Tug of War

Given a set of n integers, divide the set in two subsets of n/2 sizes each such that the difference of the sum of two subsets is as minimum as possible. If n is even, then sizes of two subsets must be strictly n/2 and if n is odd, then size of one subset must be (n-1)/2 and size of other subset must be (n+1)/2.

For example, let given set be {3, 4, 5, -3, 100, 1, 89, 54, 23, 20}, the size of set is 10. Output for this set should be {4, 100, 1, 23, 20} and {3, 5, -3, 89, 54}. Both output subsets are of size 5 and sum of elements in both subsets is same (148 and 148).
Let us consider another example where n is odd. Let given set be {23, 45, -34, 12, 0, 98, -99, 4, 189, -1, 4}. The output subsets should be {45, -34, 12, 98, -1} and {23, 0, -99, 4, 189, 4}. The sums of elements in two subsets are 120 and 121 respectively.

The following solution tries every possible subset of half size. If one subset of half size is formed, the remaining elements form the other subset. We initialize current set as empty and one by one build it. There are two possibilities for every element, either it is part of current set, or it is part of the remaining elements (other subset). We consider both possibilities for every element. When the size of current set becomes n/2, we check whether this solutions is better than the best solution available so far. If it is, then we update the best solution.

Following is C++ implementation for Tug of War problem. It prints the required arrays.

```
#include <iostream>
#include <stdlib.h>
#include <limits.h>
using namespace std;

// function that tries every possible solution by calling itself recursively
void TOWUtil(int* arr, int n, bool* curr_elements, int no_of_selected_elements,
             bool* soln, int* min_diff, int sum, int curr_sum, int curr_position)
{
    // checks whether the it is going out of bound
    if (curr_position == n)
        return;

    // checks that the numbers of elements left are not less than the
    // number of elements required to form the solution
    if ((n/2 - no_of_selected_elements) > (n - curr_position))
        return;

    // consider the cases when current element is not included in the solution
    TOWUtil(arr, n, curr_elements, no_of_selected_elements,
              soln, min_diff, sum, curr_sum, curr_position+1);

    // add the current element to the solution
    no_of_selected_elements++;
    curr_sum = curr_sum + arr[curr_position];
    curr_elements[curr_position] = true;

    // checks if a solution is formed
    if (no_of_selected_elements == n/2)
    {
        // checks if the solution formed is better than the best solution so far
        if (abs(sum/2 - curr_sum) < *min_diff)
        {
            *min_diff = abs(sum/2 - curr_sum);
            for (int i = 0; i<n; i++)
                soln[i] = curr_elements[i];
        }
    }
    else
    {
        // consider the cases where current element is included in the solution
        TOWUtil(arr, n, curr_elements, no_of_selected_elements, soln,
                  min_diff, sum, curr_sum, curr_position+1);
    }

    // removes current element before returning to the caller of this function
    curr_elements[curr_position] = false;
}

// main function that generate an arr
void tugOfWar(int *arr, int n)
```

```
{
    // the boolen array that contains the inclusion and exclusion of an element
    // in current set. The number excluded automatically form the other set
    bool* curr_elements = new bool[n];

    // The inclusion/exclusion array for final solution
    bool* soln = new bool[n];

    int min_diff = INT_MAX;

    int sum = 0;
    for (int i=0; i<n; i++)
    {
        sum += arr[i];
        curr_elements[i] =  soln[i] = false;
    }

    // Find the solution using recursive function TOWUtil()
    TOWUtil(arr, n, curr_elements, 0, soln, &min_diff, sum, 0, 0);

    // Print the solution
    cout << "The first subset is: ";
    for (int i=0; i<n; i++)
    {
        if (soln[i] == true)
            cout << arr[i] << " ";
    }
    cout << "\nThe second subset is: ";
    for (int i=0; i<n; i++)
    {
        if (soln[i] == false)
            cout << arr[i] << " ";
    }
}

// Driver program to test above functions
int main()
{
    int arr[] = {23, 45, -34, 12, 0, 98, -99, 4, 189, -1, 4};
    int n = sizeof(arr)/sizeof(arr[0]);
    tugOfWar(arr, n);
    return 0;
}
```

Output:

```
The first subset is: 45 -34 12 98 -1
The second subset is: 23 0 -99 4 189 4
```

This article is compiled by Ashish Anand and reviewed by GeeksforGeeks team. 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: Backtracking

Tweet 8+1 2

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

87 Comments GeeksforGeeks 1 Login

Recommend 1 Share Sort by Newest

Join the discussion…

xxhantian · 2 months ago

I think there is a dp solution to this problem
1) if n is odd, add a “0” element to the set and makes n even
2) since there could be negative integers in the set, find the smallest , say a
3) all the elements add a positive integer |a| (it won't influnce the result since step 1) ensures that both subset add the same amount)
4) sum up all the elements in the set, say result is "r" ,
5) difine size = （r+1）/2 , then this problem is converted to a knapsack problem with knapsack size “size”

Reply • Share ›

rverma · 5 months ago

This is O(nlogn) solution .Please correct it if u find any error

http://ideone.com/o1Jd0e

Reply • Share ›

Aditya Goel → rverma · 3 months ago

comments would have been nice in this long program.

1 Reply • Share ›

rihansh · 5 months ago

http://ideone.com/ZbzEfo

Reply • Share ›

Guest · 6 months ago

#include <stdio.h>

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

void printSubset(int i,int W[],int set_size)
{
int j;
for(j=0;j<set_size;j++) {="" if(1<<j="" &="" i)="" printf("%d="" ",w[j]="" );="" }="" printf("\n");=""
}="" void="" printsubsets(int="" w[],int="" set_size,int="" p)="" {="" int=""
pow_size="pow(2,set_size);" int="" i,j,c="0;" int="" total_sum="0,temp=0;" int=""
mindiff="INT_MAX;" for(i="0;i&lt;set_size;i++)" total_sum+="W[i];" for(i="0;i&lt;pow_size;i++)"
{="" int="" t="0;c=0;" for(j="0;j&lt;set_size;j++)" {="" if(i="" &="" 1<<j="" ){="" t="t+W[j];" c++;=""
}="" }="" if(c="=p){" int="" diff="total_sum" -="" t="" ;="" if(fabs(t-diff)="" <="" mindiff){=""
mindiff="fabs(t-diff);" temp="i;" }="" }="" }="" printf("the="" min="" difference="" possible="" is=""
:="" %d\n",="" mindiff);="" printf("one="" of="" the="" half="" subset="" is:="" ");=""
printsubset(temp,w,set_size);="" }="" *="" driver="" program="" to="" test="" above=""
functions="" *="" int="" main()="" {="" int="" w[]="{23," 45,="" -34,="" 12,="" 0,="" 98,="" -99,=""
4,="" 189,="" -1,="" 4};="" int="" t;="" if(sizeof(w)="" sizeof(int)%2="=" 0)=""
t="sizeof(W)/sizeof(int)/2;" else="" t="(sizeof(W)/sizeof(int)-1)/2;" printsubsets(w,sizeof(w)=""
sizeof(int),t);="" getchar();="" return="" 0;="" }="">
```

^ | ⌄ • Reply • Share ›

Guest • 6 months ago

```
//This one is easy to understand code :
// This will give the min difference possible and one of the subsets formed
// author Abhishek Jaiswal

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

void printSubset(int i,int W[],int set_size)
{
int j;
for(j=0;j<set_size;j++) {="" if(1<<j="" &="" i)="" printf("%d="" ",w[j]="" );="" }="" printf("\n");=""
}="" void="" printsubsets(int="" w[],int="" set_size,int="" p)="" {="" int=""
pow_size="pow(2,set_size);" int="" i,j,c="0;" int="" total_sum="0,temp=0;" int=""
mindiff="INT_MAX;" for(i="0;i&lt;set_size;i++)" total_sum+="W[i];" for(i="0;i&lt;pow_size;i++)"
{="" int="" t="0;c=0;" for(j="0;j&lt;set_size;j++)" {="" if(i="" &="" 1<<j="" ){="" t="t+W[j];" c++;=""
}="" }="" if(c="=p){" int="" diff="total_sum" -="" t="" ;="" if(fabs(t-diff)="" <="" mindiff){=""
mindiff="fabs(t-diff);" temp="i;" }="" }="" }="" printf("the="" min="" difference="" possible="" is=""
:="" %d\n",="" mindiff);="" printf("one="" of="" the="" half="" subset="" is:="" ");=""
printsubset(temp,w,set_size);="" }="" *="" driver="" program="" to="" test="" above=""
functions="" *="" int="" main()="" {="" int="" w[]="{23," 45,="" -34,="" 12,="" 0,="" 98,="" -99,=""
4,="" 189,="" -1,="" 4};="" int="" t;="" if(sizeof(w)="" sizeof(int)%2="=" 0)=""
```

t="sizeof(W)/sizeof(int)/2;" else="" t="(sizeof(W)/sizeof(int)-1)/2;" printsubsets(w,sizeof(w)="" sizeof(int),t);="" getchar();="" return="" 0;="" }="">

Reply • Share ›

helper • 6 months ago

this is exponential :(coz it is recursive

Reply • Share ›

Vishal Patil • 7 months ago

A python solution to the problem

https://github.com/vishpat/Pra...

Reply • Share ›

ankush → Vishal Patil • 5 months ago

you don't need to calculate s2 (line 15-17). the diff (line 18) can be calculated as

diff = (total_sum - 2 * sum(s1))

total_sum is sum(numbers) which can be calculated outside the loop

Reply • Share ›

GuojiaAgain • 7 months ago

Can we improve it with modified subset sum ? Since the total sum is static, and there are totally two groups, so what we need to do is to select n/2 numbers and make it sum up to half of the sum as close as possible. Since there are totally two groups so, one must larger or equal half sum and another is opposite.

Reply • Share ›

Puneet • 8 months ago

memory optimized solution where we don't need extra array to store the results. We simply keep track of the minimum subset difference. Below is java code for the same:

```
package com.ds.algo.msos;

public class MinSumSubSetsWithoutLibraryOptimized {

int arraySize = 1;
int arrayCount = 0;
int min = 10000000;

public static void main(String[] args) {

MinSumSubSetsWithoutLibraryOptimized minSumOfSubsets = new
MinSumSubSetsWithoutLibraryOptimized();
int[] inputArray = { 1, 777, 1, 65, 3, 4 };
int result = minSumOfSubsets.getMinSumOfSubsets(inputArray);
System.out.println(result);
}
```

see more

^ | ⌄ • Reply • Share ›

Puneet • 8 months ago

We can get all possible subsets. We can compute this value for all possible subsets. value = (sum of complete array - sum of distinct subsets) - sum of distinct subsets in an array. The minimum value of this result array is the result: Here is java code for the same:

```
package com.ds.algo.msos;

public class MinSumSubSetsWithoutLibrary {

int arraySize = 1;
int arrayCount = 0;
int[] list;

public static void main(String[] args) {

MinSumSubSetsWithoutLibrary minSumOfSubsets = new MinSumSubSetsWithoutLibrary();

int[] inputArray = { 1, 777, 772, 65 ,3 , 4};
int result = minSumOfSubsets.getMinSumOfSubsets(inputArray);
System.out.println(result);
```

see more

^ | ⌄ • Reply • Share ›

Guest • 8 months ago

To much unnecessary computation. By marking
curr_elements[curr_position] = false;
Code is computing same values again. As someone mentioned in the comments, we can definitely use DP to store the bottom results while taking calling the second recursion(include the element) for a particular index.

^ | ⌄ • Reply • Share ›

Jon Snow • 9 months ago

How to get the minimum difference? If we simply print min_diff after printing both subset I don't think it works. For example for the array {1,2,3,4,5} minimum difference is 1 for subset {1,2,5} {3,4}. But if we print min_diff it prints zero. Any idea?

^ | ⌄ • Reply • Share ›

Chakshu Jon Snow • 8 months ago

you know nothin Jon Snow :D :P

3 • Reply • Share ›

The Big Idiot • 10 months ago

What is the complexity of the above solution ? is it O(2^n)??

2 • Reply • Share ›

Suvodip Bhattacharya • 10 months ago

```
#include<iostream>
#include<climits>
#include <algorithm>
using namespace std;
int barr[100];

void Half_Set_Util(int arr[],int barr[],int n,int curr,int half_value,int c_sum,int carr)
{
if(curr == n/2 || carr<0) return;

if(c_sum+arr[carr] < half_value && curr < n/2) {
barr[curr]=arr[carr];
c_sum+=arr[carr];
arr[carr]=INT_MIN;
Half_Set_Util(arr,barr,n,curr+1,half_value,c_sum,carr-1); }

else if(c_sum+arr[carr] == half_value && curr == n/2-1) {
barr[curr]=arr[carr];
```

see more

• Reply • Share ›

AlienOnEarth • a year ago

Another Solution can be to use Subset sum algorithm to find sum/2. Once we get the set for subset sum for sum/2, rest of the elements are in another subset.

• Reply • Share ›

Coder011 AlienOnEarth • a year ago

consider this case :
int a[]={1,1,1,1,1,1,1,7};
subset sum will print true, whereas acc. to question size of two subsets should equal n/2 i.e 4

• Reply • Share ›

AlienOnEarth Coder011 • a year ago

Yes correct. But as per the question, it is implicit that the solution {n/2} and {n/2} or {n/2} and {n/2+1} already exists. But this subset algorithm may not work in the situation where there are multiple subsets with the same sum. In that can, we will have to check explicitly whether subset size is n/2 or n/2+1.

^ | ⌄ • Reply • Share ›

minhaz ➦ AlienOnEarth • a year ago

Hey Hi,
I think Coder011 has Valid Point. We cannot directly apply the solution you r suggesting. We have to have an additional logic to make sure that the half sum is produced by exactly half number of the elements. Not by less than half elements.

But really good suggestion by u.

^ | ⌄ • Reply • Share ›

Coder011 ➦ AlienOnEarth • a year ago

Not getting ur line of thought, how is it implicit that the solution already exists?? and please clarify what is "sum". If it is the sum of the entire array, then how are you justifying the line :

"Once we get the set for subset sum for sum/2, rest of the elements are in another subset."

If you think that the ur solution fails when **there are multiple subsets with the same sum** , then what will be answer for this case:

int a[]={1,2,4,8,16,31}
where no two subsets of size n/2 have the same sum?

^ | ⌄ • Reply • Share ›

AlienOnEarth • a year ago

This problem is called balanced partitioning problem. It is NP-Complete. So even if we use DP, we can only verify whether such subsets exits. This solution is present on Geeksforgeeks.

http://www.geeksforgeeks.org/d...

There is no known solution which can solve this problem in polynomial time.

3 ^ | ⌄ • Reply • Share ›

mani992 • a year ago

1 more thing can be done to remove recursion:
1> suppose there are n elements,then iterate over 0->n
2>in each iteration ,
suppose i=1 and N=4

Now, get the bit at the positions from j=0 to j=3 in i,
if the bit=1 then include that arr[j] in the set1

else
include that in set2
you can get the bit by using : (i&(1<<j)> 0)
3>calculate the diff at the end of each iteration and compare the result. Hope I was clear enough.

• Reply • Share ›

mani992 • a year ago

1 more solution could be for +ve numbers only in O(nlogn)
1> find the total sum of the array elements.
2>let median=total_sum/2
3>sort the array in descending order
4> put the element in the first set if the sum(first set)<=median
else
put it in 2nd set.
5>iterate over all numbers once

• Reply • Share ›

Rajan Chaudhary • a year ago

is their any other solution instead of compiling every possible set, can we use dynamic programming to optimize it

• Reply • Share ›

AlienOnEarth Rajan Chaudhary • a year ago

This problem is called balanced partitioning problem. It is NP-Complete. So even if we use DP, we can only verify whether such subsets exits. This solution is present on Geeksforgeeks.

http://www.geeksforgeeks.org/d...

There is no known solution which can solve this problem in polynomial time.

1 • Reply • Share ›

Guest • a year ago

Please explain this
// removes current element before returning to the caller of this function
curr_elements[curr_position] = false;

1 • Reply • Share ›

toughtimes Guest • a year ago

same doubt is it necessary??

• Reply • Share ›

np toughtimes • 10 months ago

it is necessary because when you pass an array you pass its address so if we set as true it will be true for every call means we are setting the true at that memory location, after sometime we will find all the array elements filled with true... So after completion of that element we are making value as false.....So it will not be reflected in other calls

• Reply • Share ›

Qiang Zhang • a year ago

Can we do something like the followings:
0. we maintain the sum of current left array and right array, which is initialized as 0;
1. sort the array nlogn;
2. for each pair of adjacent number, we add the larger one the array has smaller sum;
3. continues this, until we finish.

1 • Reply • Share ›

Guest Qiang Zhang • 6 months ago

Step 1 is good. We can sort it in nlogn (In descending order right?!)
Step 2: we can start picking up the element and put it in of the either result set that results in smaller difference of the sum.
E.g. if LeftSum>RightSum then add the element to right result array or to left result array.
Time complexity would be n^2 logn.

• Reply • Share ›

Guest Qiang Zhang • a year ago

consider this array 1000,10,9,8,7,6,5,4,3

• Reply • Share ›

Sriharsha g.r.v • a year ago

how about this...
1:sort it in o(nlogn) time....
2:place alternate elements in two different arrays..
it forms the answer....

• Reply • Share ›

vamshi Sriharsha g.r.v • a year ago

doesnt work for 1, 2, 10, 11

5 • Reply • Share ›

renu • 2 years ago

can someone pls tell me what is the time complexity of the above code.
in microsoft interview the problem was asked to be solved in o(n) time complexity without using sorting .can it be really done?

Reply • Share ›

rahul → renu • 2 years ago

Complexity is exponential n it is NP hard.So no polynomial 0(n) solution is possible for this.It cant be done in 0(n) with n without sorting...U have to check all possible combinations:).

1 • Reply • Share ›

Gautam • 2 years ago

//Guys i try using Backtracking......
#include<stdio.h>
#include<math.h>
int size=11;//If you r intrested then change the size
int arr1[size]={0};
int Final[size];
int MinDiff=100;//we just assume the MinDiff it may b more
void FindSubset(int Input[],int Index,int RequiredSum,int Tmpsum)
{
if(Index>=size)
{
int x=0;
x=abs(Tmpsum-RequiredSum);
if(MinDiff>x)
{
int i=0;
for(;i<size;i++) {="" final[i]="arr1[i];" }="" mindiff="x;" }="" return;="" }=""
arr1[index]="Input[Index];" findsubset(input,index+1,requiredsum,tmpsum+input[index]);=""
arr1[index]="0;" findsubset(input,index+1,requiredsum,tmpsum);="" }="" int="" main()="" {=""
int="" input[size];="" int="" i="0,TotalSum=0,RequiredSum=0;" for(;i<size;i++)="" {=""
printf("enter="" %dth="" element:\n",i);="" scanf("%d",&input[i]);="" totalsum+="Input[i];" }=""
requiredsum="TotalSum/2;" findsubset(input,0,requiredsum,0);="" call="" for="" findsubset=""
function="" for(i="0;i&lt;size;i++)" {="" if(final[i]!="0)" printf("%d\t",final[i]);="" }="">

2 • Reply • Share ›

Vinit Gupta • 2 years ago

Guys we can solve this by generating all permutation of that array and the answer will be the permutation which has min difference between initial n/2 and last n/2 element.

code for above implementation as follows:-

#include<iostream>

```
#include<stdlib.h>
#include<math.h>
using namespace std;
int minDiff= 10000;
int sol[1000];
void replace_Val(int inp[],int i,int j){
int temp=inp[i];
inp[i]=inp[j];
inp[j]=temp;
}
int diff(int a[],int n){
int left=0;
```

see more

• Reply • Share ›

Sachin Aglave • 2 years ago

we could have done using dp also.

• Reply • Share ›

rakitic • 2 years ago

@geeks...what is the purpose for making the current_position false in last statement ??

```
}
else
{
// consider the cases where current element is included in the solution
TOWUtil(arr, n, curr_elements, no_of_selected_elements, soln,
min_diff, sum, curr_sum, curr_position+1);
}

// removes current element before returning to the caller of this function
curr_elements[curr_position] = false;
}
```

• Reply • Share ›

deb rakitic • 2 years ago

I think it's the backtracking step...

• Reply • Share ›

Parin • 2 years ago

Is this code correct?

Also,if you can answer if this is efficient enough?

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

```
class demo
{
    public static void main(String args[]) throws Exception
    {
        Thread d=new Thread();
        int ss=0,mm=0,hr=0;
        while(true)
        {
                d.sleep(1000);
                ss++;
                if(ss==60)
                {
                        mm++;
```

see more

^ | ˅ • Reply • Share ›

Parin • 2 years ago

IS this code correct?
And also if you can answer,please tell me if this is efficient or not.

```
import java.util.*;
class demo
{
static int n=0,count=0,taken=0,remain=0,min=20000,test=0;
static int x[],y[],a[];
public static void main(String args[])
{
System.out.print("Enter array size.");
Scanner sc=new Scanner(System.in);
n=sc.nextInt();
x=new int[n];
y=new int[n];
a=new int[n];
System.out.println("Enter elements");
for(int i=0;i<n;i++)
{
```

see more

1 ^ | ˅ • Reply • Share ›

bateesh • 2 years ago

Can anyone explain how to apply partioning approach in this?I tried but unable to do so as we need to maintain n/2 elements also.We can also do it by creating a 2-d matrix and of n*n.where matrix[i][j] will represent sum of elements from i to j and apply same approach as of recurssion

matrix[i][j] will represent sum of elements from i to j and apply same approach as of recurssion but it will not have overlapping subproblems as we have already obtained sum from i to j.Say i have 6 elements then to select 3 from them(1,2,3,4,5,6) i will start from reverse order.Say,i start from 4,5,6 first.Then move one backward to 3 then from 3 we fix 3 and find sum 4-5 (3+4+5),then 4 and 6(3+4+6),then 5-6(3+5+6).And find minimum and update minimum solution set if possible.Then move to 2 and find all set from 2 to n of size 3 which will have one element as 2.Plz comment on this approach.
0(n^2) is not possible for this i think.
• Reply • Share ›

Gautam bateesh • 2 years ago
plz copy my code and run it using gcc and go step by step i think u will find all the thing clearly
• Reply • Share ›

Rahul • 2 years ago
I guess there will be two solutions for this example.
One that is stated above and other can be:
First set:45 -34 12 98 0
Second set:23 -1 -99 4 189 4
• Reply • Share ›

AMIT • 2 years ago
I think better algorithms exists for such problems of divisions into half equal partitions--KL algorithm(algorithm for partitioning) was one such algorithm with a non-exponential time complexity.Please correct me if I am wrong
1 • Reply • Share ›

Vishnu Vasanth R • 2 years ago
Hi All, The solution for Tug of war looks like Backtracking | Set 4 (Subset Sum)problem. I have reused its function to simplify the answer. It gives correct solution. Someone please update the answer with my code. This is way too easy.

```
[sourcecode language="C++"]
/* Paste your code here (You may delete these lines if not writing code) */
// TugOfWar.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"

#include <iostream>
#include <stdlib.h>
#include <limits.h>
using namespace std;
static int total_nodes;
```

```
// function that tries every possible solution by calling itself recursively
void TOWUtil(int* arr, int n, bool* curr_elements, int no_of_selected_elements,
```

see more

• Reply • Share ›

subhasish • 2 years ago

Please check the following algo and point out if something is wrong :

1. sort the array
2. Initialize two sets with two largest numbers
3. Loop until #remaining element <= 1
---a. If sum of set 1 is greater than sum of set 2 include smallest among remaining elements, else include greatest element
---b. Similarly add one element to set 2
4. If any item remains (odd number of elements) add to the set having smaller sum

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

• Reply • Share ›

Load more comments

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
    - Ashish Aggarwal

      Try Data Structures and Algorithms Made Easy -...

      Algorithms · 17 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) · 1 hour ago

- saeid aslami

  thanks

  Greedy Algorithms | Set 7 (Dijkstra's shortest path algorithm) · 1 hour ago

- Cracker

  Implementation:...

  Implement Stack using Queues · 2 hours ago

-