GeeksforGeeks

A computer science portal for geeks

Practice

IDE	Q&A	GeeksQuiz

Find four elements that sum to a given value | Set 2 (O(n^2Logn) Solution)

Given an array of integers, find all combination of four elements in the array whose sum is equal to a given value X.

For example, if the given array is $\{10, 2, 3, 4, 5, 9, 7, 8\}$ and X = 23, then your function should print "3 5 7 8" (3 + 5 + 7 + 8 = 23).

Sources: Find Specific Sum and Amazon Interview Question

We have discussed a O(n^3) algorithm in the previous post on this topic. The problem can be solved in O(n^2Logn) time with the help of auxiliary space.

Thanks to itsnimish for suggesting this method. Following is the detailed process.

Let the input array be A[].

- 1) Create an auxiliary array aux[] and store sum of all possible pairs in aux[]. The size of aux[] will be n*(n-1)/2 where n is the size of A[].
- 2) Sort the auxiliary array aux[].
- 3) Now the problem reduces to find two elements in aux[] with sum equal to X. We can use method 1 of this post to find the two elements efficiently. There is following important point to note though. An element of aux[] represents a pair from A[]. While picking two elements from aux[], we must check whether the two elements have an element of A[] in common. For example, if first element sum of A[1] and A[2], and second element is sum of A[2] and A[4], then these two elements of aux[] don't represent four distinct elements of input array A[].

Following is C implementation of this method.

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

// The following structure is needed to store pair sums in aux[]
struct pairSum
{
   int first; // index (int A[]) of first element in pair
   int sec; // index of second element in pair
   int sum; // sum of the pair
};
```

```
// Following function is needed for library function qsort()
int compare (const void *a, const void * b)
{
    return ( (*(pairSum *)a).sum - (*(pairSum*)b).sum );
}
// Function to check if two given pairs have any common element or not
bool noCommon(struct pairSum a, struct pairSum b)
    if (a.first == b.first || a.first == b.sec ||
            a.sec == b.first || a.sec == b.sec)
        return false;
    return true;
}
// The function finds four elements with given sum X
void findFourElements (int arr[], int n, int X)
    int i, j;
    // Create an auxiliary array to store all pair sums
    int size = (n*(n-1))/2;
    struct pairSum aux[size];
    /* Generate all possible pairs from A[] and store sums
       of all possible pairs in aux[] */
    int k = 0:
    for (i = 0; i < n-1; i++)
        for (j = i+1; j < n; j++)
            aux[k].sum = arr[i] + arr[j];
            aux[k].first = i;
            aux[k].sec = j;
            k++;
        }
    }
    // Sort the aux[] array using library function for sorting
    qsort (aux, size, sizeof(aux[0]), compare);
    // Now start two index variables from two corners of array
    // and move them toward each other.
    i = 0;
    j = size-1;
    while (i < size && j >=0 )
        if ((aux[i].sum + aux[i].sum == X) && noCommon(aux[i], aux[i]))
            printf ("%d, %d, %d, %d\n", arr[aux[i].first], arr[aux[i].sec],
                                      arr[aux[j].first], arr[aux[j].sec]);
            return:
        else if (aux[i].sum + aux[j].sum < X)</pre>
        else
            j--;
    }
// Driver program to test above function
int main()
```

```
int arr[] = {10, 20, 30, 40, 1, 2};
int n = sizeof(arr) / sizeof(arr[0]);
int X = 91;
findFourElements (arr, n, X);
return 0;
}
```

Run on IDE

Output:

```
20, 1, 30, 40
```

Please note that the above code prints only one quadruple. If we remove the return statement and add statements "i++; j-;", then it prints same quadruple five times. The code can modified to print all quadruples only once. It has been kept this way to keep it simple.

Time complexity: The step 1 takes $O(n^2)$ time. The second step is sorting an array of size $O(n^2)$. Sorting can be done in $O(n^2 Logn)$ time using merge sort or heap sort or any other O(n Logn) algorithm. The third step takes $O(n^2)$ time. So overall complexity is $O(n^2 Logn)$.

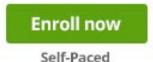
Auxiliary Space: O(n^2). The big size of auxiliary array can be a concern in this method.

Please write comments if you find any of the above codes/algorithms incorrect, or find other ways to solve the same problem.



Querying with Transact-SQL





65 Comments Category: Arrays

Related Posts:

- Longest Span with same Sum in two Binary arrays
- Count Inversions of size three in a give array
- Find the subarray with least average

- Count triplets with sum smaller than a given value
- Find zeroes to be flipped so that number of consecutive 1's is maximized
- Reorder an array according to given indexes
- Find maximum value of Sum(i*arr[i]) with only rotations on given array allowed
- · Find maximum average subarray of k length

@geeksforgeeks, Some rights reserved

(Login to Rate and Mark)			
Average Difficulty: 3.5/5.0 Based on 11 vote(s)	Add to TODO List Mark as DONE		
Like Share 6 people like this. Be the first of your friend	ds.		
Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.			

Contact Us!

About Us!

Advertise with us!