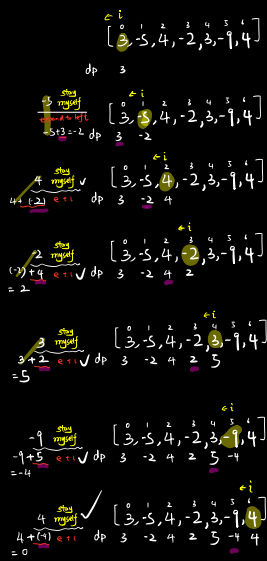
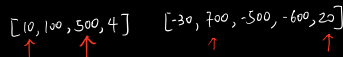


# maximum pre-sum of subarray dp



don't choose adjacent pair



dp[i] = 0 if can't choose p, max sum  
dp[i] = a[i] if can't choose p, max sum

dp[0] = arr[0] cur [3, 7] arr [7, 3]  
dp[1] = max(arr[0], arr[1]) dp [3, 7] dp [7, 3]

① arr[i]

dp[i] = dp[i-1] inherit what's best previous?

② arr[i]

dp[i] = { ① arr[i] (isolated)

② arr[i] + dp[i-2] because you can't choose dp[i-1]

dp[i] max pre-sum that end with

int[] arr; max pre-sum (arr) given arr  
if n == 1 return arr[0]  
if n == 2 return max(arr[0], arr[1])

dp[0]

dp[0] = arr[0]

dp[1] = max(arr[0], arr[1])

for (i = 2; i < n; i++) {

p1 = dp[i-1] // truly jmc arr[i]

p2 = arr[i]

p3 = arr[i] + dp[i-2] // previous best

dp[i] = max(p1, p2, p3)

}  
return dp[n-1]

7



Use few variable rolling update  
instead of dp

n == 1 return arr[0]  
n == 2 return max(arr[0], arr[1])

let last = arr[0] dp[0]  
let = max(arr[0], arr[1]) dp[1]

for (i = 2; i < n; i++) {

p1 = last

p2 = arr[i]

p3 = arr[i] + last

cur = max(p1, p2, p3)

last = cur

}  
return last



2560. House Robber IV

Medium

👁

🗨

🔒

🔖

198 House Robber III

213 House Robber II

837 House Robber III

🔖

🔖

🔖

🔖

There are several consecutive houses along a street, each of which has some money inside. There is also a robber, who wants to steal money from the houses, but he refuses to steal from adjacent houses.

The capability of the robber is the maximum amount of money he steals from one house of all the houses he robbed.

You are given an integer array `nums` representing how much money is stashed in each house. More formally, the `ith` house from the left has `nums[i]` dollars.

You are also given an integer `k`, representing the minimum number of houses the robber will steal from. It is always possible to steal at least `k` houses.

Return the minimum capability of the robber out of all the possible ways to steal at least `k` houses.

**Example 1:**

**Input:** `nums = [2,3,5,9]`, `k = 2`

**Output:** `5`

**Explanation:** There are three ways to rob at least 2 houses:

- Rob the houses at indices 0 and 2. Capability is `max(nums[0], nums[2]) = 5`.
- Rob the houses at indices 0 and 3. Capability is `max(nums[0], nums[3]) = 9`.
- Rob the houses at indices 1 and 3. Capability is `max(nums[1], nums[3]) = 9`.

Therefore, we return `min(5, 9, 9) = 5`.

**Example 2:**

**Input:** `nums = [2,7,9,3,1]`, `k = 2`

**Output:** `7`

**Explanation:** There are 7 ways to rob the houses. The way which leads to minimum capability is to rob the house at index 0 and 4. Return `max(nums[0], nums[4]) = 2`.

**Constraints:**

- `1 <= nums.length <= 105`
- `0 <= nums[i] <= 104`
- `1 <= k <= nums.length <= 105`

On [ \_ \_ \_ ] money in each  
at least steal k rooms  
≥ k rooms

minimum  
capability  
to steal

capacity [100, 300, 500, 50]

note cap, easier to achieve

assume f

bool f(x)

robber(nums, ability) {  
    // if not touching  
    adjacent rooms  
    how many rooms  
    carrying steal

l = 1  
r = max(nums)

if yes  
0 ~ 250 all can

if no  
251 ~ 500 → binary search

given cap, all ≥ x, don't check adjacent rooms

[ \_ \_ \_ ] how many # you can take more

int f(int l, int r)  
    // min length  
    if (r == l)  
        return ans[0] <= x ? 1 : 0  
    if (r == l + 1)  
        return (ans[0] <= x || ans[1] <= x) ? 1 : 0  
    dp[l] = ans[0] <= x ? 1 : 0  
    dp[l+1] = ans[0] <= x || ans[1] <= x ? 1 : 0  
    for (int i = l+2; i <= r; i++)  
        dp[i] = (ans[i-2] <= x || ans[i-1] <= x) ? 1 : 0  
    return dp[r]

do not check adjacent num

if ans[i] <= x  
    p1 > (p1, p2)  
    ans[i]

because p2 > 0  
p1 = 0 + dp[i-2] < dp[i-1]

return ans

minCap(nums, k) {  
    l = 1, r = 10  
    for (l, r = minCap) {  
        r = max(nums, r)  
    }  
    // ans = 0  
    while (l <= r) {  
        m = (l + r) / 2  
        if (rob(nums, m) >= k :  
            → ans = m ← eligible cap but may not be min  
            record ans as potential result  
        else : // not eligible, don't record ans, go to right side  
            l = m + 1  
    }  
    return ans

class Solution:

def minCapability(self, nums: List[int], k: int) -> int:

    r = max(nums)  
    ans = 0  
    while l <= r:  
        m = (l + r) // 2  
        if self.robber(nums, m) >= k:  
            ans = m  
            r = m - 1  
        else:  
            l = m + 1  
    return ans

def robber(self, nums: List[int], ability: int) -> int:

    lastLast = 1 if nums[0] <= ability else 0  
    if len(nums) == 1:  
        return lastLast  
    last = 1 if nums[0] <= ability or nums[1] <= ability else 0  
    ans = max(lastLast, last)  
    for i in range(2, len(nums)):  
        p1 = last  
        p2 = 0  
        if nums[i] <= ability:  
            p2 = lastLast + 1  
        cur = max(p1, p2)  
        ans = max(ans, cur)  
        lastLast = last  
        last = cur  
    return ans