<https://leetcode.com/problems/min-cost-climbing-stairs/description/>

You are given an integer array cost where cost[i] is the cost of ith step on a staircase. Once you pay the cost, you can either climb one or two steps.

You can either start from the step with index 0, or the step with index 1.

Return *the minimum cost to reach the top of the floor*.

**Example 1:**

**Input:** cost = [10,15,20]

**Output:** 15

**Explanation:** You will start at index 1.

- Pay 15 and climb two steps to reach the top.

The total cost is 15.

**Example 2:**

**Input:** cost = [1,100,1,1,1,100,1,1,100,1]

**Output:** 6

**Explanation:** You will start at index 0.

- Pay 1 and climb two steps to reach index 2.

- Pay 1 and climb two steps to reach index 4.

- Pay 1 and climb two steps to reach index 6.

- Pay 1 and climb one step to reach index 7.

- Pay 1 and climb two steps to reach index 9.

- Pay 1 and climb one step to reach the top.

The total cost is 6.

**Constraints:**

2 <= cost.length <= 1000

0 <= cost[i] <= 999

**Attempt 1: 2025-05-26**

**Solution 1: Native DFS (10 min. TLE 259/285)**

**Style 1: Base case as n == 0 or n == 1**

class Solution {

    public int minCostClimbingStairs(int[] cost) {

        int len = cost.length;

        return Math.min(helper(cost, len - 1), helper(cost, len - 2));

    }

    private int helper(int[] cost, int n) {

        if(n < 0) {

            return 0;

        }

        if(n == 0 || n == 1) {

            return cost[n];

        }

        return cost[n] + Math.min(helper(cost, n - 1), helper(cost, n - 2));

    }

}

Time Complexity: O(2^n)

Space Complexity: O(n)

**Style 2: Base case as n == len - 2 or n == len - 1**

class Solution {

    public int minCostClimbingStairs(int[] cost) {

        return Math.min(helper(cost, 0), helper(cost, 1));

    }

    private int helper(int[] cost, int n) {

        if(n == cost.length - 1 || n == cost.length - 2) {

            return cost[n];

        }

        return cost[n] + Math.min(helper(cost, n + 1), helper(cost, n + 2));

    }

}

Time Complexity: O(2^n)

Space Complexity: O(n)

**Solution 2: Memoization (10 min)**

**Style 1: Base case as n == 0 or n == 1**

class Solution {

    public int minCostClimbingStairs(int[] cost) {

        int len = cost.length;

        Integer[] memo = new Integer[len + 1];

        return Math.min(helper(cost, len - 1, memo), helper(cost, len - 2, memo));

    }

    private int helper(int[] cost, int n, Integer[] memo) {

        if(n < 0) {

            return 0;

        }

        if(n == 0 || n == 1) {

            return cost[n];

        }

        if(memo[n] != null) {

            return memo[n];

        }

        return memo[n] = cost[n] + Math.min(helper(cost, n - 1, memo), helper(cost, n - 2, memo));

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Style 2: Base case as n == len - 2 or n == len - 1**

class Solution {

    public int minCostClimbingStairs(int[] cost) {

        Integer[] memo = new Integer[cost.length + 1];

        return Math.min(helper(cost, 0, memo), helper(cost, 1, memo));

    }

    private int helper(int[] cost, int n, Integer[] memo) {

        if(n == cost.length - 1 || n == cost.length - 2) {

            return cost[n];

        }

        if(memo[n] != null) {

            return memo[n];

        }

        return memo[n] = cost[n] + Math.min(helper(cost, n + 1, memo), helper(cost, n + 2, memo));

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Solution 3: DP (10 min)**

**Style 1: Base case as n == 0 or n == 1**

class Solution {

    public int minCostClimbingStairs(int[] cost) {

        int len = cost.length;

        int[] dp = new int[len];

        dp[0] = cost[0];

        dp[1] = cost[1];

        for(int i = 2; i < len; i++) {

            dp[i] = cost[i] + Math.min(dp[i - 1], dp[i - 2]);

        }

        return Math.min(dp[len - 1], dp[len - 2]);

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Style 2: Base case as n == len - 2 or n == len - 1**

class Solution {

    public int minCostClimbingStairs(int[] cost) {

        int len = cost.length;

        int[] dp = new int[len];

        dp[len - 1] = cost[len - 1];

        dp[len - 2] = cost[len - 2];

        for(int i = len - 3; i >= 0; i--) {

            dp[i] = cost[i] + Math.min(dp[i + 1], dp[i + 2]);

        }

        return Math.min(dp[0], dp[1]);

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Solution 3: Space Optimized DP (10 min)**

**Style 1: Base case as n == 0 or n == 1**

class Solution {

    public int minCostClimbingStairs(int[] cost) {

        int len = cost.length;

        int a = cost[0];

        int b = cost[1];

        int c = 0;

        // The iteration relation:

        // Initial status:  a     b     c

        // start forwards:      new\_a new\_b new\_c

        // so new value for a is old b

        // new value for b is old c

        for(int i = 2; i < len; i++) {

            c = cost[i] + Math.min(a, b);

            a = b;

            b = c;

        }

        return Math.min(a, b);

    }

}

Time Complexity: O(n)

Space Complexity: O(1)

**Style 2: Base case as n == len - 2 or n == len - 1**

class Solution {

    public int minCostClimbingStairs(int[] cost) {

        int len = cost.length;

        int a = cost[len - 2];

        int b = cost[len - 1];

        int c = 0;

        // The iteration relation:

        // Initial status:         c     a     b

        // start backwards:new\_c new\_a new\_b

        // so new value for b is old a

        // new value for a is old c

        for(int i = len - 3; i >= 0; i--) {

            c = cost[i] + Math.min(a, b);

            b = a;

            a = c;

        }

        return Math.min(a, b);

    }

}

Time Complexity: O(n)

Space Complexity: O(1)

**Refer to**

<https://leetcode.com/problems/min-cost-climbing-stairs/solutions/476388/4-ways-step-by-step-from-recursion-top-down-dp-bottom-up-dp-fine-tuning/>

We start at either step 0 or step 1. The target is to reach either last or second last step, whichever is minimum.

**Step 1 - Identify a recurrence relation between subproblems.** In this problem,

Recurrence Relation:

mincost(i) = cost[i]+min(mincost(i-1), mincost(i-2))

Base cases:

mincost(0) = cost[0]

mincost(1) = cost[1]

**Step 2 - Convert the recurrence relation to recursion**

// Recursive Top Down - O(2^n) Time Limit Exceeded

public int minCostClimbingStairs(int[] cost) {

    int n = cost.length;

    return Math.min(minCost(cost, n-1), minCost(cost, n-2));

}

private int minCost(int[] cost, int n) {

    if (n < 0) return 0;

    if (n==0 || n==1) return cost[n];

    return cost[n] + Math.min(minCost(cost, n-1), minCost(cost, n-2));

}

**Step 3 - Optimization 1 - Top Down DP - Add memoization to recursion** - From exponential to linear.

// Top Down Memoization - O(n) 1ms

int[] dp;

public int minCostClimbingStairs(int[] cost) {

    int n = cost.length;

    dp = new int[n];

    return Math.min(minCost(cost, n-1), minCost(cost, n-2));

}

private int minCost(int[] cost, int n) {

    if (n < 0) return 0;

    if (n==0 || n==1) return cost[n];

    if (dp[n] != 0) return dp[n];

    dp[n] = cost[n] + Math.min(minCost(cost, n-1), minCost(cost, n-2));

    return dp[n];

}

**Step 4 - Optimization 2 -Bottom Up DP - Convert recursion to iteration** - Getting rid of recursive stack

// Bottom up tabulation - O(n) 1ms

public int minCostClimbingStairs(int[] cost) {

    int n = cost.length;

    int[] dp = new int[n];

    for (int i=0; i<n; i++) {

        if (i<2) dp[i] = cost[i];

        else dp[i] = cost[i] + Math.min(dp[i-1], dp[i-2]);

    }

    return Math.min(dp[n-1], dp[n-2]);

}

**Step 5 - Optimization 3 - Fine Tuning - Reduce O(n) space to O(1)**.

// Bottom up computation - O(n) time, O(1) space

public int minCostClimbingStairs(int[] cost) {

    int n = cost.length;

    int first = cost[0];

    int second = cost[1];

    if (n<=2) return Math.min(first, second);

    for (int i=2; i<n; i++) {

        int curr = cost[i] + Math.min(first, second);

        first = second;

        second = curr;

    }

    return Math.min(first, second);

}

**Refer to**

[L70.Climbing Stairs](note://WEBce2fed61238e31617ae6d19bbb7f7a01)