[i] = temp
        return n_sq[n]

my_sol = Solution()
print('Should print 0:', my_sol.numSquares(0))
print('Should print 1:', my_sol.numSquares(1))
print('Should print 2:', my_sol.numSquares(2))
print('Should print 3:', my_sol.numSquares(3))
print('Should print 1:', my_sol.numSquares(4))
print('Should print 2:', my_sol.numSquares(5))
print('Should print 3:', my_sol.numSquares(6))
print('Should print 4:', my_sol.numSquares(7))
print('Should print 2:', my_sol.numSquares(8))
print('Should print 1:', my_sol.numSquares(9))
print('Should print 2:', my_sol.numSquares(10))
print('Should print 3:', my_sol.numSquares(11))
print('Should print 3:', my_sol.numSquares(12))
```

## Faster method

Using Lagrange's four square theorem/Bachet's conjecture (https://en.wikipedia.org/wiki/Lagrange%27s_four-square_theorem) which states that every natural number can be represented as the sum of four integer squares.

## Performance

- Runtime: 52 ms, faster than 97.96% of Python3 online submissions for Perfect Squares.
- Memory Usage: 12.6 MB, less than 74.91% of Python3 online submissions for Perfect Squares.

## Complexity Analysis

```
O(n) in time
O(n) in space
```

In [ ]:

```
import math

class Solution:
    def numSquares(self, n: 'int') -> 'int':
        if n < 4:
            return n
        max_root = int(math.sqrt(n))
        if n == max_root * max_root:
            return 1
        squares = list(map(lambda x: x**2, list(range(max_root+1))))
        if self.twoSum(squares, n, 0):
            return 2
        if self.threeSum(squares, n):
            return 3
        return 4

    def twoSum(self, squares, n, start):
        left = start
        right = len(squares) - 1
        while(left <= right):
            my_sum = squares[left] + squares[right]
            if my_sum == n:
                return True
            elif my_sum > n:
                right -= 1
            else:
                left += 1
        return False

    def threeSum(self, squares, n):
        for i in range(len(squares)):
            if self.twoSum(squares, n-squares[i], i):
                return True
        return False

my_sol = Solution()
print('Should print 0:', my_sol.numSquares(0))
print('Should print 1:', my_sol.numSquares(1))
print('Should print 2:', my_sol.numSquares(2))
print('Should print 3:', my_sol.numSquares(3))
print('Should print 1:', my_sol.numSquares(4))
print('Should print 2:', my_sol.numSquares(5))
print('Should print 3:', my_sol.numSquares(6))
print('Should print 4:', my_sol.numSquares(7))
print('Should print 2:', my_sol.numSquares(8))
print('Should print 1:', my_sol.numSquares(9))
print('Should print 2:', my_sol.numSquares(10))
print('Should print 3:', my_sol.numSquares(11))
print('Should print 3:', my_sol.numSquares(12))
```

# 494. Target Sum (https://leetcode.com/problems/target-sum/)

### Medium

You are given a list of non-negative integers, a1, a2, ..., an, and a target, S. Now you have 2 symbols + and -. For each integer, you should choose one from + and - as its new symbol.

Find out how many ways to assign symbols to make sum of integers equal to target S.

Example 1:

```
Input: nums is [1, 1, 1, 1, 1], S is 3.
Output: 5
```

Explanation:

```
-1+1+1+1+1 = 3
+1-1+1+1+1 = 3
+1+1-1+1+1 = 3
+1+1+1-1+1 = 3
+1+1+1+1-1 = 3
```

There are 5 ways to assign symbols to make the sum of nums be target 3.

Note:

- The length of the given array is positive and will not exceed 20.
- The sum of elements in the given array will not exceed 1000.
- Your output answer is guaranteed to be fitted in a 32-bit integer.

## Performance

- Runtime: 204 ms, faster than 84.04% of Python3 online submissions for Target Sum.
- Memory Usage: 13.4 MB, less than 21.15% of Python3 online submissions for Target Sum.

## Complexity Analysis

```
O(n * m) in time (m is the number of possible sums)
O(m) in space
```

In [ ]:

```
# Timed out!!!
class Solution:
    def findTargetSumWays(self, nums: 'List[int]', S: 'int') -> 'int':
        n = len(nums)
        count_sum = 0
        for i in range(2**n):
            if self.dot(self.binList(i, n), nums) == S:
                count_sum += 1
        return count_sum

    def binList(self, i, n):
        return [1 if digit=='1' else -1 for digit in bin(i)[2:].zfill(n)]

    def dot(self, u, v):
        return sum(x*y for (x, y) in zip(u, v))

my_sol = Solution()
print('Should print 5:', my_sol.findTargetSumWays([1, 1, 1, 1, 1], 3))
print('Should print 1:', my_sol.findTargetSumWays([1], 1))
```

In [ ]:

```python
# Timed out!!!
class Solution:
    def __init__(self):
        self.count = 0

    def findTargetSumWays(self, nums: 'List[int]', S: 'int') -> 'int':
        self.findTargetSumWaysRec(nums, S, 0, 0)
        return self.count

    def findTargetSumWaysRec(self, nums, S, right, nsum):
        if right == 0:
            self.count = 0
        if right == len(nums):
            if nsum == S:
                self.count += 1
        else:
            self.findTargetSumWaysRec(nums, S, right + 1, nsum + nums[right]);
            self.findTargetSumWaysRec(nums, S, right + 1, nsum - nums[right]);

my_sol = Solution()
print('Should print 5:', my_sol.findTargetSumWays([1, 1, 1, 1, 1], 3))
print('Should print 1:', my_sol.findTargetSumWays([1], 1))
```

In [ ]:

```python
from collections import defaultdict

class Solution:
    def findTargetSumWays(self, nums: 'List[int]', S: 'int') -> 'int':
        count_sum = defaultdict(int)
        count_sum[0] = 1
        for num in nums:
            new_count_sum = defaultdict(int)
            for k in count_sum:
                new_count_sum[k + num] += count_sum[k]
                new_count_sum[k - num] += count_sum[k]
                #print('current num = {}, count_sum[s={:3}] = {}, new_count_sum[s={:3}]
            #print('')
            count_sum = new_count_sum
        return count_sum[S]

my_sol = Solution()
print('Should print 5:', my_sol.findTargetSumWays([1, 1, 1, 1, 1], 3))
print('Should print 1:', my_sol.findTargetSumWays([1], 1))
print('Should print 5:', my_sol.findTargetSumWays([1, 2, 3, 4, 5], 3))
```

## 198. House Robber (https://leetcode.com/problems/house-robber/)

### Easy

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight without alerting the police.

Example 1:

```
Input: [1,2,3,1]
Output: 4
```

Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3). Total amount you can rob = 1 + 3 = 4.
Example 2:

```
Input: [2,7,9,3,1]
Output: 12
```

Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1). Total amount you can rob = 2 + 9 + 1 = 12.

### Performance

- Runtime: 36 ms, faster than 69.78% of Python3 online submissions for House Robber.
- Memory Usage: 13.2 MB, less than 5.00% of Python3 online submissions for House Robber.

### Complexity Analysis

```
O(n) in time.
O(n) in space.
```

In [ ]:

```
class Solution:
    def rob(self, nums: 'List[int]') -> 'int':
        if len(nums) == 0:
            return 0
        return self.robRec(nums, 0)

    def robRec(self, nums: 'List[int]', rob_sum: int) -> 'int':
        if len(nums) == 1:
            return nums[0] + rob_sum
        if len(nums) == 2:
            return max(nums) +rob_sum
        if len(nums) > 2:
            # max(rob nums[0], don't rob nums[0])
            return max(self.robRec(nums[2:], rob_sum+nums[0]) , self.robRec(nums[1:], r

my_sol = Solution()
print('Should print 4', my_sol.rob([1,2,3,1]))
print('Should print 12', my_sol.rob([2,7,9,3,1]))
```

In [ ]:

```
class Solution:
    def rob(self, nums: 'List[int]') -> 'int':
        sums = [0]*(len(nums)+2)
        for i in range(len(nums)):
            sums[i] = max(nums[i] + sums[i-2], sums[i-1])
        #print(sums)
        return sums[len(nums)-1]

my_sol = Solution()
print('Should print 4', my_sol.rob([1,2,3,1]))
print('Should print 12', my_sol.rob([2,7,9,3,1]))
```

In [ ]: