[**Maximum Subarray**](https://leetcode.com/problems/maximum-subarray/)

#include <vector>

#include <climits>

using namespace std;

class Solution {

public:

int maxSubArray(vector<int>& nums) {

int maxSum = INT\_MIN, currentSum = 0;

for (int num : nums) {

currentSum += num;

maxSum = max(maxSum, currentSum);

if (currentSum < 0) currentSum = 0;

}

return maxSum;

  }

};

**Longest Increasing Subsequence**

#include <vector>

#include <algorithm>

using namespace std;

class Solution {

public:

int lengthOfLIS(vector<int>& nums) {

int n = nums.size();

if (n == 0) return 0;

vector<int> dp(n, 1);

int maxLen = 1;

for (int i = 1; i < n; i++) {

for (int j = 0; j < i; j++) {

if (nums[j] < nums[i]) {

dp[i] = max(dp[i], dp[j] + 1);

}

}

maxLen = max(maxLen, dp[i]);

}

return maxLen;

}

};

**Perfect Squares**

#include <vector>

#include <cmath>

#include <algorithm>

#include <limits> // Added for INT\_MAX

using namespace std;

class Solution {

public:

int numSquares(int n) {

vector<int> dp(n + 1, INT\_MAX);

dp[0] = 0;

for (int i = 1; i <= n; i++) { // Fixed typo

for (int j = 1; j \* j <= i; j++) {

dp[i] = min(dp[i], 1 + dp[i - j \* j]);

}

}

return dp[n];

  }

};

**House Robber**

class Solution {

public:

int rob(vector<int>& nums) {

if (nums.empty()) return 0;

if (nums.size() == 1) return nums[0];

vector<int> dp(nums.size());

dp[0] = nums[0];

dp[1] = max(nums[0], nums[1]);

for (int i = 2; i < nums.size(); i++) {

dp[i] = max(dp[i-1], dp[i-2] + nums[i]);

}

return dp.back();

    }

};