

arr[] = (2, 3, 2)
 1. 0 - robbed
 2. 0 - not robbed

1. [2, a[0]]
 don't steal
 $dp[i][0] = \max(dp[i+1][0], dp[i+1][1])$
 $dp[i][1] = \max(dp[i+1][1], a[i] + dp[i+1][0])$
 ↑
 steal

2. [1, n-1]

The idea is to fill the dp table based on previous values. For each index, we either include it or exclude it to compute the maximum value. The table is filled in an iterative manner from $j=2$ to $j=n-1$.

$$dp[j] = \max(arr[j] + dp[j-2], dp[j-1])$$

The idea is to split the circular house robbery problem into two linear subproblems by considering two scenarios: one where we exclude the last house (array[0 ... n-2]) and another where we exclude the first house (array[1 ... n-1]).

```

C++ (g++ 5.4)
Start Timer
1 // Driver Code Ends
2
3 class Solution {
4 public:
5     // Tabulation approach to find the maximum
6     // value. Here x and y are the starting and
7     // ending indices of the subarray.
8     int maxValTab(int x, int y, vector<int> &arr) {
9         int n = arr.size();
10
11         vector<int> dp(n);
12
13         // For first house, taking is only option
14         dp[x] = arr[x];
15
16         // For second house, we can either take
17         // from 1st or 2nd.
18         dp[x+1] = max(arr[x], arr[x+1]);
19
20         for (int j=x+2; j<=y; j++) {
21             int take = arr[j] + dp[j-2];
22             int noTake = dp[j-1];
23             dp[j] = max(take, noTake);
24         }
25
26         return dp[y];
27     }
28
29     // Function to calculate the maximum stolen value
30     int maxVal(vector<int> &arr) {
31         int n = arr.size();
32
33         // Base cases
34         if (n == 0) return 0;
35         if (n == 1) return arr[0];
36
37         int ans = 0;
38
39         // Skipping last house
40         ans = max(ans, maxValTab(0, n-2, arr));
41
42         // Skipping first house
43         ans = max(ans, maxValTab(1, n-1, arr));
44
45         return ans;
46     }
47 };
48
49 // Driver Code Ends

```