# Find the size of given array

int arr[] = {12, 3, 4, 15};

int n = sizeof(arr) / sizeof(arr[0]);

# Matrix Multiplication

void multiply(int mat1[][N],

              int mat2[][N],

              int res[][N])

{

    int i, j, k;

    for (i = 0; i < N; i++)

    {

        for (j = 0; j < N; j++)

        {

            res[i][j] = 0;

            for (k = 0; k < N; k++)

                res[i][j] += mat1[i][k] \*

                             mat2[k][j];

        }

    }

}

# [How does XOR work in C to find a number occurring odd number of times?](https://stackoverflow.com/questions/30808046/how-does-xor-work-in-c-to-find-a-number-occurring-odd-number-of-times)

This code does not count the number of occurrences of an odd number. Instead, it finds a single number in an array that occurs an odd number of times.

Your test array has these numbers:

2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2

Their counts are as follows:

2 - 4 times

3 - 2 times

4 - 4 times

5 - 3 times

Only 5 is listed an odd number of times.

XOR has these two properties:

Y ^ 0 = Y

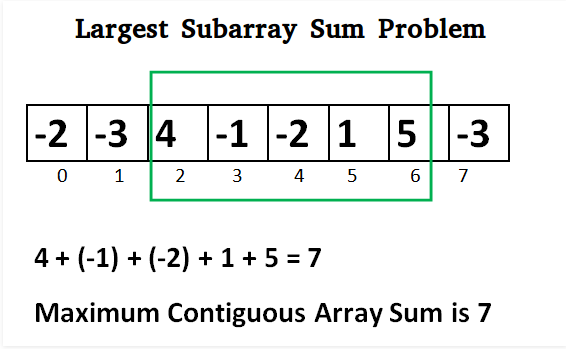
X ^ X ^ Y = Y

for any value of X and Y. In other words, XOR-ing any number Y with zero leaves the value unchanged, and XOR-ing a number X twice with any value Y leaves that original value unchanged. The order of operations does not matter. Since res starts at zero, XOR-ing together all numbers from your array produce 5 - the only value that is not XOR-ed an even number of times.

This trick would not work if there were multiple or no numbers that occur an odd number of times.

# Largest Sum Contiguous Subarray

Write an efficient program to find the sum of contiguous subarray within a one-dimensional array of numbers which has the largest sum.



## Kadane’s Algorithm:

Initialize:

max\_so\_far = 0

max\_ending\_here = 0

Loop for each element of the array

(a) max\_ending\_here = max\_ending\_here + a[i]

(b) if(max\_ending\_here < 0)

max\_ending\_here = 0

(c) if(max\_so\_far < max\_ending\_here)

max\_so\_far = max\_ending\_here

return max\_so\_far

**Explanation:**  
Simple idea of the Kadane’s algorithm is to look for all positive contiguous segments of the array (max\_ending\_here is used for this). And keep track of maximum sum contiguous segment among all positive segments (max\_so\_far is used for this). Each time we get a positive sum compare it with max\_so\_far and update max\_so\_far if it is greater than max\_so\_far

Lets take the example:

{-2, -3, 4, -1, -2, 1, 5, -3}

max\_so\_far = max\_ending\_here = 0

for i=0, a[0] = -2

max\_ending\_here = max\_ending\_here + (-2)

Set max\_ending\_here = 0 because max\_ending\_here < 0

for i=1, a[1] = -3

max\_ending\_here = max\_ending\_here + (-3)

Set max\_ending\_here = 0 because max\_ending\_here < 0

for i=2, a[2] = 4

max\_ending\_here = max\_ending\_here + (4)

max\_ending\_here = 4

max\_so\_far is updated to 4 because max\_ending\_here greater

than max\_so\_far which was 0 till now

for i=3, a[3] = -1

max\_ending\_here = max\_ending\_here + (-1)

max\_ending\_here = 3

for i=4, a[4] = -2

max\_ending\_here = max\_ending\_here + (-2)

max\_ending\_here = 1

for i=5, a[5] = 1

max\_ending\_here = max\_ending\_here + (1)

max\_ending\_here = 2

for i=6, a[6] = 5

max\_ending\_here = max\_ending\_here + (5)

max\_ending\_here = 7

max\_so\_far is updated to 7 because max\_ending\_here is

greater than max\_so\_far

for i=7, a[7] = -3

max\_ending\_here = max\_ending\_here + (-3)

max\_ending\_here = 4

## Kadane’s algo Program:

|  |
| --- |
| // C++ program to print largest contiguous array sum  #include<iostream>  #include<climits>  using namespace std;    int maxSubArraySum(int a[], int size)  {      int max\_so\_far = INT\_MIN, max\_ending\_here = 0;        for (int i = 0; i < size; i++)      {          max\_ending\_here = max\_ending\_here + a[i];          if (max\_so\_far < max\_ending\_here)              max\_so\_far = max\_ending\_here;            if (max\_ending\_here < 0)              max\_ending\_here = 0;      }      return max\_so\_far;  }    /\*Driver program to test maxSubArraySum\*/  int main()  {      int a[] = {-2, -3, 4, -1, -2, 1, 5, -3};      int n = sizeof(a)/sizeof(a[0]);      int max\_sum = maxSubArraySum(a, n);      cout << "Maximum contiguous sum is " << max\_sum;      return 0;  } |

**Output:**

Maximum contiguous sum is 7

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;     for (int 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

Following is another simple implementation. The implementation handles below case:

## When all numbers in array are negative.

|  |
| --- |
| #include<iostream>  using namespace std;    int maxSubArraySum(int a[], int size)  {     int max\_so\_far = a[0];     int curr\_max = a[0];       for (int i = 1; i < size; i++)     {          curr\_max = max(a[i], curr\_max+a[i]);          max\_so\_far = max(max\_so\_far, curr\_max);     }     return max\_so\_far;  }    /\* Driver program to test maxSubArraySum \*/  int main()  {     int a[] =  {-2, -3, 4, -1, -2, 1, 5, -3};     int n = sizeof(a)/sizeof(a[0]);     int max\_sum = maxSubArraySum(a, n);     cout << "Maximum contiguous sum is " << max\_sum;     return 0;  } |

**Output:**

Maximum contiguous sum is 7

## To print the subarray with the maximum sum,

We maintain indices whenever we get the maximum sum.

|  |
| --- |
| // C++ program to print largest contiguous array sum  #include<iostream>  #include<climits>  using namespace std;    int maxSubArraySum(int a[], int size)  {      int max\_so\_far = INT\_MIN, max\_ending\_here = 0,         start =0, end = 0, s=0;        for (int i=0; i< size; i++ )      {          max\_ending\_here += a[i];            if (max\_so\_far < max\_ending\_here)          {              max\_so\_far = max\_ending\_here;              start = s;              end = i;          }            if (max\_ending\_here < 0)          {              max\_ending\_here = 0;              s = i + 1;          }      }      cout << "Maximum contiguous sum is "          << max\_so\_far << endl;      cout << "Starting index "<< start          << endl << "Ending index "<< end << endl;  }    /\*Driver program to test maxSubArraySum\*/  int main()  {      int a[] = {-2, -3, 4, -1, -2, 1, 5, -3};      int n = sizeof(a)/sizeof(a[0]);      int max\_sum = maxSubArraySum(a, n);      return 0;  } |

**Output:**

Maximum contiguous sum is 7

Starting index 2

Ending index 6

Now try below question  
Given an array of integers (possibly some of the elements negative), write a C program to find out the \*maximum product\* possible by multiplying ‘n’ consecutive integers in the array where n <= ARRAY\_SIZE. Also print the starting point of maximum product subarray.