Maximum sum of non-adjacent elements / HOUSE ROBBER

Given an array/list of 'N' integers. You are supposed to return the maximum sum of the subsequence with the constraint that no two elements are adjacent in the given array/list.

So! 1. Let's try out all subsequences with the given constraint
2. Pick the one with the maximum sum.

Pick subsequent with no adjacent element

- Elements in my array will be my indexes.

2 main points

1) Pick : Take this element into your subsequence.
2) notpick : I wont take this element in my subsequence.

1. Express recur in index
2. Do stuffs on the index
3. Return the best.

f(ind) ⇒ max sum

In the provided question at any ind position you can either pick or notpick the element. You'll pick the element if you haven't picked the previous element. You'll not pick an element if you have picked the previous element (since it'll become adjacent element). These are the stuffs you can do

∴ pick = a[ind] + f(ind-2);
(picking the element) (bcz you cannot pick the prev element)

notpick = 0 + f(ind-1)
(notpicking the curr element) (you can pick prev element bcz you are not picking the current element)

At the end return the best

return max(pick, notpick);

```
f(ind) {
 if (ind == 0) return a[ind];
 if (ind < 0) return 0;
 pick = a[ind] + f(ind-2);
notpick = 0 + f(ind);
return max(pick, notpick))
}
```

Since our solution has overlapping subproblems then it indicates that we can optimize it.

Memoization

1) declare dp array & initialize it to -1
2) Use the array to store the already computed recursive values & to fetch them:

```
f(ind) {
 if (ind == 0) return a[ind];
 if (ind < 0) return 0;
 if (dp[ind] != -1) return dp[ind];
 pick = a[ind] + f(ind-2);
notpick = 0 + f(ind-1);
return dp[ind] = max(pick, notpick);
}
```

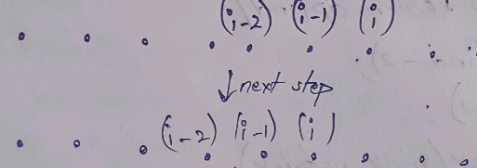
Tabulation (Bottom - Up)

1) Initialize DP array
2) Write all Base Case in the form of dp
3) Use array to store value instead of doing recursion

```
dp[0] = a[0];
for(int i=1; i<n; i++){
  pick = a[ind];
  if(i>1) pick += dp[i-2];
  notpick = 0 + dp[i-1];
  dp[i] = max(pick, notpick);
```

Space Optimization

Instead of using array you store the value in variable here



At any position since you only need $(i-1)$ & $(i-2)$ You store the $(i-1)$ & $(i-2)$ value in variables & try to update these variable instead of using an array

```
prev2 = prev1
prev1 = curr
```

Code

```
int prev1 = a[0];
int prev2 = 0;
for(int i=1; i<n; i++){
    int pick = a[i]
    if (i>1) pick += prev2;
    int notpick = 0 + prev1;
    int curr = max(pick, notpick);
    prev2 = prev1;
    prev1 = curr;
}
return prev;
}
```