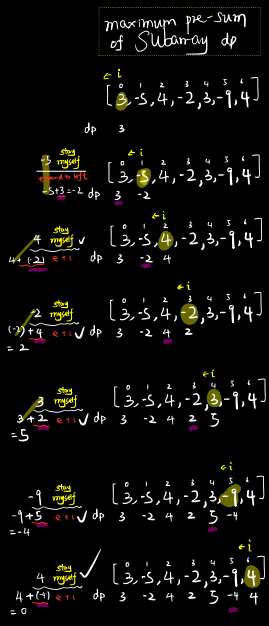
maximum pre-sum of Subarray dp



don't choose adjacent pair

[10, 100, 500, 4] [-30, 700, -500, -600, 20]

dp[0] 0-0 if can choose, max sum

dp[1] 0-1 if can choose, max sum

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

① arr[i] ✗

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

② arr[i] ✓

$$dp[i] = \begin{cases} ① \ arr[i] \ \text{(isolated)} \\ ② \ arr[i] + dp[i-2] \end{cases}$$

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

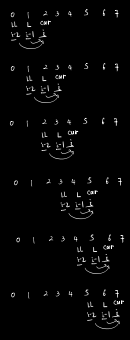
dp[] max presum that endwith

given arr

```
int[] maxpresum (arr){ , arr.size
    if n==1
        return arr[0]
    if n==2
        return max(arr[0], arr[1])
    dp[n]
    dp[0] = arr[0]
    dp[1] = max(arr[0], arr[1])
    for(i=2; i<n; i++){
        p1 = dp[i-1] // definitely ignore arr[i]
        p2 = arr[i]
        p3 = arr[i] + dp[i-2]   // previous best
        dp[i] = max(p1, p2, p3)
    }
    return dp[n-1]
}
```

Use few variable rolling update instead of dp

```
n==1
    return arr[0]
n==2
    return max(arr[0], arr[1])
lastlast = arr[0]   dp[0]
last = max(arr[0], arr[1])   dp[1]
for(i=2; i<n; i++){
    p1 = last
    p2 = arr[i]
    p3 = arr[i] + lastlast
    cur = max(p1, p2, p3)
    lastlast = last
    last = cur
}
return last
```



2560. House Robber IV

198 House Robber I
213 House Robber II
337 House Robber III

Medium · 699 · 17 · Companies

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 homes, but he **refuses to steal from adjacent homes**.

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 i^{th} 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:

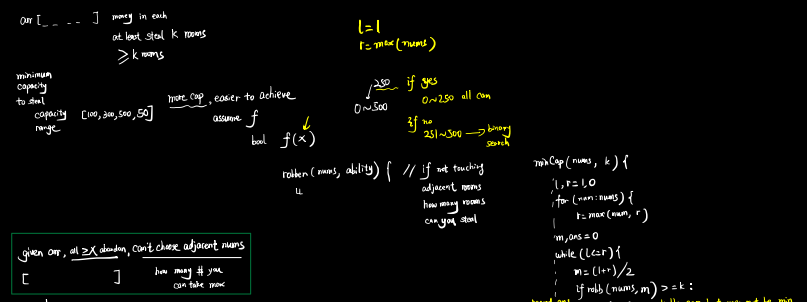
```
Input: nums = [2,3,5,9], k = 2
Output: 5
Explanation:
There are three ways to rob at least 2 houses:
- Rob the houses at indices 0 and 2. Capability is max(nums[0], nums[2]) = 5.
- Rob the houses at indices 0 and 3. Capability is max(nums[0], nums[3]) = 9.
- Rob the houses at indices 1 and 3. Capability is max(nums[1], nums[3]) = 9.
Therefore, we return min(5, 9, 9) = 5.
```

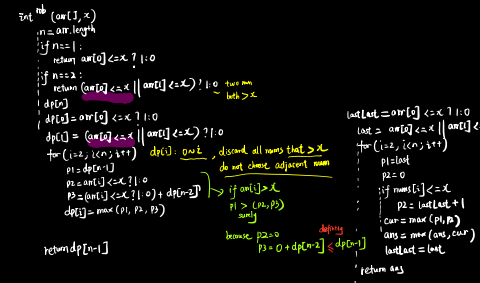
Example 2:

```
Input: nums = [2,7,9,3,1], k = 2
Output: 2
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 <= 10^5`
- `1 <= nums[i] <= 10^9`
- `1 <= k <= (nums.length + 1)/2`






```python
class Solution:
    def minCapability(self, nums: List[int], k: int) -> int:
        l = 1
        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
```