Maximum sum such that no two elements are adjacent

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. So 3 2 7 10 should return 13 (sum of 3 and 10) or 3 2 5 10 7 should return 15 (sum of 3, 5 and 7). Answer the question in most efficient way.

Examples :

```
Input : arr[] = {5, 5, 10, 100, 10, 5}
Output : 110

Input : arr[] = {1, 2, 3}
Output : 4

Input : arr[] = {1, 20, 3}
Output : 20
```

We strongly recommend that you click here and practice it, before moving on to the solution.

Algorithm:

Loop for all elements in arr[] and maintain two sums incl and excl where incl = Max sum including the previous element and excl = Max sum excluding the previous element.

Max sum excluding the current element will be max(incl, excl) and max sum including the current element will be excl + current element (Note that only excl is considered because elements cannot be adjacent).

At the end of the loop return max of incl and excl.

Example:

```
  arr[] = {5,  5, 10, 40, 50, 35}

  inc = 5
  exc = 0

  For i = 1 (current element is 5)
  incl =  (excl + arr[i])  = 5
  excl =  max(5, 0) = 5

  For i = 2 (current element is 10)
  incl =  (excl + arr[i]) = 15
  excl =  max(5, 5) = 5

  For i = 3 (current element is 40)
  incl = (excl + arr[i]) = 45
  excl = max(5, 15) = 15

  For i = 4 (current element is 50)
  incl = (excl + arr[i]) = 65
  excl =  max(45, 15) = 45

  For i = 5 (current element is 35)
  incl =  (excl + arr[i]) = 80
  excl = max(5, 15) = 65

And 35 is the last element. So, answer is max(incl, excl) =  80
```

Thanks to Debanjan for providing code.

Implementation:

C/C++ Java Python

```
#include<stdio.h>

/*Function to return max sum such that no two elements
 are adjacent */
int FindMaxSum(int arr[], int n)
{
  int incl = arr[0];
  int excl = 0;
  int excl_new;
  int i;

  for (i = 1; i < n; i++)
  {
     /* current max excluding i */
     excl_new = (incl > excl)? incl: excl;

     /* current max including i */
     incl = excl + arr[i];
     excl = excl_new;
  }

   /* return max of incl and excl */
   return ((incl > excl)? incl : excl);
}

/* Driver program to test above function */
int main()
{
  int arr[] = {5, 5, 10, 100, 10, 5};
  int n = sizeof(arr) / sizeof(arr[0]);
  printf("%d \n", FindMaxSum(arr, 6));
  return 0;
}
```

Run on IDE

Output:

```
110
```

Time Complexity: O(n)

Now try the same problem for array with negative numbers also.