GTU Department of Computer Engineering CSE 331/503 - Fall 2020 Homework 2 Report

Akif KARTAL
171044098

Problem Definition

The problem is to find if a subset of given array elements can sum up to the target num. This problem called as SUBSETSUM which is one of the popular problem in dynamic programming.

Inputs: arr(Array), num(target sum), size(size of Array)

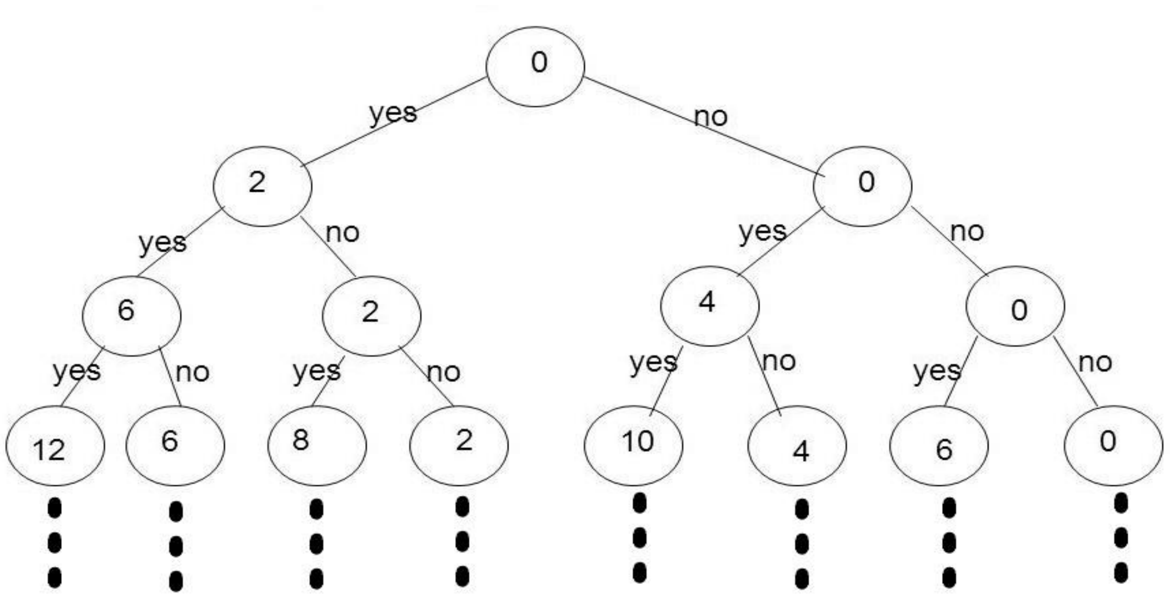
Output: Possible or Not possible (0 or 1)

My Solution

Since, this problem is a recursive backtracking problem I made my algorithm by using following approach.

To solve this problem, I have two value **current sum** and **total sum**. While searching a solution I am checking two cases, one of **include current element** of array into the sum and try to find solution or **not include current** element of array into the sum and try to find solution. To get better performance, if current sum exceed the target, I will terminate that search and if total sum is less than target again I will terminate that search.

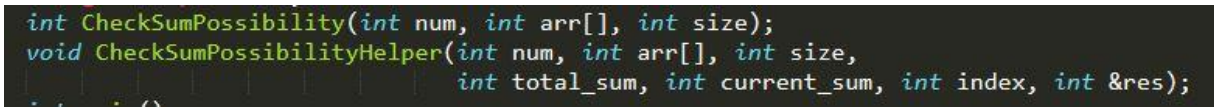
In the following sample picture **yes** stands for continue **no** stands for not continue to search.



Each node contains current sum and total sum of elements

- **Note:** To implement this algorithm in C++ I had to use helper function for extra parameters. I will mention about this next page.

My Functions in C++



```
int CheckSumPossibility(int num, int arr[], int size);
void CheckSumPossibilityHelper(int num, int arr[], int size,
                               int total_sum, int current_sum, int index, int &res);
```

First function is given function in homework I used it to check some extreme cases such as num is equal to 0 which is already Not Possible case and to initialize variables such as total sum.

Second Function is my recursive function with following properties;

Parameters:
total_sum : total sum of given array (sum of all element in the array)

current_sum: sum of subsets (initialized with 0)

index: current index of array (initialized with 0)

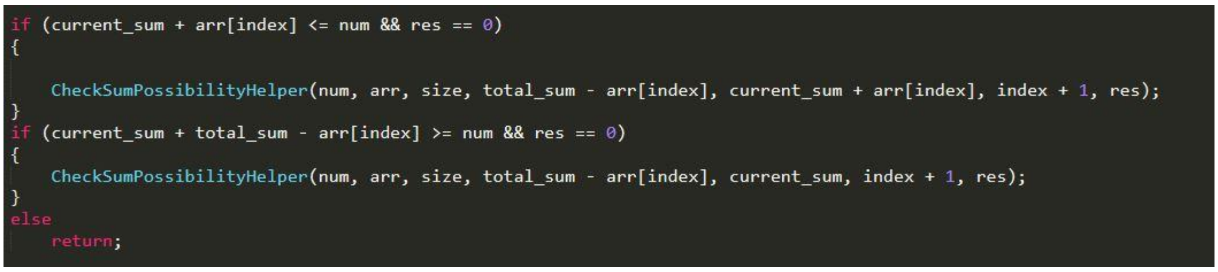
res = result of recursive search 0 or 1 (initialized with 0)

Output:

Output result is in res parameter as 0 or 1.

Note: I have also **test()** function I will explain it at end of this report.

Recursive Calls;



```
if (current_sum + arr[index] <= num && res == 0)
{

    CheckSumPossibilityHelper(num, arr, size, total_sum - arr[index], current_sum + arr[index], index + 1, res);
}
if (current_sum + total_sum - arr[index] >= num && res == 0)
{
    CheckSumPossibilityHelper(num, arr, size, total_sum - arr[index], current_sum, index + 1, res);
}
else
    return;
```

Note that in these recursive calls we are making **2 optimization** mentioned in homework pdf file;

a. you must ignore the next recursive calls when the sum exceeded the target number num.

- In if part I am checking this with “<=” operator

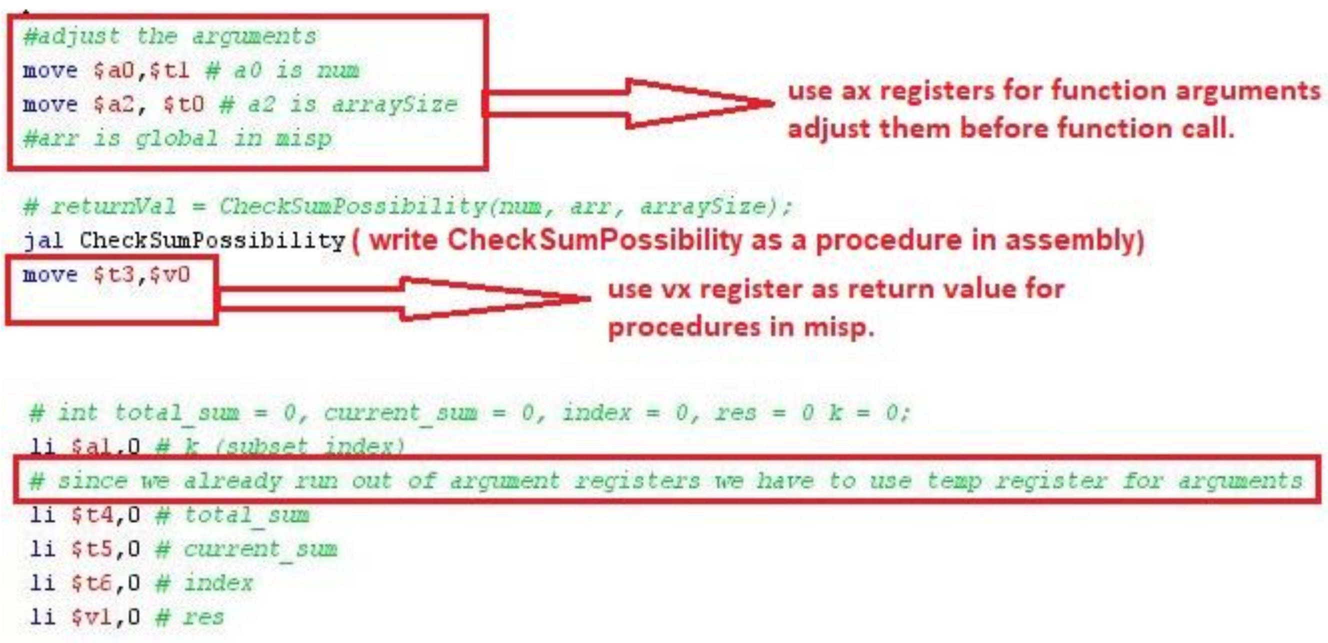
b. your program should stop when only one possible combination is found.

- In if part I am checking this with res variable
- Also in base case I am checking this.

Some Information about the my program

- I removed bonus part **in C++** to make understanding easier.
- In C++ part I didn't change **given main function** so give numbers according to this. You can check test results for this. (just hit enter after each number array size times.)

Obeying the contract in mips

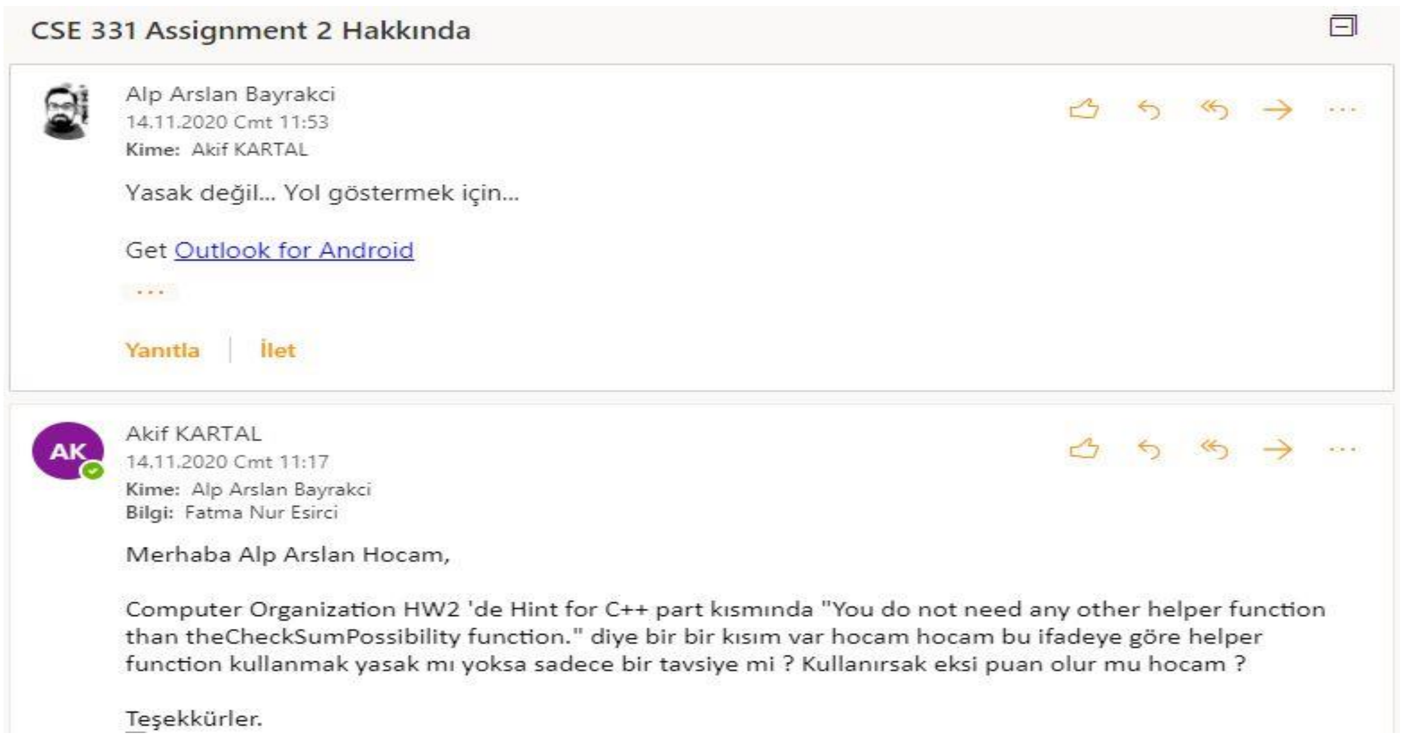


```
#adjust the arguments
move $a0,$t1 # a0 is num
move $a2, $t0 # a2 is arraySize
#arr is global in misp

# returnVal = CheckSumPossibility(num, arr, arraySize);
jal CheckSumPossibility
move $t3,$v0

# int total_sum = 0, current_sum = 0, index = 0, res = 0 k = 0;
li $a1,0 # k (subset index)
# since we already run out of argument registers we have to use temp register for arguments
li $t4,0 # total_sum
li $t5,0 # current_sum
li $t6,0 # index
li $v1,0 # res
```

use ax registers for function arguments adjust them before function call.
(write CheckSumPossibility as a procedure in assembly)
use vx register as return value for procedures in misp.

- **Note About Mars Part:** In mars part I didn't use memory for variables other than arrays. Because this is the mips rule, if registers are enough to perform operations don't use memory for better performance.

Information about my helper function



CSE 331 Assignment 2 Hakkında

Alp Arslan Bayrakci
14.11.2020 Cmt 11:53
Kime: Akif KARTAL

Yasak değil... Yol göstermek için...

Get Outlook for Android

...

Yanıtla | İlet

Akif KARTAL
14.11.2020 Cmt 11:17
Kime: Alp Arslan Bayrakci
Bilgi: Fatma Nur Esirci

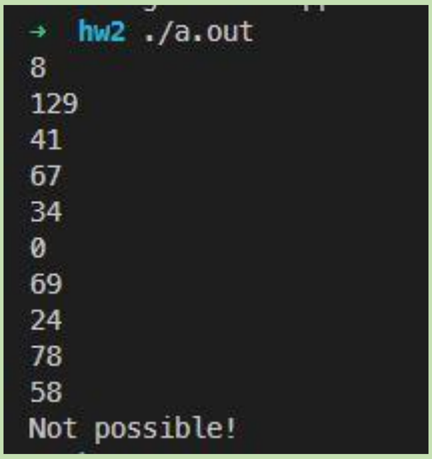
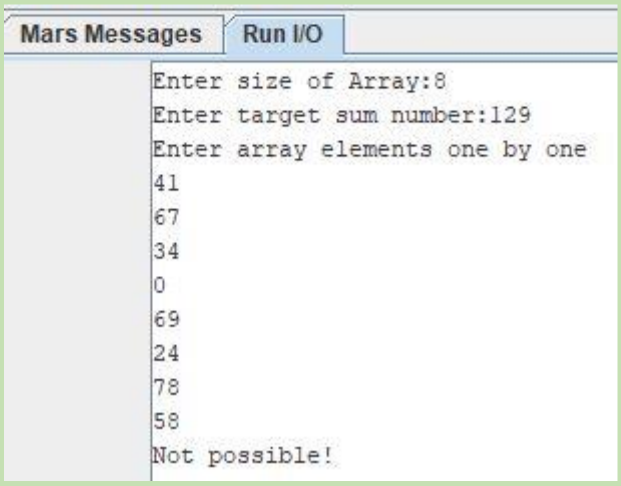
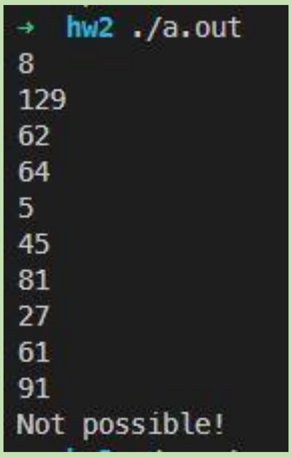
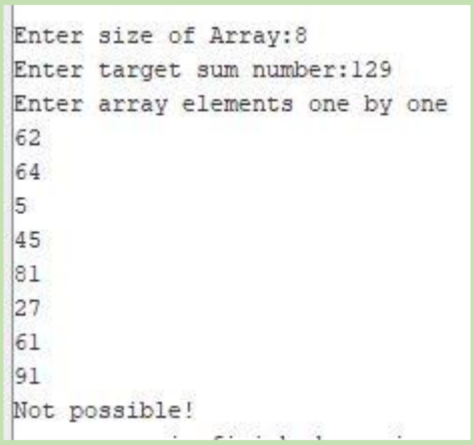
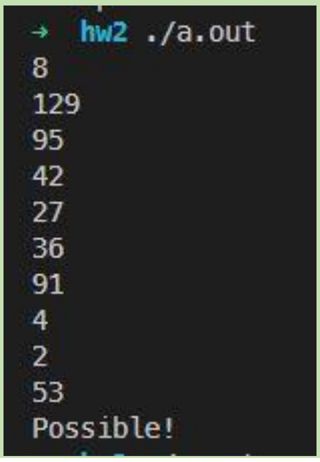
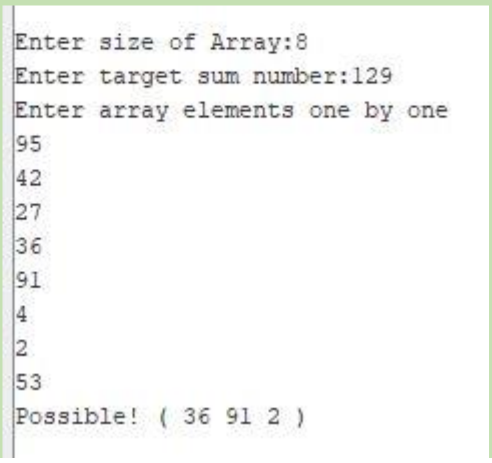
Merhaba Alp Arslan Hocam,

Computer Organization HW2 'de Hint for C++ part kısmında "You do not need any other helper function than theCheckSumPossibility function." diye bir bir kısım var hocam hocam bu ifadeye göre helper function kullanmak yasak mı yoksa sadece bir tavsiye mi ? Kullanırsak eksi puan olur mu hocam ?

Teşekkürler.

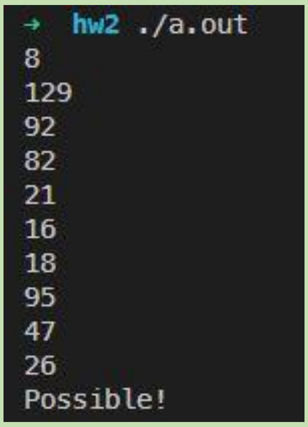
Test Results(C++ and Mars Parts)

- **Note that** bonus part implemented in my assembly program.

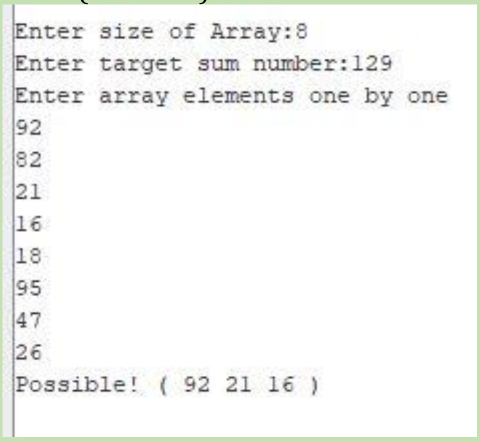
Number	Test Result
1	8 129 41 67 34 0 69 24 78 58 Not possible!  → hw2 ./a.out 8 129 41 67 34 0 69 24 78 58 Not possible!  Mars Messages · Run I/O Enter size of Array:8 Enter target sum number:129 Enter array elements one by one 41 67 34 0 69 24 78 58 Not possible!
2	8 129 62 64 5 45 81 27 61 91 Not possible!  → hw2 ./a.out 8 129 62 64 5 45 81 27 61 91 Not possible!  Enter size of Array:8 Enter target sum number:129 Enter array elements one by one 62 64 5 45 81 27 61 91 Not possible!
3	8 129 95 42 27 36 91 4 2 53 Possible! (36 91 2)  → hw2 ./a.out 8 129 95 42 27 36 91 4 2 53 Possible!  Enter size of Array:8 Enter target sum number:129 Enter array elements one by one 95 42 27 36 91 4 2 53 Possible! (36 91 2)

4

8 129
92 82 21 16 18 95 47
Possible! (92 21 16)



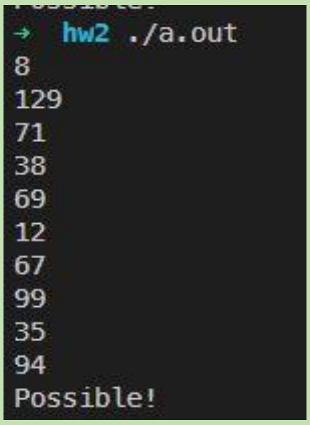
```
➜  hw2 ./a.out
8
129
92
82
21
16
18
95
47
26
Possible!
```



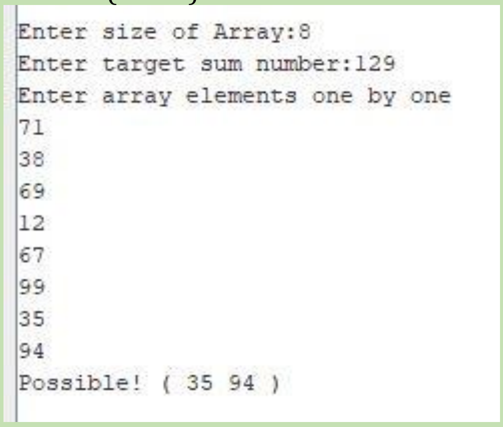
```
Enter size of Array:8
Enter target sum number:129
Enter array elements one by one
92
82
21
16
18
95
47
26
Possible! ( 92 21 16 )
```

5

8 129
71 38 69 12 67 99 35 94
Possible! (35 94)



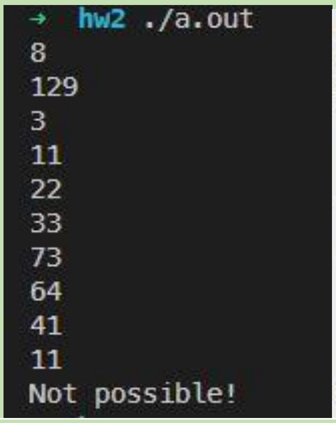
```
➜  hw2 ./a.out
8
129
71
38
69
12
67
99
35
94
Possible!
```



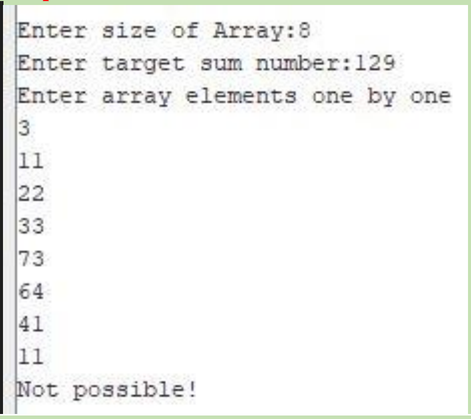
```
Enter size of Array:8
Enter target sum number:129
Enter array elements one by one
71
38
69
12
67
99
35
94
Possible! ( 35 94 )
```

6

8 129
3 11 22 33 73 64 41 11
Not possible!



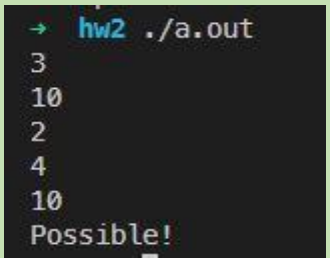
```
➜  hw2 ./a.out
8
129
3
11
22
33
73
64
41
11
Not possible!
```



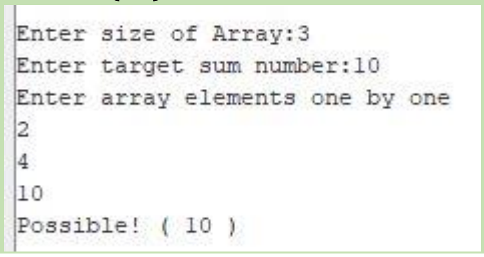
```
Enter size of Array:8
Enter target sum number:129
Enter array elements one by one
3
11
22
33
73
64
41
11
Not possible!
```

7

3 10
2 4 10
Possible! (10)



```
➜  hw2 ./a.out
3
10
2
4
10
Possible!
```



```
Enter size of Array:3
Enter target sum number:10
Enter array elements one by one
2
4
10
Possible! ( 10 )
```

Bonus Test Results about Optimization

- I tested my algorithm with given sample txt file “Prob2_SampleOutput_Optimized.txt”.
- In these tests I read each array from **example.txt** which is same as “Prob2_SampleOutput_Optimized.txt” but only contains array and target number then, I tested with my solution these arrays after that I wrote the result in the my_results.txt file.

Note: İf you want to see my test results just uncommented line 41 in my main function and comment other parts of main and run program results will be in the my_results.txt.

Comparison between Results (My results won!)

İf we compare **my results and given Optimized results** we will observe that **either my results are correct or my average number of function call is lower than given sample file.**

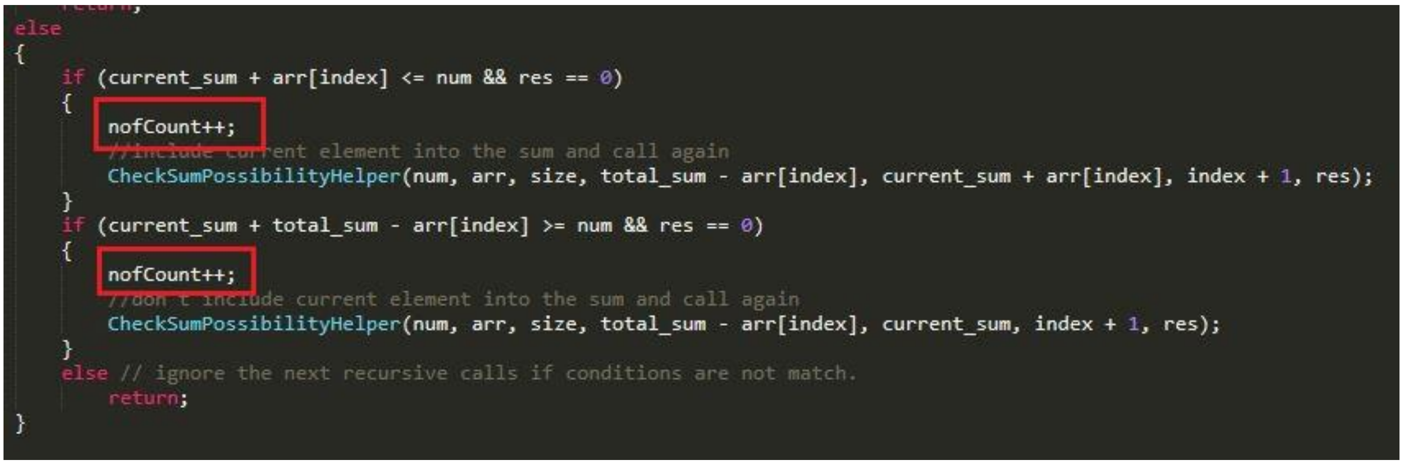
My number of function call **average is**: 630/25 : **25.2**

Given Sample File number of function call **average is**: 726/25 : **29.04**

- In some cases my number of call count can be higher but important part is in general not just one case.

Measuring number of function call

To measure My number of function call I just add two line in my algorithm;

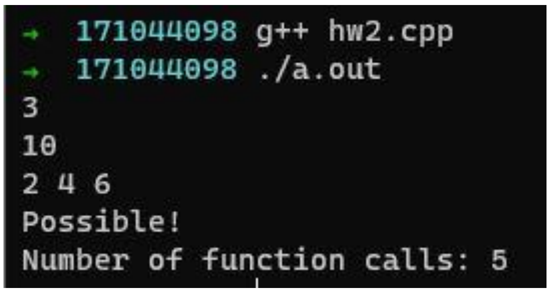


```
else
{
    if (current_sum + arr[index] <= num && res == 0)
    {
        nofCount++;
        //include current element into the sum and call again
        CheckSumPossibilityHelper(num, arr, size, total_sum - arr[index], current_sum + arr[index], index + 1, res);
    }
    if (current_sum + total_sum - arr[index] >= num && res == 0)
    {
        nofCount++;
        //don't include current element into the sum and call again
        CheckSumPossibilityHelper(num, arr, size, total_sum - arr[index], current_sum, index + 1, res);
    }
    else // ignore the next recursive calls if conditions are not match.
        return;
}
```

In test function I just read given 25 array in sample file and I checked them with my algorithm and I printed the result into my_results.txt file.

Last Update! (I did many :))

After last PS, I started to print number of function call as in following example;



```
→  171044098 g++ hw2.cpp
→  171044098 ./a.out
3
10
2 4 6
Possible!
Number of function calls: 5
```

So many thing has changed and my report expand with these changes.