Given an array of positive numbers, find the maximum sum of a subsequence with the constraint that no 2 numbers in the sequence should be adjacent in the array.
5, 5, 10, 100, 10, 5 -> 110 5,100, 5
1, 2, 3 -> 4
5, 10, 100, 5
elem of subsequence is not adjacent in array ?
input: arr, repeated? Y order? N, range of value ? 0< .<100,000
output: int
corner case： input: None?Y []?Y only one elem? Y

idea: brute force
method:

1. for loop to scan elem of arr, find all possible combination of arr, get sum of them tO(2**N), sO(2**N)
2. find max value of these sum of them tO(2**N), sO(1)

time O(2**N) space O(2**N)

idea: dynamic program method
method:

1. dp[i]: until arr[i], the max value of subsequence
2. dp[i] = max(dp[i -1], dp[i - 2] + arr[i])
3. dp[0] = 5, dp[1] = 5

time O(n**2) space O(N)
5, 5, 10, 100, 10, 5 -> 110 -> 5,100, 5
dp[0] = 5
dp[1] = 5
dp[2] = max(dp[1], dp[0] + arr[2]) = max(5, 5+ 10) = 15
dp[3] = max(dp[2], dp[1] + arr[3]) = max(15, 5 + 100) = 105
dp[4] = max(dp[3], dp[2] + arr[4]) = max (105, 15 + 10) = 105
dp[5] = max(dp[4], dp[3] + arr[5]) = max (105, 105 + 5) = 110

```
class MySolution:
        def findMax(self, arr):  # find the not adjacent  biggest sum, output: int
                if not arr:  # corner case
                        return -1
                if len(arr) == 1:
                        return arr[0]

                dp = [0] * len(arr)
                dp[0] = arr[0]
                dp[1] = arr[1]
                for i in range(2, len(arr)):  # dp method to find max value
        dp[i] = max(dp[i - 1], dp[i - 2] + arr[i])
    return dp[-1]
```

test:

test case: None -> - 1 only one elem -> arr[0]

test case: `[2,7,9,3,1]`

```
dp[0] = 2
dp[1] = 7
dp[2] = max(dp[1], dp[0] + arr[2]) = max(7, 11) = 11
dp[3] = max(dp[2], dp[1] + arr[3]) = max(11, 7 + 3) = 11
dp[4] = max(dp[3], dp[2] + arr[4]) = max(11, 11 + 1) = 12
return dp[-1] = 12
```