Maximum subarray problem

```
#include<stdio.h>
int maxSubArraySum(int a[], int size)
{
   int max_so_far = 0, max_ending_here = 0;
   int i;
   for(i = 0; i < size; i++)
   {
     max_ending_here = max_ending_here + a[i];
     if(max_ending_here < 0)
        max_ending_here = 0;
     if(max_so_far < max_ending_here)
        max_so_far = max_ending_here;
    }
    return max_so_far;
}

/*Driver program to test maxSubArraySum*/
int main()
{
   int a[] = {-2, -3, 4, -1, -2, 1, 5, -3};
   int max_sum = maxSubArraySum(a, 8);
   printf("Maximum contiguous sum is %d\n", max_sum);
   getchar();
   return 0;
}
```

Notes:
Algorithm doesn't work for all negative numbers. It simply returns 0 if all numbers are negative. For handling this we can add an extra phase before actual implementation. The phase will look if all numbers are negative, if they are it will return maximum of them (or smallest in terms of absolute value). There may be other ways to handle it though.

Above program can be optimized further, if we compare max_so_far with max_ending_here only if max_ending_here is greater than 0.

```
int maxSubArraySum(int a[], int size)
{
   int max_so_far = 0, max_ending_here = 0;
   int i;
   for(i = 0; i < size; i++)
   {
```

```
     max_ending_here = max_ending_here + a[i];
     if(max_ending_here < 0)
         max_ending_here = 0;

     /* Do not compare for all elements. Compare only
        when  max_ending_here > 0 */
     else if (max_so_far < max_ending_here)
         max_so_far = max_ending_here;
   }
   return max_so_far;
}
```

Time Complexity: O(n)
Algorithmic Paradigm: Dynamic Programming

```
int maxSubSum3( int [ ] a )
    {
        int maxSum = 0;
        int thisSum = 0;
        int i=0;
        int j=0;

        while (j < a.length)
        {
            thisSum =thisSum + a[ j ];

            if( thisSum > maxSum )
            {
                maxSum = thisSum;
                seqStart = i;
                seqEnd  = j;
            }
            else if( thisSum < 0 )
            {
                i = j + 1;
                thisSum = 0;
            }
            j=j+1;
        }

        return maxSum;
    }
```