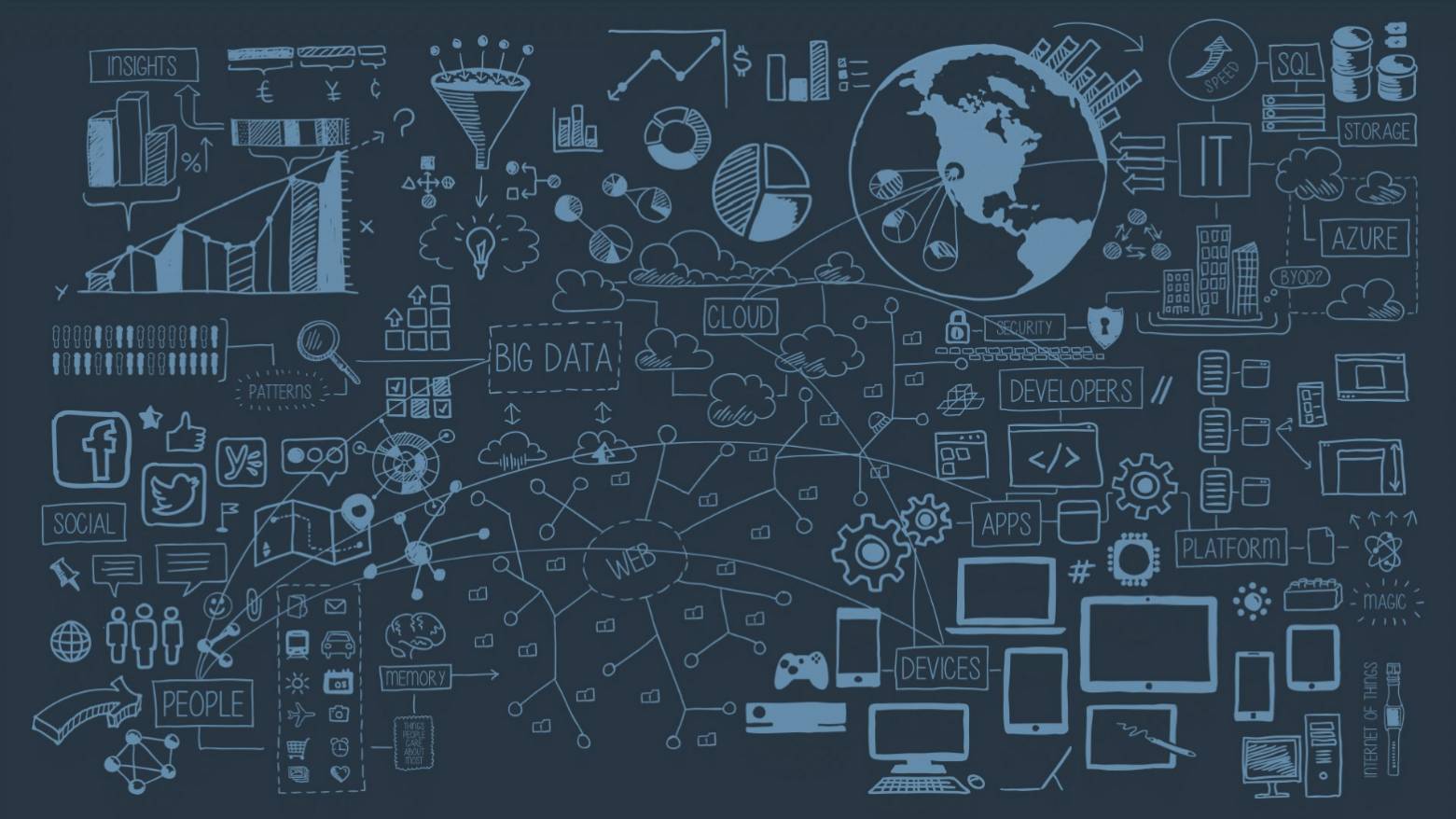
DATA STRUCTURES AND ALGORITHMS PROJECT REPORT



Team Details:-

Nishant Shukla | 18BCE0348

Sahaj Adlakha | 18BCE0054

Deva S | 18BCE2103

Title- Solution to Subset- Sum problem using Approximation and Backtracking.

Language Used- C

Abstract

Problem Statement:-

The Subset Sum problem is a non-deterministic problem where we need to find a resultant number from a set of numbers to perform the addition of subset of a set. Non-polynomial problems comprise of the set of decision problems where answer to any instance of the problem is true then it can be easily proved why the solution is true. Non-Deterministic

Polynomial problems are the collection of problems where if the solution is true, then it provides the complexity of the problem in polynomial time.

Introduction

Objective:-

In the subset sum problem we have to find a subset s of the given set S=<S1, S2, S3………Sn> where the elements of the set S are n positive integers in such a manner that s belongs to S and the sum of the elements of subset s` is equal to some positive integer ‘X’. It is well known that the above problem is NP complete; there is trivial solution with computational complexity O(2n) . The sub set problem can be solved by using various approaches like dynamic techniques, backtracking approaches etc.

Dynamic programming is a useful technique for making a sequence of interrelated decisions. It offers a step wise technique for finding the optimal combination of decisions.

The multistage decision policy with recursive approach will provides an efficient way while using Dynamic programming. In the scenario of multistage decision process the problem is divided into several parts known as sub problems, then each and every sub problem will be resolved individually and the end result will be found by combining the results of all the sub problems.

Constraints:-

We take the elements in the given set in sorted order (ascending).

Algorithms used:-

1. Subset Sum Approximation.
2. Backtracking.

Subset Sum Approximation:-

Inputs:

Integer N, number of elements

Elements, a set of N distinct positive integers

Integer C, the target sum

Output:

All subsets of the set Elements that have a sum equal to C.

Important declarations:

SSP (int N, int Elements[N], int C)

int Bitmap[N];

int K;

int PSum;

int Iteration;

int Count;

int I, J;

boolean Found;

Where

1. Bitmap is an array of size N and is used to store ones and zeros; 1 at position x indicated that the xth element is part of the identified solution. N is the size of the set of integers in hand.
2. K is the index of the last element in the sorted array that is smaller than the required sum C.
3. PSum holds the current partial sum.
4. Iteration is the number of already elapsed iterations (required to control the number of iterations covered).
5. Count holds the total number of already identified valid solutions.

The detailed algorithm example can be seen in the used reference paper.

Backtracking:-

The main condition to take care is:-

if(subset is satisfying the constraint)

print the subset

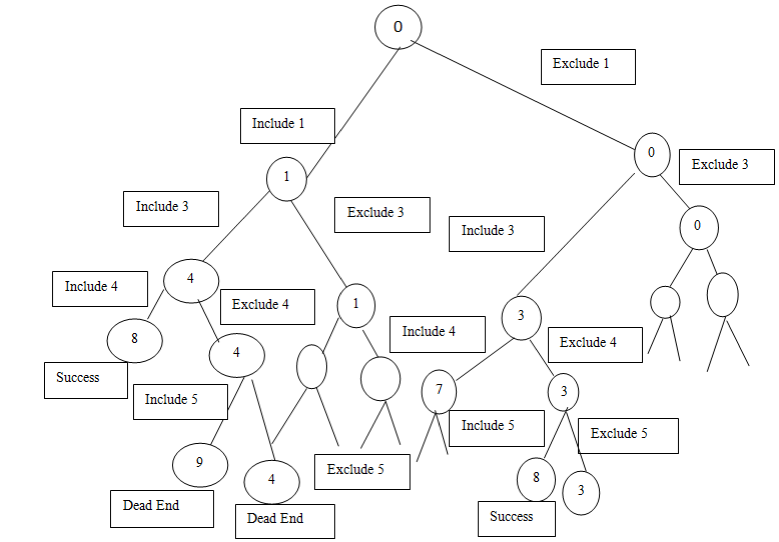
exclude the current element and consider next element

else

generate the nodes of present level along breadth of tree and

recursion for next levels

The method is as follows:-



Real time Application of subset sum problem:-

Computer Passwords:-

A computer needs to verify a user's identity before allowing him or her access to an account. The simplest system would have the machine keep a copy of the password in an internal file, and compare it with what the user types. A drawback is that anyone who sees the internal file could later impersonate the user.

I believe this alternative is actually implemented on some systems the computer generates a large number (say 500) of ai. They are stored in the internal file. A password is a subset of {1, 2, 3…….500}. (in practice, there is a program to convert a normal sequence-of-symbols password to such a subset.) Instead of having the password for the user, the computer keeps the total associated with the appropriate subset. When the user types in the subset, the computer tests whether the total is correct. It does not keep a record of the subset. Thus impersonation is possible only if somebody can reconstruct the subset knowing the ai and the total.

Message Verification:-

A sender wants to send messages to a receiver. Keeping the message secret is important. So receiver wants to be sure that the message he/she is receiving is not from a fraud and has not been tampered. So with the help of subset sum problem we can generate a verification code so that anybody who wants to read, first verify this by a password.

Used in Industries:-

Suppose a factory gets a contract to finish a product in 2 days and it has several machines which take different time to complete the work, using the subset sum algorithm we can find out which machines are suitable to complete the work on time.

Base paper explanation:-

Why did we choose these base papers?

1-The authors of the base papers have adapted a very easy method of explanation making it simple for the readers to understand the complex algorithms.

1. These papers were easy to access online.
2. There is proper pictorial/tabular explanation of the algorithm techniques taken up.
3. Proper references are stated at the end
4. One of the papers also contains the practical applications of the problem and algorithm technique.

Code for Subset-Sum Approximation:-

#include <stdio.h>

#include<stdbool.h>

#include<time.h>

//#include"helper.h"

void swap(int \*a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void bsort(int arr[], int n)

{

for(int i = 0; i < n-1 ; i++)

{

for(int j = 0; j < n - i -1; j++)

{

if(arr[j] > arr[j + 1])

swap(&arr[j], &arr[j + 1]);

}

}

}

int main()

{

clock\_t t;

t = clock();

printf("\n\n\n<<<SUBSET SUM PROBLEM WITH APPROXIMATION APPROACH>>>>\n\n");

int N=30;

/\*printf("Enter the total number of elements: ");

scanf("%d",&N);\*/

int Elements[N],i;

for(int i=0;i<N;i++)

{

Elements[i]=i+1;

}

//N=sizeof(Elements)/sizeof(Elements[0]);

/\*for(i=0;i<N;i++)

{printf("Enter element number %d: ",i);

scanf("%d",&Elements[i]);}\*/

int C=30;

/\*printf("Enter the Target Sum: ");

scanf("%d",&C);\*/

int bitmap[N], K, PSum, Iteration, Count, I, J;

bool Found;

bsort(Elements, N) ;

Iteration =1; Count =0;

int count=0;

for(i=0;i<N;i++)

{

if(Elements[i]<=C)

{

count++;

}

}

N=count;

/\*for(int k=0;k<N;k++)

{

printf("%d\t",Elements[k]);

}\*/

K=N;

//printf("%d\t%d\t%d\n",count,N,K);

for(I=K-1;I>=0;I--)

{

bitmap[I]=0;

}

//checked

PSum=0;

MID:

for(I=K-1;I>=0;I--)

{

if ((PSum + Elements[I]) <=C)

{

bitmap[I]=1;

PSum=PSum + Elements[I];

}

//printf("%d\t%d\t%d\n",PSum,Elements[I],bitmap[I]);

}

//printf("\n PSUM=%d\n",PSum);

/\*for(int k=0;k<N;k++)

{

printf("%d\n",bitmap[k]);

}\*/

if (C==PSum)

{

printf("One possible subset for sum equal to %d\n>>",C);

for(i=0;i<N;i++)

{

if (bitmap[i]==1)

{

printf("%d\t",Elements[i]);

}

}

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

Count++;

Found=true;

}

//printf("\n count=%d\n",Count);

J=0;

Found=false;

while((J<N-1)&&(!Found))

{

if( (bitmap[J]==0)&&(bitmap[J+1]==1))

{

Found = true;

break;

}

else

J++;

}

//printf("%d--",J);

//printf("\n");

if (!Found)

goto END;

else

{

bitmap[J]=1;

bitmap[J+1]=0;

for((I=0);I<J;I++)

{

bitmap[I]=0;

}

}

//important checking code down here!!!!!!!

/\* for(int k=0;k<N;k++)

{

printf("%d\t",bitmap[k]);

}\*/

//printf("\n");

//print(bitmap,N);

PSum=0;

for(I=N-1;I>=J;I--)

{

if (bitmap[I]==1)

{

PSum+=Elements[I];

}

}

K=J-1;

// printf("%d\n",PSum);

Iteration++;

//printf("%d",Iteration);

goto MID;

END:

{printf("\n\nITERATIONS = %d\nCOUNT = %d\n\*\*\*\*\*\*\*\*THE END\*\*\*\*\*\*\*\*\*\n",Iteration,Count);

t = clock() - t;

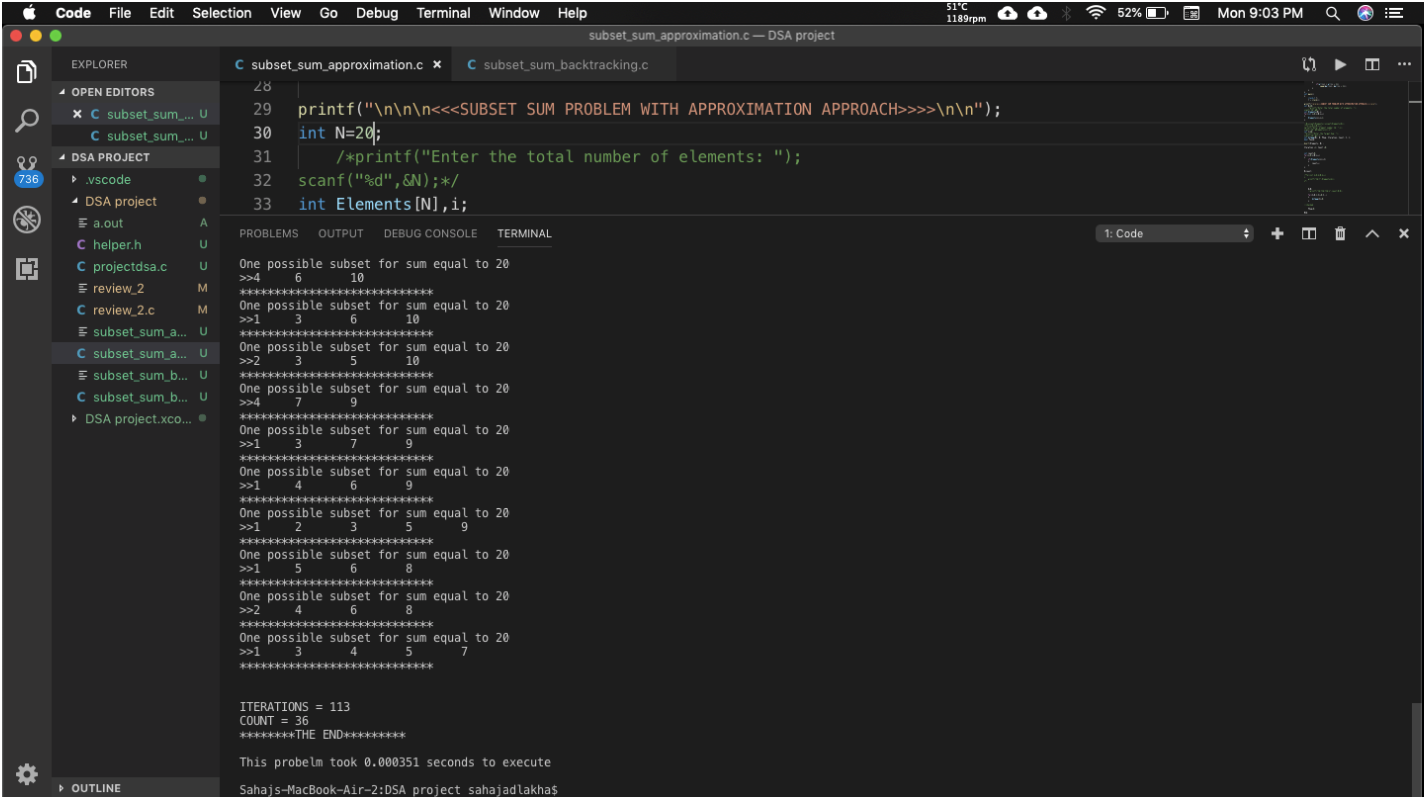
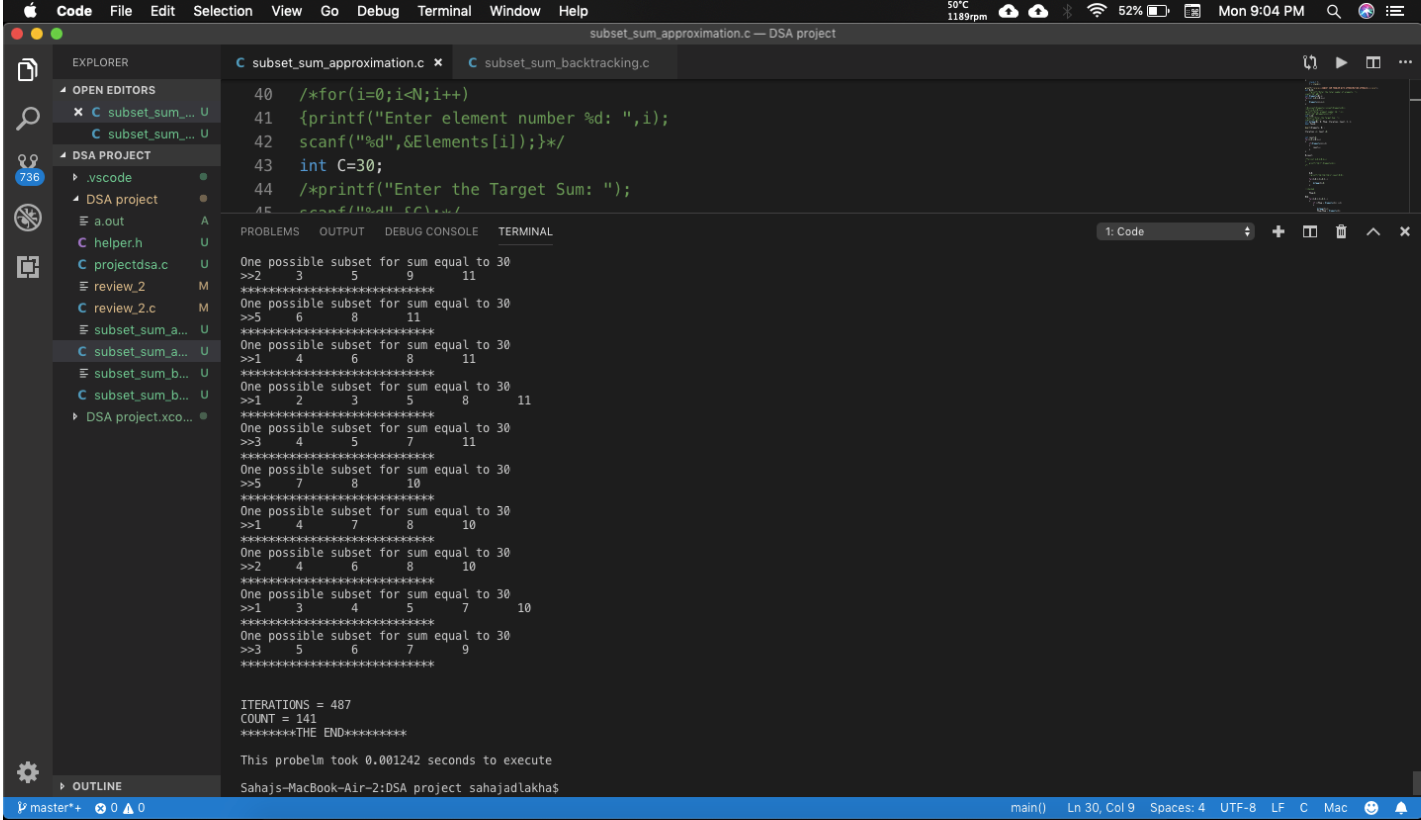
double time\_taken = ((double)t)/CLOCKS\_PER\_SEC; // in seconds

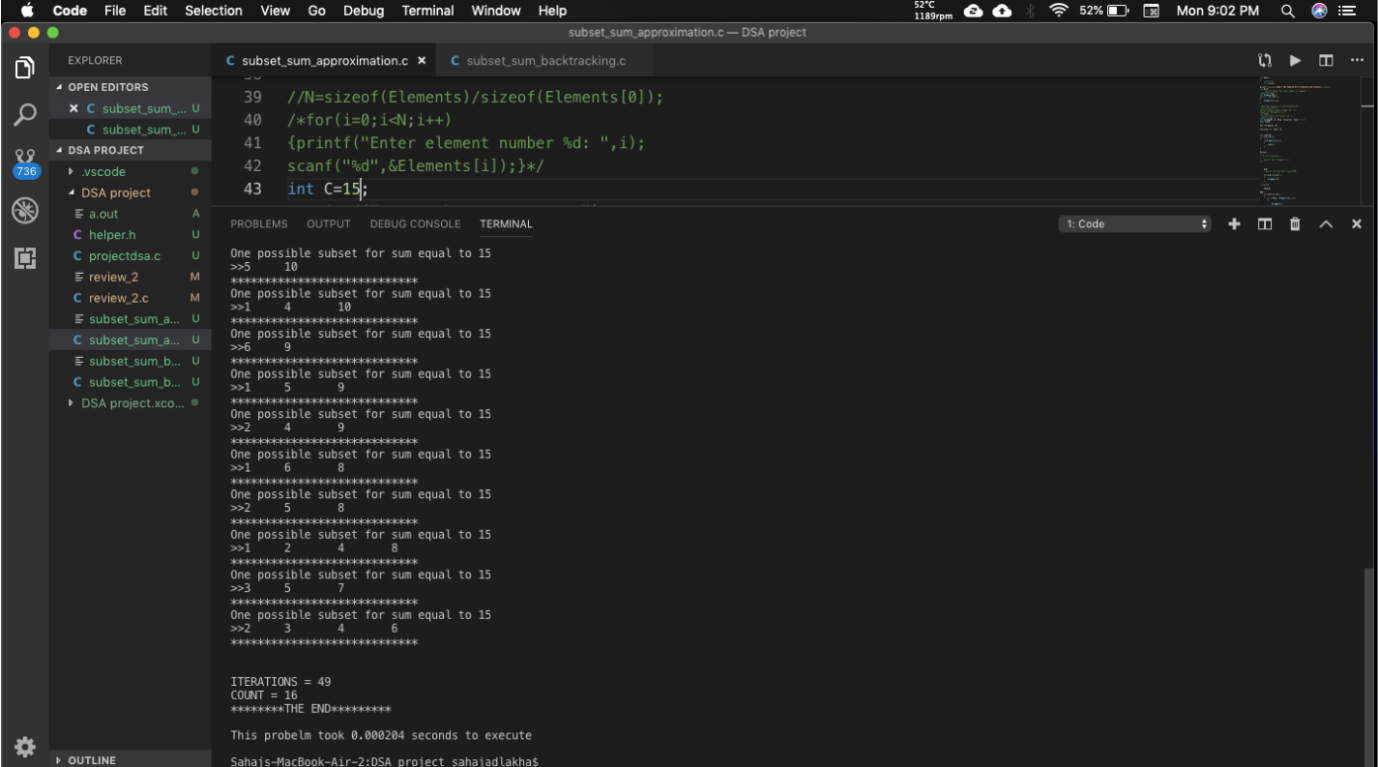
printf("\nThis problem took %f seconds to execute \n\n", time\_taken);

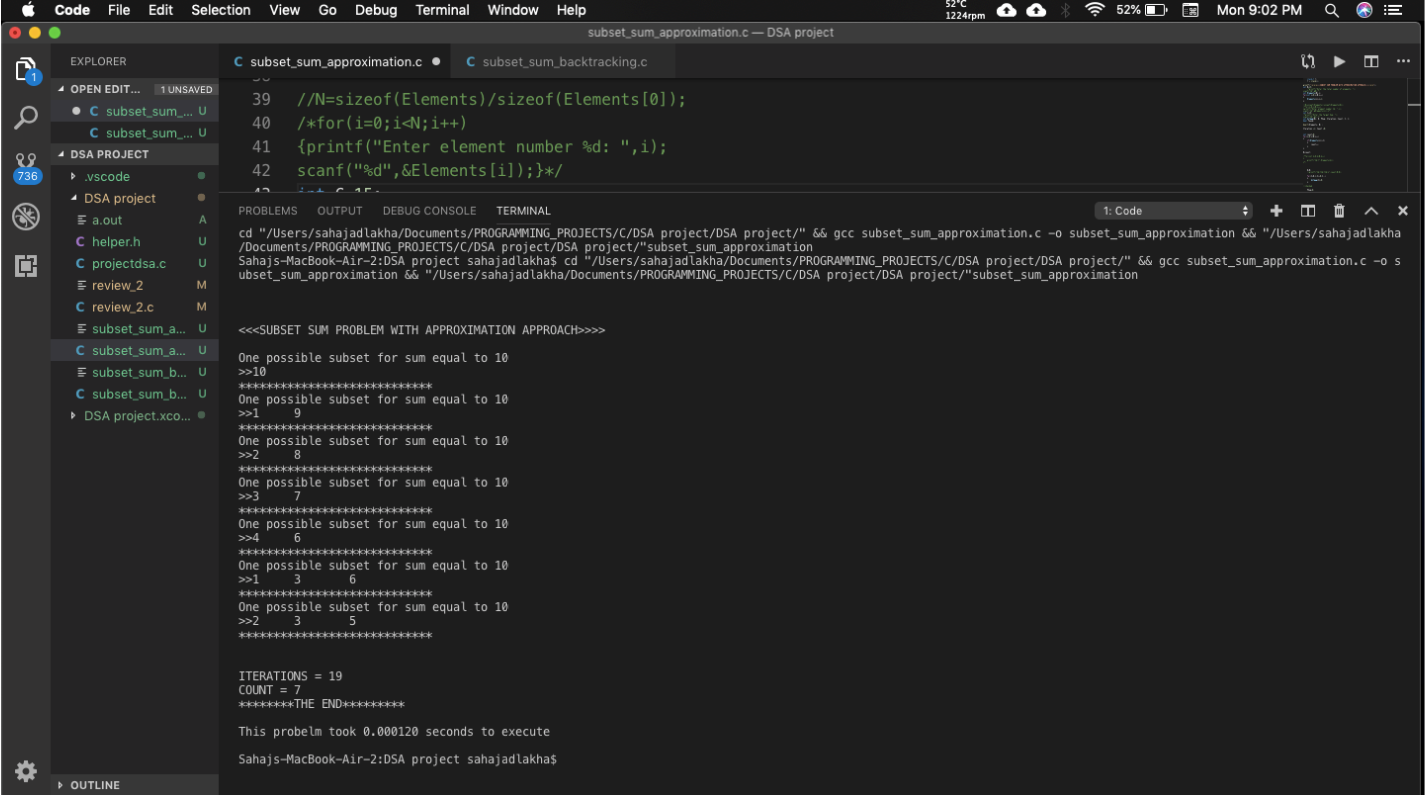
return 0;}

}

ScreenShots:-







Code for Backtracking approach:-

#include<stdio.h>

#include<stdlib.h>

void displaySubset(int subSet[], int size)

{

int i;

for(i = 0; i < size; i++)

{

printf("%d ",subSet[i]);

}

printf("\n");

}

void subsetSum(int set[], int subSet[], int n, int subSize, int total, int nodeCount ,int sum)

{

if( total == sum)

{

displaySubset(subSet, subSize);

subsetSum(set,subSet,n,subSize-1,total-set[nodeCount],nodeCount+1,sum);

return;

}

else

{

int i;

for(i = nodeCount; i < n; i++ )

{ //find node along breadth

subSet[subSize] = set[i];

subsetSum(set,subSet,n,subSize+1,total+set[i],i+1,sum);

}

}

}

void findSubset(int set[], int size, int sum)

{

int \*subSet = (int\*)malloc(size\*sizeof(int));

subsetSum(set, subSet, size, 0, 0, 0, sum);

free(subSet);

}

int main()

{

int weights[] = { 5,7,10, 18, 12, 20, 15,};

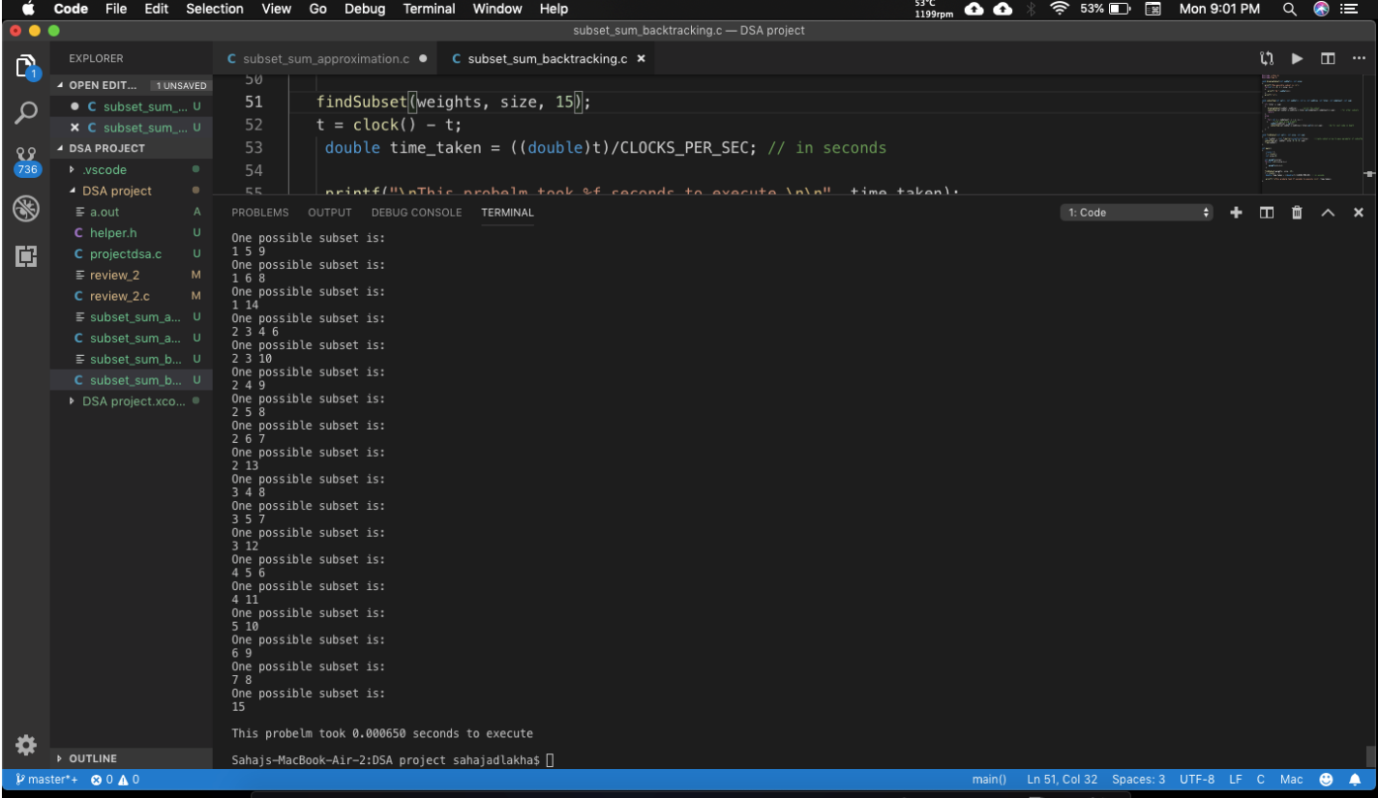
int size = 7;

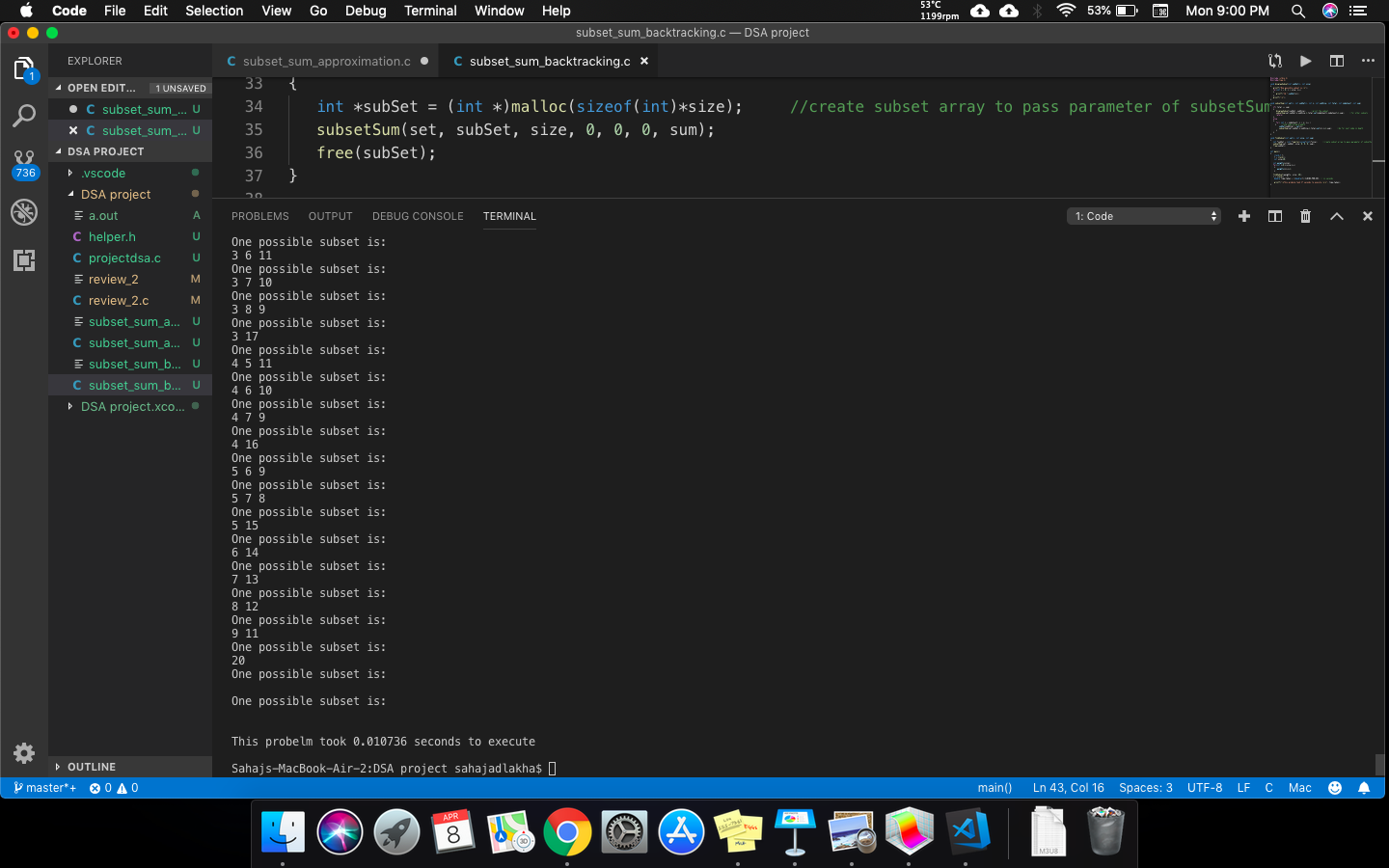
printf("The possible subsets are:-\n");

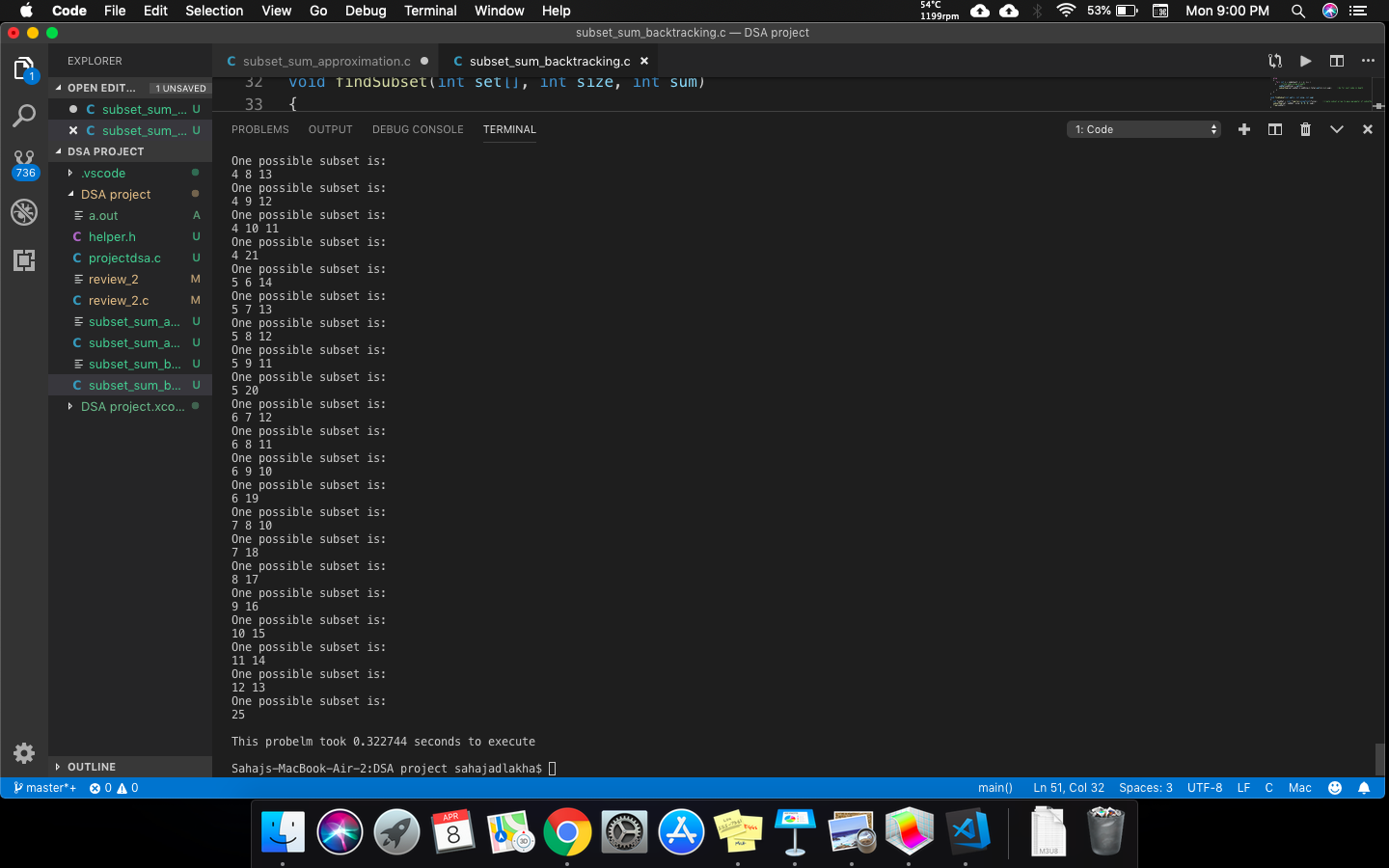
findSubset(weights, size, 30);

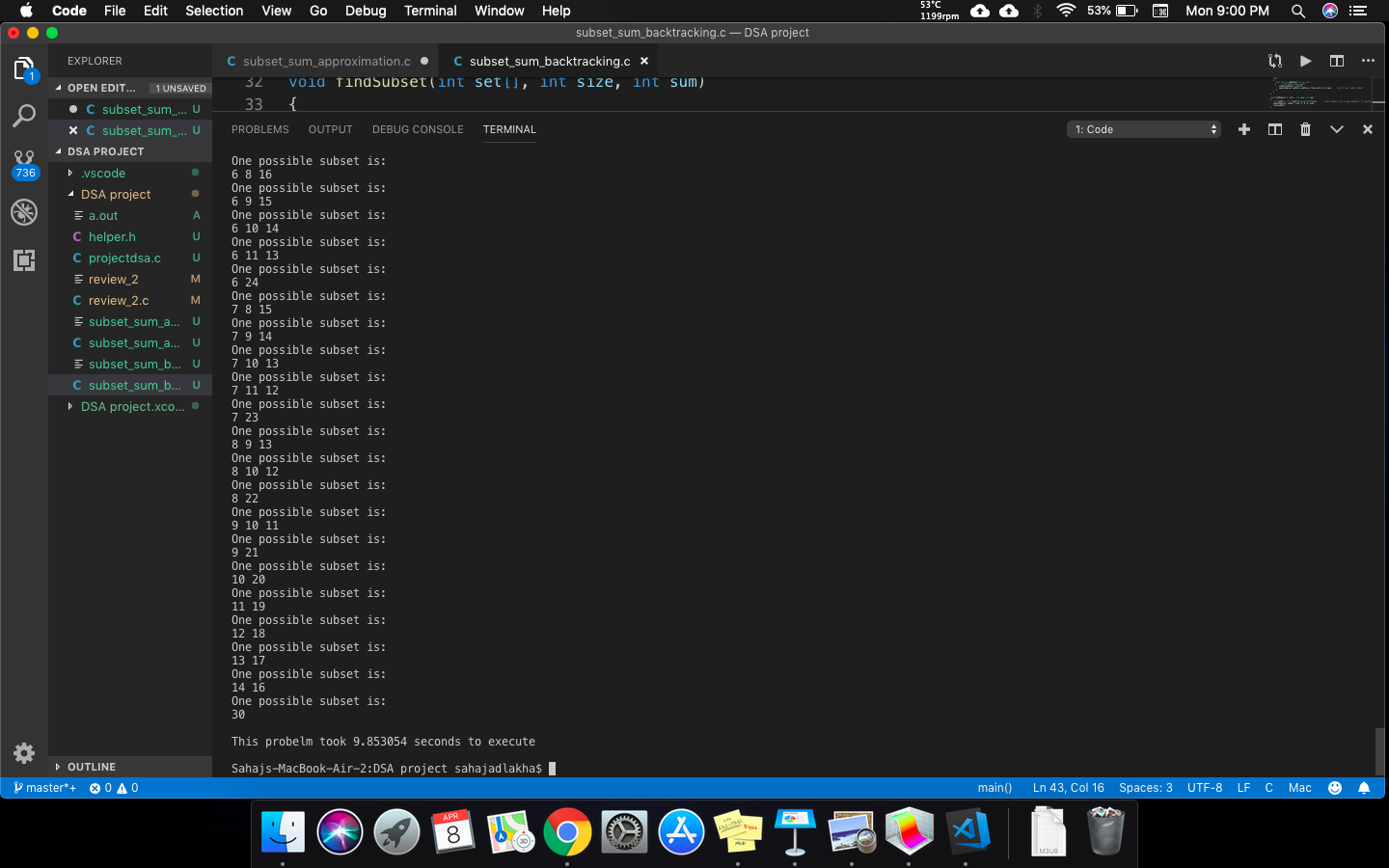
}

Screenshots:-



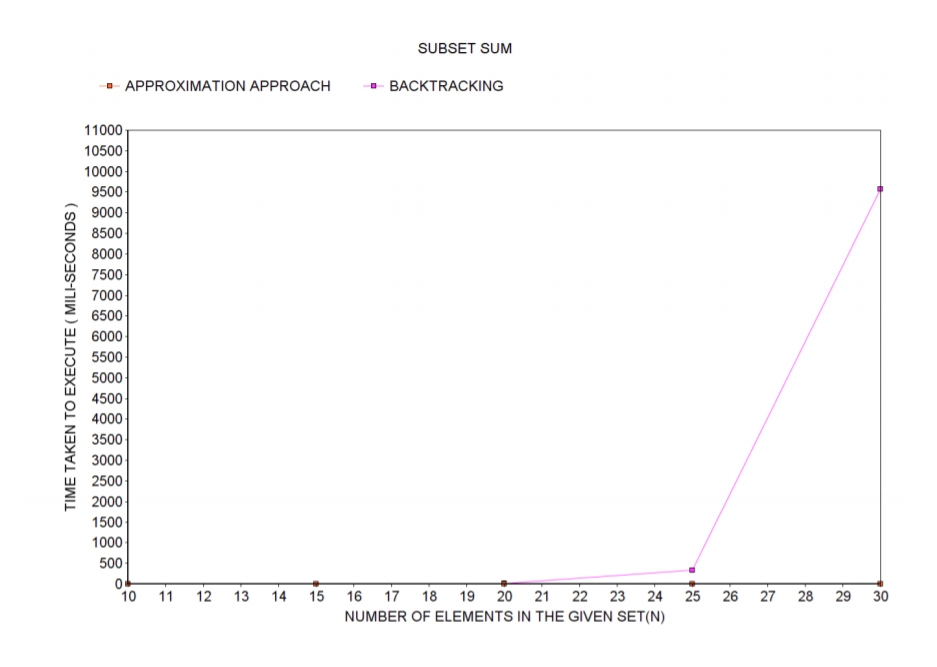






We tried to find various subsets by given different sets as input for each algorithm and drawn a graph which compares the efficiency of both algorithms.

Graph(Using online graph maker):-



Conclusion:-

As we can see clearly in the graph that the time taken in the approximation approach is way less than that in backtracking. We can say that approximation method is a much faster method. Although the length of code is bigger of approximation approach than backtracking, it is easier to understand the approximation algorithm. The use of bitmap is a unique way of selecting favourable elements.

References:-

* <https://www.geeksforgeeks.org/>
* <https://www.onlinecharttool.com/graph/fullscreen>
* <https://www.slideshare.net/ijlalhk/subset-sum-problem-dynamic-and-brute-force-approch>
* <https://cs.stackexchange.com/questions/13181/time-complexity-of-a-backtrack-algorithm>
* <https://www.geeksforgeeks.org/how-to-measure-time-taken-by-a-program-in-c/>