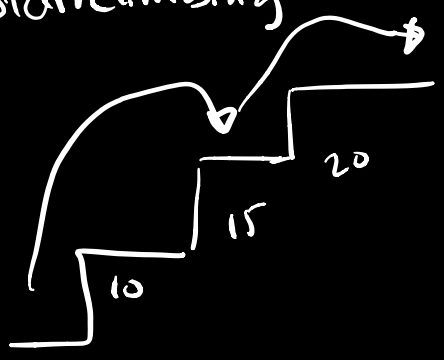
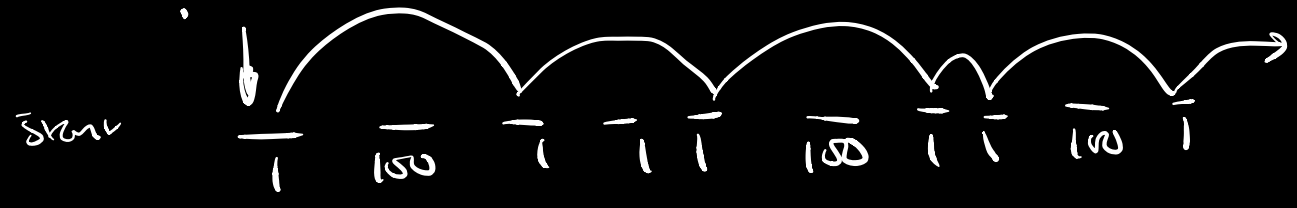
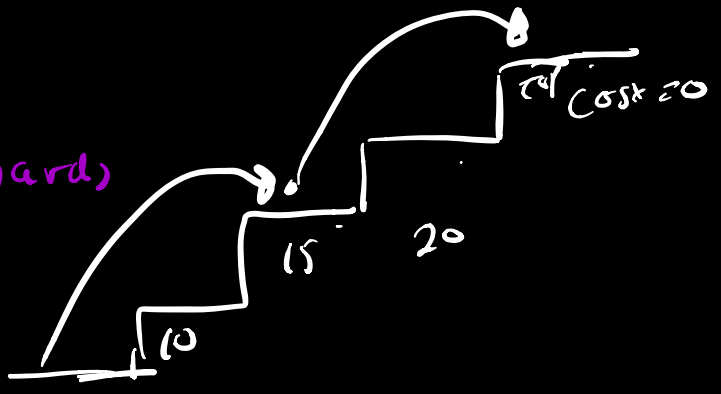
Min cost staircimbing



Cost = 15



1+1+1+1+1+1 = 6



working backwards

10 15 20 — n=3

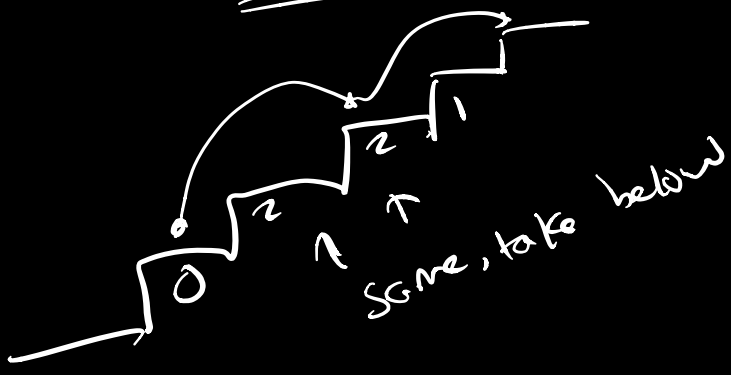
Start @ i=2

if (i==n) cost += 0

else

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

Code failed at case 0,2,2,1



2 ≠ 3

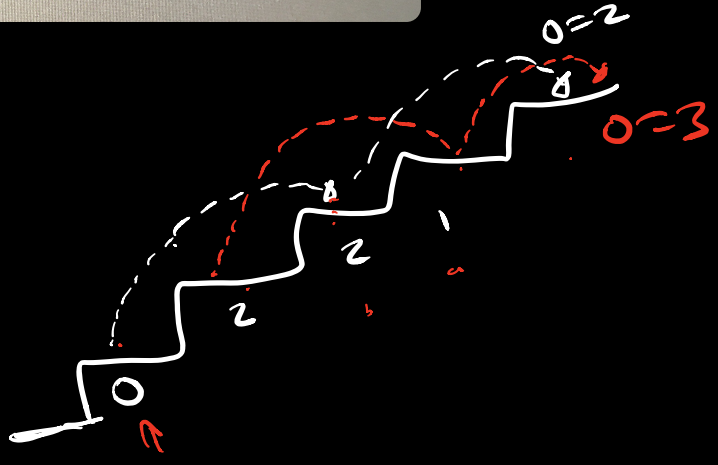
→ forgot you can start at step 0 or step 1

```
class Solution {
public:
    int minCostClimbingStairs(vector<int>& cost) {

        int n = cost.size();
        if (!n || n==1) return 0;
        int payAmount = 0;
        for(int i=n; i>=2;  ){
            int above = cost[i-1];
            int below = cost[i-2];

            if (above < below) {
                //take above
                i--;
                payAmount += above;
            } else {
                //take below esp. if equal
                i-= 2;
                payAmount += below;
            }
        }
        return payAmount;
    }
};
```

Starting from end didn't work.

because you can start at 0 or 1

242/276 passed



1 0 1 0

1 1 0 1

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

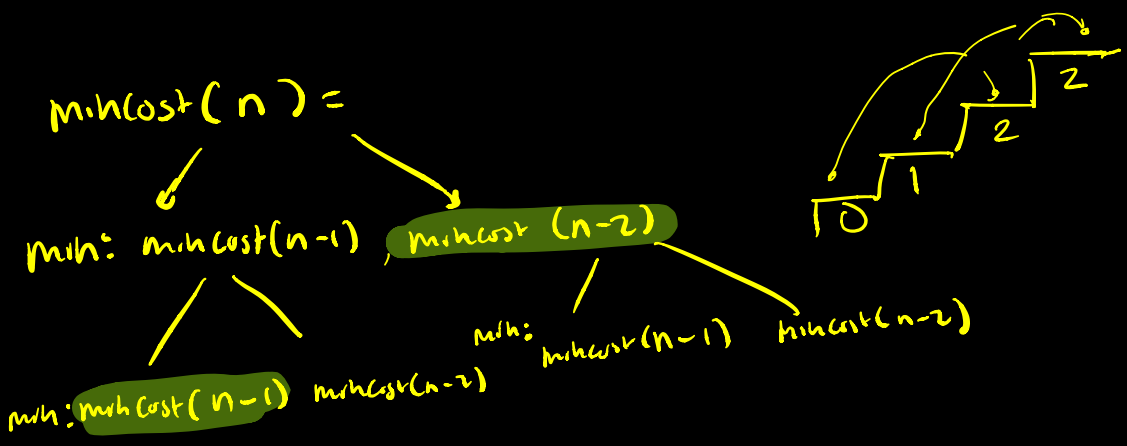
given stair i, the optimal cost to reach i is the cost of stair i plus the minimum between the optimal cost to reach i-1, and the optimal cost to reach i-2.

$$m[i] = cost[i] + \min(\text{optimal cost to reach } i-2, \text{optimal cost to reach } i-1)$$

base case $m[0] = A[0]$

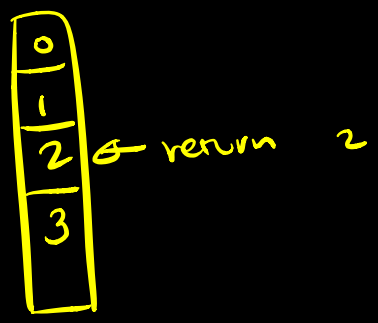
$m[1] = A[1]$

n = cost.size() - 1



do also minCost(n-1)

0 1 2 2




```
public:
    int minCostClimbingStairs(vector<int>& cost) {
        int dp[cost.size()]; //stores optimal cost to reach ith step

        dp[0] = cost[0];
        dp[1] = cost[1]; //base cases
        if (cost.size() < 2) return min(dp[0], dp[1]);
        for(int i=2; i < cost.size(); i++) {
            //store optimal cost to reach ith step in dp array
            dp[i] = cost[i] + min(dp[i-1], dp[i-2]);
        }
        return min(dp[cost.size()-1], dp[cost.size()-2]);
    }
};
```