Divide and Conquer | Set 3 (Maximum Subarray Sum)

You are given a one dimensional array that may contain both positive and negative integers, find the sum of contiguous subarray of numbers which has the largest sum.

For example, if the given array is {-2, -5, **6, -2, -3, 1, 5**, -6}, then the maximum subarray sum is 7 (see highlighted elements).

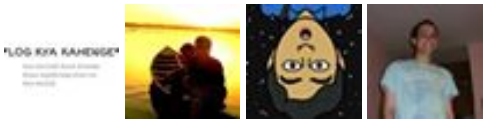
The naive method is to run two loops. The outer loop picks the beginning element, the inner loop finds the maximum possible sum with first element picked by outer loop and compares this maximum with the overall maximum. Finally return the overall maximum. The time complexity of the Naive method is O(n^2).

Using **Divide and Conquer** approach, we can find the maximum subarray sum in O(nLogn) time. Following is the Divide and Conquer algorithm.

1) Divide the given array in two halves
2) Return the maximum of following three
….**a)** Maximum subarray sum in left half (Make a recursive call)
….**b)** Maximum subarray sum in right half (Make a recursive call)
….**c)** Maximum subarray sum such that the subarray crosses the midpoint

The lines 2.a and 2.b are simple recursive calls. How to find maximum subarray sum such that the subarray crosses the midpoint? We can easily find the crossing sum in linear time. The idea is simple, find the maximum sum starting from mid point and ending at some point on left of mid, then find the maximum sum starting from mid + 1 and ending with sum point on right of mid + 1. Finally, combine the two and return.

```
// A Divide and Conquer based program for maximum subarray sum problem
```

Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion

```
#include <stdio.h>
#include <limits.h>

// A utility funtion to find maximum of two integers
int max(int a, int b) { return (a > b)? a : b; }

// A utility funtion to find maximum of three integers
int max(int a, int b, int c) { return max(max(a, b), c); }

// Find the maximum possible sum in arr[] auch that arr[m] is part of
int maxCrossingSum(int arr[], int l, int m, int h)
{
    // Include elements on left of mid.
    int sum = 0;
    int left_sum = INT_MIN;
    for (int i = m; i >= l; i--)
    {
        sum = sum + arr[i];
        if (sum > left_sum)
          left_sum = sum;
    }

    // Include elements on right of mid
    sum = 0;
    int right_sum = INT_MIN;
    for (int i = m+1; i <= h; i++)
    {
        sum = sum + arr[i];
        if (sum > right_sum)
          right_sum = sum;
    }

    // Return sum of elements on left and right of mid
    return left_sum + right_sum;
}

// Returns sum of maxium sum subarray in aa[l..h]
int maxSubArraySum(int arr[], int l, int h)
{
   // Base Case: Only one element
   if (l == h)
     return arr[l];

   // Find middle point
   int m = (l + h)/2;

   /* Return maximum of following three possible cases
```


Popular Posts

```
      a) Maximum subarray sum in left half
      b) Maximum subarray sum in right half
      c) Maximum subarray sum such that the subarray crosses the midpo
   return max(maxSubArraySum(arr, l, m),
              maxSubArraySum(arr, m+1, h),
              maxCrossingSum(arr, l, m, h));
}

/*Driver program to test maxSubArraySum*/
int main()
{
   int arr[] = {2, 3, 4, 5, 7};
   int n = sizeof(arr)/sizeof(arr[0]);
   int max_sum = maxSubArraySum(arr, 0, n-1);
   printf("Maximum contiguous sum is %d\n", max_sum);
   getchar();
   return 0;
}
```

Time Complexity: maxSubArraySum() is a recursive method and time complexity can be expressed as following recurrence relation.
T(n) = 2T(n/2) + $\Theta(n)$
The above recurrence is similar to Merge Sort and can be solved either using Recurrence Tree method or Master method. It falls in case II of Master Method and solution of the recurrence is $\Theta(nLogn)$.

The Kadane's Algorithm for this problem takes O(n) time. Therefore the Kadane's algorithm is better than the Divide and Conquer approach, but this problem can be considered as a good example to show power of Divide and Conquer. The above simple approach where we divide the array in two halves, reduces the time complexity from O(n^2) to O(nLogn).

References:
Introduction to Algorithms by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Tpoics:

- Remove minimum elements from either side such that 2*min becomes more than max
- Divide and Conquer | Set 6 (Search in a Row-wise and Column-wise Sorted 2D Array)
- Bucket Sort
- Kth smallest element in a row-wise and column-wise sorted 2D array | Set 1
- Find the number of zeroes
- Find if there is a subarray with 0 sum
- Divide and Conquer | Set 5 (Strassen's Matrix Multiplication)
- Count all possible groups of size 2 or 3 that have sum as multiple of 3

Writing code in comment? Please use **ideone.com** and share the link here.

36 Comments GeeksforGeeks

Sort by Newest

Recent Comments

Aman Hi, Why arent we checking for conditions...

Write a C program to Delete a Tree. · 11 minutes ago

kzs please provide solution for the problem...

Backtracking | Set 2 (Rat in a Maze) · 15 minutes ago

Sanjay Agarwal bool tree::Root_to_leaf_path_given_sum(tree...

Root to leaf path sum equal to a given number · 40 minutes ago

GOPI GOPINATH @admin Highlight this sentence "We can easily...

Count trailing zeroes in factorial of a number · 41 minutes ago

newCoder3006 If the array contains negative numbers also. We...

Find subarray with given sum · 1 hour ago

newCoder3006 Code without using while loop. We can do it...

Find subarray with given sum · 1 hour ago

Join the discussion…

Sameer Sharma • 3 months ago

it supposed to be the maximum array
output:
Here is the array with the maximum sum
22 + 500 + -67 + 1 = 456

Reply • Share ›

Sameer Sharma • 3 months ago

hey, can some one help me please!!!

I have a problem where i have to implement 2D Array so i can get output of 45

Here's my CODE:

```
#include <iostream>

using namespace std;

// constants

#define NUM_ARRAYS 5

#define NUM_ELEMS 4

// function prototypes (fill in any missing formal parameter lists)

int CalcArraySum(int array[], int numItems);

void DispArray(int array[], int numItems);
```

see more

Reply • Share ›

Aggie • 4 months ago

This problem can be solved in O(n) time.
the code is as following:

int maxSubArray(int A[], int n) {

int result,i,cur;

result=INT_MIN;

cur=0;

for(i=0;i<n;i++) {="" cur="cur+A[i];" if(cur="">result) result=cur;

if(cur<0) cur=0;

}

return result;

}

1 ▲ | ▼ • Reply • Share ›

Nakul • 5 months ago

This program can't be done in O(n) time. And if done then there will be cases
efficient one is O(nlogn) algorithm.

▲ | ▼ • Reply • Share ›

prashant saxena ➦ Nakul • 5 months ago

Check the link mentioned in the text about Kadane's Algorithm. I think it

3 ▲ | ▼ • Reply • Share ›

prashant saxena • 5 months ago

depending on the indices taken to calculate left+Right.

Reply • Share ›

Kartik ➚ prashant saxena • 5 months ago

This algo is taken from CLRS book. So can't be wrong, there may be s

Reply • Share ›

prashant saxena ➚ Kartik • 5 months ago

Ah.. yeah looks correct.. the case I mentioned will be taken car
crossingmax function... my bad. Thanks for correcting me

Reply • Share ›

Faisal • 7 months ago

The Algorithm you provide here, is not correct. Test the following case:
[-2, -5, 6, -2, 3, -10, 5, -6] :)

Reply • Share ›

Test • 8 months ago

with the method of Dynamic programming

==========

```
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#define TRUE 1

void main()

{
```

int *A=NULL;

int n=0;

int i.:

see more

Reply • Share ›

mrn • 8 months ago

int max=0,prev=a[0];
for(int i=1;i<n;i++) {="" if(prev+a[i]<0)="" prev="0;" else{="" prev="prev+a[i];" if(
}="">

Reply • Share ›

numid • a year ago

I think it can be done in a batter way,please point out any invalid test case if an

```
 /*You are given a one dimensional array that may contain both positi
find the sum of contiguous subarray of numbers which has the largest s

#include<iostream>

using namespace std;

int main()
{
    int arr[]={-2,-5,6,-2,-3,1,-10,-20,5,9,-7,-8,8,14};
    int i,size,ptr,qtr,sum,prev,diff;
    size=sizeof(arr)/sizeof(arr[0]);
    ptr=0;
    qtr=size-1;
```

```
    while(arr[ptr]<=0)
```

see more

• Reply • Share ›

Nakul numid • 5 months ago

When the input is {-2,-5,6} the output is shown by above program is 0 \

• Reply • Share ›

subhasish • a year ago

It can be done in O(n)

The time comlexity O(nLogn) is only because of the O(n) part in the recursion time complexity O(n).

The idea is to propagate the maximum sum from left end, right end to top.

So the method will return a structure of four elements :
l : max sum from left
r : max sum from right
m : max mum
t : total sum

For base case (single element) every thing will be equal to that element

Now combine step will be as follows :

```
 t = L.t + R.t
l = ((L.t+R.l)>L.l) ?(L.t+R.l) : L.l;
r = ((R.t+L.r)>R.r) ?(R.t+L.r) : R.r;
m = max(L.m, R.m, L.r+R.l);
```

So the combine step becomes O(1), overall time complexity becomes O(n)

• Reply • Share ›

abhishek08aug • a year ago

Intelligent :D

```
/* Paste your code here (You may delete these lines if not writing c
```

1 • Reply • Share ›

ravisingh3531 • a year ago

The O(n) solution for above problem without divide and concurs

```
#include <stdio.h>
#define SIZE 9
int main()
{
        int array[SIZE] ={1, 2, 3, -4, -5, -6, 7, 8, 9};
        int index;
        int finalsum = 0;
        int sum=0,temp;
        int flag = 0;
        for(index=0;index<SIZE;index++)
        {
                if(flag == 0)
                {
                        if(array[index] > 0)
                        {
                                sum = sum + array[index];
```

see more

Nakul ravisingh3531 • 5 months ago

if the input array is {-2,-5,6} the output should be 6 but it shows 0 whicl

Reply • Share ›

Sandeep Jain • a year ago

Thanks for pointing this out. We have corrected it now.

1 Reply • Share ›

Sandeep Jain • a year ago

Thanks for pointing this out. We have corrected it now.

1 Reply • Share ›

Suraj Kaushal • a year ago

In this algorithm you have written return "minimum" of three possible cases in

1 Reply • Share ›

Sandip • a year ago

You could reduce the time to merge the solution of sub problems to constant t overall algorithm to O(n).

For that you would need to additionally return 1) Optimal solution including star solution including end element in subarray.

Here is the C# code.

[sourcecode language="C#"]
class Program
{
class Solution
{
internal int StartIndex { get; set; }

internal int EndIndex { get; set; }
internal int Sum { get; set; }

internal void CopyValues(Solution s)
{

see more

2 ▲ | ▼ • Reply • Share ›

Saroj Smart • a year ago

As one new visitor would learn this as an efficient algorithm for max sum suba
separate post. This can be posted as a complementary approach to the origin
When I saw this as an update from Geeksforgeeks today, I expected, someon
Kadane&#039s algorithm.

▲ | ▼ • Reply • Share ›

rspr • a year ago

find the O(n) solution for the above problem:

```
 /*-2 -5 6 -2 -3 1 5 -6
-2 -5 6 4  1  2 7 1
sum = 7
*/
#include<stdio.h>
int A[100];
int find_largest_sum(int A[],int n_elements){

        int largest=A[0];
        int tmp=A[0];
        for(int i=1;i<n_elements;++i)
        {
                if(tmp+A[i] > A[i])
```

```
            else
                    tmp=A[i];
```

see more

Reply • Share ›

GM ➚ rspr • a year ago

I think u missed to update largest in your code.

```
if(tmp+A[i] > A[i])
            tmp+=A[i];
        else
            if(tmp>largest)
                largest = tmp;
            tmp=A[i];
```

Reply • Share ›

Ravi Chandra • a year ago

My suggestion is re-post the same problem with the updated solution.

Reply • Share ›

GeeksforGeeks ➚ Ravi Chandra • a year ago

@Ravi Chandra: The O(n) solution is already published here. The mai
and simple example of Divide and Conquer.

Reply • Share ›

Sai Nikhil • a year ago

D&C is O(nlg(n)), where as kadane(DP) is just O(n), so why use former inste

1 Reply • Share ›

Oh I did not have idea about Kadan&#039e algorithm. Thanks for letting me kn

1 ^ | v • Reply • Share ›

GeeksforGeeks • a year ago

Nipun: This implementation doesn&#039t seem naive :) It seems to be implem discussed here (http://www.geeksforgeeks.org/l...). Please let us know if you

1 ^ | v • Reply • Share ›

Nipun Agarwal • a year ago

```
int CalMaxSum(int *arr, int length)
{
int maxSum = arr[0];
int localSum = arr[0];

for(int i=1; i<length; i++)
{
localSum += arr[i];
if(localSum > maxSum)
{
maxSum = localSum;
}
else if(localSum < 0)
{
localSum = 0;
if(arr[i] > maxSum)
maxSum = arr[i];
continue;
```

see more

^ | v • Reply • Share ›

GeeksforGeeks • a year ago

The problem is same, this solution is different from the solution present at http
solution discussed there is Kadane&#039s algorithm and has time complexity

1 ^ | v • Reply • Share ›

Amarnath Raju Vysyaraju • a year ago

http://www.geeksforgeeks.org/l...

Same isn&#039t it?

^ | v • Reply • Share ›

GeeksforGeeks • a year ago

Nipun: Thanks for sharing your thoughts. Could you provide a working code of

1 ^ | v • Reply • Share ›

Aditya • a year ago

We can solve it with the complicity of 'n' !

n log n is expensive and big head to deal

^ | v • Reply • Share ›

Nipun Agarwal • a year ago

why do we need 2 loops in naive approach. we can easily find it in O(n) time.
taking the above example -2, -5, 6, -2, -3, 1, 5, -6.

Let maxsum = a[0] initially.
Now maxsum = -2.
for(i=1;i<n;i++)
//if negative number do not add and go on until u find a positive number.
keep on adding number until the sum > 0 and also keep the max sum.

1 ^ | v • Reply • Share ›

Siva Krishna • a year ago

I think max_cross_sum is not the correct way.Let us take the initial array as [

cross sum it is 30(6(left)+24(right)) but here the sum came from left:[1 2 3]anc crossing sum...

[source language="C"][/source]
/* Paste your code here (You may delete these lines if not writing code) */
[/source code]

• Reply • Share ›

Subscribe Add Disqus to your site