

Advanced Programming LAB II

EXPERIMENT - 7

Submitted by,

Jiya | 22BCS14856

22BCS_FL_IOT-601 (A)

53. Maximum Subarray

<https://leetcode.com/problems/maximum-subarray/description/>

```
class Solution {
    public int maxSubArray(int[] nums) {
        int n = nums.length;
        int max = Integer.MIN_VALUE, sum = 0;
        for(int i=0;i<n;i++){
            sum += nums[i];
            max = Math.max(sum,max);
            if(sum<0) sum = 0;
        }
        return max;
    }
}
```

53. Maximum Subarray

Medium Topics Companies

Given an integer array `nums`, find the **subarray** with the largest sum, and return its sum.

Example 1:

Input: `nums = [-2,1,-3,4,-1,2,1,-5,4]`
Output: 6
Explanation: The subarray `[4,-1,2,1]` has the largest sum 6.

Example 2:

Input: `nums = [1]`
Output: 1
Explanation: The subarray `[1]` has the largest sum 1.

Example 3:

Input: `nums = [5,4,-1,7,8]`
Output: 23
Explanation: The subarray `[5,4,-1,7,8]` has the largest sum 23.

Constraints:

- $1 \leq \text{nums.length} \leq 10^5$
- $-10^4 \leq \text{nums}[i] \leq 10^4$

Follow up: If you have figured out the $O(n)$ solution, try coding another solution using the **divide and conquer** approach, which is more subtle.

Seen this question in a real interview before? 1/5

35.4K 349 450 Online

Code

```
1 class Solution {
2     public int maxSubArray(int[] nums) {
3         int n = nums.length;
4         int max = Integer.MIN_VALUE, sum = 0;
5         for(int i=0;i<n;i++){
6             sum += nums[i];
7             max = Math.max(sum,max);
8             if(sum<0) sum = 0;
9         }
10        return max;
11    }
12 }
```

Saved Ln 12, Col 2

Testcase Test Result Accepted

All Submissions

Accepted 210 / 210 testcases passed
Jya submitted at Apr 03, 2025 19:37

Runtime 1 ms Beats 99.56%
Memory 57.56 MB Beats 8.80%

Analyze Complexity

Bar chart showing runtime performance across different test cases. The first bar (1ms) is significantly higher than the others, indicating it is the slowest test case.

322. Coin Change

<https://leetcode.com/problems/coin-change/description/>

```
class Solution {
    public int coinChange(int[] coins, int amount) {
        int n = coins.length;
        int[] dp = new int[amount+1];
        Arrays.fill(dp, amount+1);
        dp[0] = 0;
        for(int i=1; i<=amount; i++) {
            for(int c : coins) {
                if(i-c >= 0) {
                    dp[i] = Math.min(dp[i], dp[i-c] + 1);
                }
            }
        }
        if(dp[amount]>amount) {
            return -1;
        }
        return dp[amount];
    }
}
```

The screenshot displays the LeetCode interface for the "322. Coin Change" problem. On the left, the problem description states: "You are given an integer array `coins` representing coins of different denominations and an integer `amount` representing a total amount of money. Return the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return `-1`. You may assume that you have an infinite number of each kind of coin."

Three examples are provided:

- Example 1:** Input: `coins = [1,2,5]`, `amount = 11`. Output: `3`. Explanation: `11 = 5 + 5 + 1`.
- Example 2:** Input: `coins = [2]`, `amount = 3`. Output: `-1`.
- Example 3:** Input: `coins = [1]`, `amount = 0`. Output: `0`.

Constraints:

- $1 \leq \text{coins.length} \leq 12$
- $1 \leq \text{coins}[i] \leq 2^{31} - 1$
- $0 \leq \text{amount} \leq 10^4$

At the bottom left, it shows "19.7K" votes, "164" comments, and "321 Online" users.

On the right, the "Code" editor shows a Java solution using dynamic programming. The code initializes a `dp` array with `amount+1` and fills it with `amount+1`. It then iterates through each coin and updates the `dp` array. The final result is `dp[amount]` if it is less than or equal to `amount`, otherwise `-1`.

Below the code editor, the "Testcase" tab shows "Accepted 189 / 189 testcases passed". The "Runtime" section indicates "15 ms" and "Beats 77.31%". The "Memory" section indicates "44.21 MB" and "Beats 79.91%". A bar chart shows the distribution of runtime times, with a peak at 15ms.

91. Decode Ways

<https://leetcode.com/problems/decode-ways/description/>

```
class Solution {
    public int numDecodings(String s) {
        int n = s.length();
        if(s.charAt(0)=='0')
            return 0;

        int[] dp = new int[n+1];
        dp[0] = 1;
        dp[1] = 1;
        for(int i=2;i<=n;i++) {
            int one = Integer.valueOf(s.substring(i-1,i));
            int two = Integer.valueOf(s.substring(i-2,i));
            if(one>=1 && one<=9) {
                dp[i] = dp[i] + dp[i-1];
            }
            if(two>=10 && two<=26) {
                dp[i] = dp[i] + dp[i-2];
            }
        }
        return dp[n];
    }
}
```

DescriptionEditorialSolutionsSubmissions

91. Decode Ways

MediumTopicsCompanies

You have intercepted a secret message encoded as a string of numbers. The message is **decoded** via the following mapping:

"1" -> 'A'

"2" -> 'B'

...

"25" -> 'Y'

"26" -> 'Z'

However, while decoding the message, you realize that there are many different ways you can decode the message because some codes are contained in other codes ("2" and "5" vs "25").

For example, "11106" can be decoded into:

- "AAJF" with the grouping (1, 1, 10, 6)
- "KJF" with the grouping (11, 10, 6)

The grouping (1, 11, 06) is invalid because "06" is not a valid code (only "6" is valid).

Note: there may be strings that are impossible to decode.

Given a string s containing only digits, return the **number of ways to decode** it. If the entire string cannot be decoded in any valid way, return 0.

The test cases are generated so that the answer fits in a 32-bit integer.

Example 1:

Input: s = "12"

Output: 2

12.3K253106 Online

Solved

JavaAuto

```
1 class Solution {
2     public int numDecodings(String s) {
3         int n = s.length();
4         if(s.charAt(0)=='0')
5             return 0;
6
7         int[] dp = new int[n+1];
8         dp[0] = 1;
9         dp[1] = 1;
10        for(int i=2;i<=n;i++) {
11            int one = Integer.valueOf(s.substring(i-1,i));
12            int two = Integer.valueOf(s.substring(i-2,i));
13            if(one>=1 && one<=9) {
14                dp[i] = dp[i] + dp[i-1];
15            }
16            if(two>=10 && two<=26) {
17                dp[i] = dp[i] + dp[i-2];
18            }
19        }
20        return dp[n];
21    }
22 }
```

Ln 17, Col 41

TestcaseTest ResultAccepted

All Submissions

Accepted269 / 269 testcases passed

Jiya submitted at Apr 03, 2025 19:53

Runtime

1 msBeats 82.95%

Memory

42.28 MBBeats 16.87%

Analyze Complexity

0%

25%

50%

75%

1ms

2ms

3ms

4ms

5ms

279. Perfect Squares

<https://leetcode.com/problems/perfect-squares/description/>

```
class Solution {
    public int numSquares(int n) {
        int[] dp = new int[n + 1];
        Arrays.fill(dp, Integer.MAX_VALUE);
        dp[0] = 0;
        for (int i = 1; i <= n; ++i) {
            int min_val = Integer.MAX_VALUE;
            for (int j = 1; j * j <= i; ++j) {
                min_val = Math.min(min_val, dp[i - j * j] + 1);
            }
            dp[i] = min_val;
        }
        return dp[n];
    }
}
```

279. Perfect Squares

Medium

Given an integer n , return the least number of perfect square numbers that sum to n .

A **perfect square** is an integer that is the square of an integer; in other words, it is the product of some integer with itself. For example, 1, 4, 9, and 16 are perfect squares while 3 and 11 are not.

Example 1:
Input: $n = 12$
Output: 3
Explanation: $12 = 4 + 4 + 4$.

Example 2:
Input: $n = 13$
Output: 2
Explanation: $13 = 4 + 9$.

Constraints:

- $1 \leq n \leq 10^4$

Seen this question in a real interview before? 1/5

Accepted 941.8K | Submissions 1.7M | Acceptance Rate 55.5%

Topics: Dynamic Programming, Array

Companies: Google, Facebook, Microsoft, Amazon, Apple, etc.

11.5K | 112 | 35 Online

```
class Solution {
    public int numSquares(int n) {
        int[] dp = new int[n + 1];
        Arrays.fill(dp, Integer.MAX_VALUE);
        dp[0] = 0;
        for (int i = 1; i <= n; ++i) {
            int min_val = Integer.MAX_VALUE;
            for (int j = 1; j * j <= i; ++j) {
                min_val = Math.min(min_val, dp[i - j * j] + 1);
            }
            dp[i] = min_val;
        }
        return dp[n];
    }
}
```

Accepted 589 / 589 testcases passed

Java submitted at Apr 03, 2025 20:00

Runtime: 22 ms | Beats: 88.91% | Memory: 42.82 MB | Beats: 59.51%

Bar chart showing runtime distribution: 3ms, 7ms, 14ms, 22ms, 25ms, 36ms, 40ms, 51ms.