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
#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 it
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
      a) Maximum subarray sum in left half
      b) Maximum subarray sum in right half
      c) Maximum subarray sum such that the subarray crosses the midpoint */
   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) + [Tex]\Theta(n)[/Tex]
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 [Tex]\Theta(nLogn)[/Tex].

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 Topics:

- Find Union and Intersection of two unsorted arrays

- Pythagorean Triplet in an array
- Maximum profit by buying and selling a share at most twice
- Design a data structure that supports insert, delete, search and getRandom in constant time
- Print missing elements that lie in range 0 – 99
- Iterative Merge Sort
- Group multiple occurrence of array elements ordered by first occurrence
- Given a sorted and rotated array, find if there is a pair with a given sum

Tags: Divide and Conquer

Tweet g+1 3

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

49 Comments GeeksforGeeks 1 Login

Recommend 1 Share Sort by Newest

Join the discussion…

Hengameh · 3 days ago

Actually, the time complexity of naive method is "O(n^3)"; since it should check the mentioned cases for all the sub-arrays, from size 1 till size of the array;
So, you missed one outer loop, which should be for the "sub-arrays". And if you add that, the time complexity of naive method will be "O(n^3)".

""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).""

Reply · Share ›

Vaibhav Raj · 3 months ago

Here is another O(n) solution in java:

```
import java.util.HashMap;

public class MaximumSubArray {

private static int[] sumEndingHere(final int[] arr){

int sumEndingHere = 0;

int[] sumArray = new int[arr.length];

for(int i = 0; i< arr.length; i++){

sumEndingHere += arr[i];
```

sumArray[i] = sumEndingHere;

}

return sumArray;

see more

1 ^ | v • Reply • Share ›

its_dark • 6 months ago

@GeeksForGeeks:

The time complexity can be reduced to O(n), by returning two extra values.
> sum starting from the first element
> sum ending at the last element

below is the code, with comments.

```
struct node{
    int left,mx,right,whole;
    node(int l, int m, int r, int w):left(l), right(r), mx(m), whole(w){}
};

class Solution {
public:
    int maxSubArray(int a[], int n) {
        node p = util(0,n-1,a);
        return p.mx;
    }
```

see more

^ | v • Reply • Share ›

Aditya Goel • 7 months ago

@GeeksforGeeks
please provide link of this approach in Maximum Subarray Sum main pain as well

^ | v • Reply • Share ›

Som Bose • 8 months ago

This solution is wrong. We need to at each step have a not of what was the interval in which the largest sum was found.

^ | v • Reply • Share ›

<HoldOnLife!#> • 9 months ago

isnt it largest sum contagious problem done using kadane and dp ?

• Reply • Share ›

Charan • 10 months ago

Can you please check once for input {13, 14, -6, -2, -3, -1, -5, -6};

• Reply • Share ›

Klove • a year ago

For an inputArray[ ] = { -2, -3, 4, -1 }, why does the algorithm return 4. Isn't it suppose to return 3 ? (4 + (-1) = 3)

• Reply • Share ›

Vivek Klove • a year ago

4 is a sub-array of size of 1 , and with max sum. so output is correct

• Reply • Share ›

Shirsh Zibbu • a year ago

@Anil Gupta
The main purpose of this problem in the book stated above was to provide an example for div-&-con algo, rather than providing the best possible solution.

• Reply • Share ›

Anil Gupta • a year ago

This can be done in bigO(n) using Kedane's Algorithm. Refer to http://en.wikipedia.org/wiki/M...

• Reply • Share ›

Sameer Sharma • a year 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 • a year 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 456

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

1 ^ | ⌄ • Reply • Share ›

arjomanD ↱ Sameer Sharma • a year ago
You haven't defined your functions as i see
you have only defined their prototypes
^ | ⌄ • Reply • Share ›

Aggie • a year 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;

}
3 ^ | ⌄ • Reply • Share ›

Nakul • a year ago
This program can't be done in O(n) time. And if done then there will be cases on which the algorithm fails. The efficient one is O(nlogn) algorithm.
^ | ⌄ • Reply • Share ›

prashant saxena ↱ Nakul • a year ago
Check the link mentioned in the text about Kadane's Algorithm. I think its done correctly

in o(n).

4 • Reply • Share ›

prashant saxena • a year ago

Yes the algo is wrong. You can't apply divide and conquer here. Your Max sum could as well be Left_MAx+Right_MAx depending on the indices taken to calculate left+Right.

• Reply • Share ›

Kartik prashant saxena • a year ago

This algo is taken from CLRS book. So can't be wrong, there may be some slight confusion though.

• Reply • Share ›

prashant saxena Kartik • a year ago

Ah.. yeah looks correct.. the case I mentioned will be taken care by non recursive function call crossingmax function... my bad. Thanks for correcting me

• Reply • Share ›

Faisal • 2 years ago

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

• Reply • Share ›

A Faisal • 6 months ago

output is 7. Correct isn't it??

• Reply • Share ›

Test • 2 years 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 • 2 years 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(max<="" prev)="" max="prev;" }="" }="">

^ | ⌄ • Reply • Share ›

numid • 2 years ago

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

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

#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 • a year ago

When the input is {-2,-5,6} the output is shown by above program is 0 whereas it should be 6.

^ | ⌄ • Reply • Share ›

subhasish • 2 years ago

It can be done in O(n)

The time comlexity O(nLogn) is only because of the O(n) part in the recursion. If it were O(1) then we would get total 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)

1 ^ | v • Reply • Share ›

abhishek08aug • 2 years ago

Intelligent :D

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

3 ^ | v • Reply • Share ›

ravisingh3531 • 2 years 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

2 ^ | v • Reply • Share ›

Nakul ravisingh3531 • a year ago

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

1 ^ | v • Reply • Share ›

Sandeep Jain • 2 years ago

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

2 ^ | v • Reply • Share ›

Sandeep Jain • 2 years ago

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

1 ^ | v • Reply • Share ›

Suraj Kaushal • 2 years ago

In this algorithm you have written return "minimum" of three possible cases in step 2. It should be maximum.

1 ^ | v • Reply • Share ›

Sandip • 2 years ago

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

For that you would need to additionally return 1) Optimal solution including starting element in subarray, 2) Optimal 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

3 • Reply • Share ›

Saroj Smart • 2 years ago

As one new visitor would learn this as an efficient algorithm for max sum subarray, I discourage putting this as a separate post. This can be posted as a complementary approach to the original post, but no way as separate post. When I saw this as an update from Geeksforgeeks today, I expected, someone might have a better algorithm than Kadane&#039s algorithm.

• Reply • Share ›

rspr • 2 years 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])
                        tmp+=A[i];
                else
                        tmp=A[i];
```

see more

• Reply • Share ›

GM ➦ rspr • 2 years 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 • 2 years ago

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

^ | ˅ • Reply • Share ›

GeeksforGeeks ➦ Ravi Chandra • 2 years ago

@Ravi Chandra: The O(n) solution is already published here. The main idea of this post is to show a good and simple example of Divide and Conquer.

^ | ˅ • Reply • Share ›

Sai Nikhil • 2 years ago

D&C is O(nlg(n)), where as kadane(DP) is just O(n), so why use former instead of later?

1 ^ | ˅ • Reply • Share ›

Nipun Agarwal • 2 years ago

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

1 ^ | ˅ • Reply • Share ›

GeeksforGeeks • 2 years ago

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

2 ^ | ˅ • Reply • Share ›

Nipun Agarwal • 2 years 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

⌃ | ⌄ • Reply • Share ›

GeeksforGeeks • 2 years ago

The problem is same, this solution is different from the solution present at http://www.geeksforgeeks.org/l... . The solution discussed there is Kadane&#039s algorithm and has time complexity O(n).

2 ⌃ | ⌄ • Reply • Share ›

Amarnath Raju Vysyaraju • 2 years ago

http://www.geeksforgeeks.org/l...
Same isn&#039t it?

⌃ | ⌄ • Reply • Share ›

GeeksforGeeks • 2 years ago

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

1 ⌃ | ⌄ • Reply • Share ›

Aditya • 2 years ago

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

n log n is expensive and big head to deal

⌃ | ⌄ • Reply • Share ›

Nipun Agarwal • 2 years 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 ⌃ | ⌄ • Reply • Share ›

Siva Krishna • 2 years ago

I think max_cross_sum is not the correct way.Let us take the initial array as [1 2 3 -4 -5 -6 7 8 9] now consider the cross sum it is 30(6(left)+24(right)) but here the sum came from left:[1 2 3]and right:[7 8 9] which was not the 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 Privacy DISQUS

Google™ Custom Search

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

- **Popular Posts**

    - All permutations of a given string
    - Memory Layout of C Programs
    - Understanding “extern” keyword in C
    - Median of two sorted arrays
    - Tree traversal without recursion and without stack!
    - Structure Member Alignment, Padding and Data Packing

  - Intersection point of two Linked Lists
  - Lowest Common Ancestor in a BST.
  - Check if a binary tree is BST or not
  - Sorted Linked List to Balanced BST
- Follow @GeeksforGeeks

- ## Recent Comments

  - Nikhil kumar

    public class...

    Print missing elements that lie in range 0 – 99 · 5 minutes ago
  - Ashish Aggarwal

    Try Data Structures and Algorithms Made Easy -...

    Algorithms · 27 minutes ago
  - Vlad

    Thanks. Very interesting lectures.

    Expected Number of Trials until Success · 1 hour ago
  - cfh

    My implementation which prints the index of the...

    Longest Even Length Substring such that Sum of First and Second Half is same · 1 hour ago
  - Gaurav pruthi

    forgot to see that part ;)

    Bloomberg Interview | Set 1 (Phone Interview) · 2 hours ago
  - saeid aslami

    thanks

    Greedy Algorithms | Set 7 (Dijkstra's shortest path algorithm) · 2 hours ago
-