# **Maximum Sum of Non-Adjacent Elements | House Robber | 1-D | DP on Subsequences**

: [House robber problem- leetcode](https://leetcode.com/problems/house-robber/)

[Coding Ninjas link](https://www.codingninjas.com/codestudio/problems/maximum-sum-of-non-adjacent-elements_843261?source=youtube&campaign=striver_dp_videos&utm_source=youtube&utm_medium=affiliate&utm_campaign=striver_dp_videos&leftPanelTab=0)

**Based on PICK- NOT PICK approach (2^N)**

\*\***1. recursive- solution:**

**int helper(int ind, vector<int> &nums){**

**if(ind == 0) return nums[ind];**

**if(ind< 0) return 0;**

**//take**

**int take = nums[ind] + helper(ind-2, nums);**

**//not- take**

**int notTake = 0 + helper(ind-1, nums);**

**return max(take, notTake);**

**}**

**int maximumNonAdjacentSum(vector<int> &nums){**

**int n = nums.size();**

**return helper(n-1, nums);**

**}**

**TC =** O(2^N), SC = O(N) for recursive stack space

\*\***2. DP- memoization solution:**

**int helper(int ind, vector<int> &nums, vector<int> &dp){**

**if(ind == 0) return nums[ind];**

**if(ind< 0) return 0;**

**if(dp[ind] != -1) return dp[ind]; //return already stored value**

**//take**

**int take = nums[ind] + helper(ind-2, nums, dp);**

**//not- take**

**int notTake = 0 + helper(ind-1, nums, dp);**

**dp[ind] = max(take, notTake); //consider max one**

**return dp[ind];**

**}**

**int maximumNonAdjacentSum(vector<int> &nums){**

**int n = nums.size();**

**vector<int> dp(n, -1);**

**return helper(n-1, nums, dp);**

**}**

TC **=** O(2^N)

SC = O(N) + O(N) for dp[]

\*\***3. DP- tabulation:**

**//DP- tabulation(bottom- up approach)**

**class Solution {**

**public:**

**int rob(vector<int>& nums) {**

**int n = nums.size();**

**vector<int> dp(n, -1);**

**dp[0] = nums[0];**

**//array indexing is 0 based so considering <n**

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

**int pick = nums[i];**

**if(i>1) //because at index 1 we will go to dp[-1] and thats not allowed to do**

**pick += dp[i-2];**

**int notPick = 0 + dp[i-1];**

**//take whichever option is better**

**dp[i] = max(pick, notPick);**

**}**

**return dp[n-1];**

**}**

**};**

TC = O(N)

SC = O(N)

\***\*4. DP- optimized space:**

**int maximumNonAdjacentSum(vector<int> &nums){**

**int n = nums.size();**

**int prev1 = nums[0];**

**int prev2 = 0;**

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

**//take**

**int take = nums[i];**

**if(i>1)**

**take += prev2;**

**//not- take**

**int notTake = 0 + prev1;**

**int cur = max(take, notTake); //consider max one**

**prev2 = prev1;**

**prev1 = cur;**

**}**

**return prev1;**

**}**

TC = O(N)

SC = O(1)