Problem statement

> In this problem, you have given n size of array and you have to find the maximum sum of elements which are not adjacent.

Example:

Input:

```
1,2,3,9
```

Here maximum sum will be `10` which is `1+9` and both are adjacent elements. we also can take `1+3` but it's 4 and we want maximum sum so answer will be 10.

Approach:

⟶ So we will find all the possible subsequences for the array and then only take the adjacent elements in our sum.

we can find all possiblem subsequences using pick and don't pick method.

```
int pick = arr[idx] + solve(idx-2); // because we don't want to take adjacent element
int dontPick = 0 + solve(idx-1);
```

At last return the maximum of `pick` and `dontPick`

```
return max(pick,dontPick);
```

⟶ What if we are at `idx=0` then there is a chance we have took 2nd index because we can't take 1st index. So we will just simply return the arr[i] because we want to take the current element too in sum.

```
if(idx == 0){
        return arr[idx];
}
```

⟶ Now there will be some edge cases because when we do `idx-2` so there is a chance it becomes negative for indexes like `1` so to handle this we have to write one more condition

```
if(idx < 0){
        return 0;
}
```

Memoization Code

⟶ Here also we can store the answers in the dp array and use them again when we find any other same function call.

```
int solve(int idx,vector<int>&dp,vector<int>&arr){
        if(idx == 0){
                return arr[idx];
        }
        if(idx<0){
                return 0;
        }
        if(dp[idx] != -1){
                return  dp[idx];
        }
        int pick = arr[idx] + solve(idx-2);
        int dontPick = 0 + solve(idx-1);
        return dp[idx] = max(pick,dontPick);
}
```

Tabulation code

```
solve(vector<int>&arr){
        vector<int>dp(n,0);
        dp[0] = arr[0]; // Base case for i=0
        for(int i=1;i<n;i++){
                int pick = arr[i] ;
                if(i>1){ // second base case for negative index
                        pick += dp[i-2];
                }
                int dontPick = 0 + dp[i-1];
                dp[i] = max(pick,dontPick);
        }
        return dp[n-1];
}
```

Space optimisation

⟶ Here if we look closely then we can see that we just need 3 values: `dp[i],dp[i-1] and dp[i-2]` so why don't we store them in variables instead of a vector.

```
solve(vector<int>&arr){
        int prev = arr[0],prev2=0;
        for(int i=1;i<n;i++){
                int pick = arr[i] ;
                if(i>1){ // second base case for negative index
                        pick += prev2;
                }
                int dontPick = 0 + prev;
                int curr = max(pick,dontPick);
                prev2 = prev;
                prev = curr;
        }
        return prev;
}
```

Complexity

```
Time complexity: O(n)
Space complexity: O(1)
```

Similar problem:

> Mr. X is a professional robber planning to rob houses along a street. Each house has a certain amount of money hidden. All houses along this street are arranged in a circle. That means the first house is the neighbor of the last one. Meanwhile, adjacent houses have a security system connected, and it will automatically contact the police if two adjacent houses were broken into on the same night.
> You are given an array/list of non-negative integers 'ARR' representing the amount of money of each house. Your task is to return the maximum amount of money Mr. X can rob tonight without alerting the police.

Approach:

⟶ So here they said that the First and Last elements are adjacent so here we can solve the question in 2 ways.

```
1. Leave the first element and run the previous question logic and find answer
2. Leave the last element and run the previous question logic and find answer
```

⟶ And our final answer will be maximum Of these both answers

Code:

```
long long int findAdjacentSum(vector<int>&arr){
        int n = arr.size();
    long long int prev = arr[0],prev2=0;
        for(int i=1;i<n;i++){
```

```cpp
                long long int pick = arr[i] ;
                if(i>1){ // second base case for negative index
                        pick += prev2;
                }
                long long int dontPick = 0 + prev;
                long long int curr = max(pick,dontPick);
                prev2 = prev;
                prev = curr;
        }
        return prev;
}
long long int houseRobber(vector<int>& arr)
{
        int n = arr.size();
        if(n == 1){ // If array have single element
                return arr[0];
        }
        vector<int>temp1(n),temp2(n);
        for(int i=0;i<n;i++){
                if(i!=0){ // Leaving first element and adding
remaining elements in temp1
                        temp1.push_back(arr[i]);
                }
                if(i!=n-1){ // Leaving second element and adding
remaining elements in temp2
                        temp2.push_back(arr[i]);
                }
        }
        return max(findAdjacentSum(temp1),findAdjacentSum(temp2));
}
```