Maximum subarray sum

Idea: fill the array R[] by determining max sub-array ending at each index i and then determine the maximum of R[] to determine the overall max sub-array sum or you can keep the track of overall max sub array sum in the same loop. We are using the later one. Main condition **R[i] = max (R[i-1]+A[i], A[i]) i.e., maximum of (**max so far plus current index value or only the current index value**)**

```
Max_SubArr_Sum(A[], N)
{
   m_sum = A[1]  // To keep the track of overall maximum sub-array sum of the given input array.
   R[1...N] // maximum sub-array ending at each index.
   R[1] = A[1] //max sub-array ending at index#1 will be A[1]

   for(i=2 to N)
   {
//max sub-array ending at index i will be equals to max(R[i-1]+A[i], A[i])
      R[i] = max(R[i-1]+A[i], A[i]);
      m_sum = max(R[i], m_sum);  //update overall maximum
   }
   return m_sum
}
```

Sample Input:

N = 8

Arr[]	-2	-3	4	-1	-2	1	5	-3

==========

Dry Run:

Hard Coded Initialization:

m_sum = -2 (to store overall maximum sub-array sum)

R[]	-2							

==========

i = 2

R	-2	-3						

m_sum = -2 (previous value retained)

==========

i = 3

R	-2	-3	4					

m_sum = 4 (value updated)

==========

i = 4

R	-2	-3	4	3				

m_sum = 4 (value updated)

==========

i = 5

R	-2	-3	4	3	1			

m_sum = 4 (previous value retained)

==========

i = 6

R	-2	-3	4	3	1	2		

m_sum = 4 (previous value retained)

==========

i = 7

R	-2	-3	4	3	1	2	7	

m_sum = 7 (value updated)

==========

i = 8

R	-2	-3	4	3	1	2	7	4

m_sum = 7 (previous value retained)

Overall max sub-array sum = 7

Time Complexity: **O(N)**

Space Complexity: **O(N)**