VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SUB SET SUM PROBLEM

REPORT SUBMITTED FOR THE FINAL REVIEW OF

COURSE NAME :DATA STRUCTURES AND ALGORITHMS COURSE CODE:-CSE2002

Ilantheral K Mam

TEAM MEMBERS

K.Surya Vamsi	18BCI0080
D.Sumanth	18BCI0067
T.Harsha vardhan	19BCT0253

TABLE OF CONTENTS

ACKNOWLEDGEMENT

The main reason for this report is to analyse the efficiency of subset sum problem using Dynamic programming approach and Backtracking. Our team firstly thanking our professor Ilantheral K mam for giving us this opportunity and providing us support in hard situations.

ABSTRACT

In this presentation, Subset Sum problem (Non Polynomial Complete problem) and analysis of its two solutions is discussed. These solutions are the Dynamic Solution algorithm and the Back Tracking algorithm. Dynamic Solution is a sound and complete algorithm that can be used to determine satisfiability and unsatisfiability with certainty. Backtracking also finds the satisfiability but will have a time complexity of exponential degree. In addition, analogy between these two algorithms and two other algorithms namely PL-Resolution and Walk-Sat algorithms for 3-CNF SAT problem, which is also NPC problem is discussed.

KEY WORDS :

- SUBSET SUM :

  Subset sum states that let there be a set A of n positive Integers and a sum s. Then the subset of A whose elements add upto the given sum is a subset sum.

- NPC PROBLEMS :

  In computational complexity theory an NP complete decision problem is one belonging to both the NP and NP hard complexity classes(NP stands for Non deterministic polynomial time).The set of NP complete problems is called NPC.

- PL-RESOLUTION :

  Propositional resolution is a powerful rule of inference for propositional logic. Using propositional resolution it is possible to build a theorem prover that is sound and complete for all of propositional logic.

WALKSAT ALGORITM :

In computer science GSAT and WalkSAT are local search algoritm to solve boolean satisfiability.

Recursive Approach :

For every element in the array has two options, either we will include that element in subset or we don't include it.

- So if we take example as int[] A = { 3, 2, 7, 1}, S = 6
- If we consider another int array with the same size as A.
- If we include the element in subset we will put 1 in that particular index else put 0.
- So we need to make every possible subsets and check if any of the subset makes the sum as S.
- *If we think carefully this problem is quite similar to " **Generate All Strings of n bits**"*
- *See the code for better explanation.*
- **Time Complexity: O(2^n).**

Dynamic programming approach:

- Base Cases:
- If no elements in the set then we can’t make any subset except for 0.
- If sum needed is 0 then by returning the empty subset we can make the subset with sum 0.
- **Given** – Set = **arrA[]**, Size = **n**, sum = **S**
- Now for every element in he set we have 2 options, either we include it or exclude it.
- for any i^{th} element-
- If include it => S = S-arrA[i], n=n-1

If exclude it => S, n=n-1.

SUBSET SUM PROBLEM :

The problem is a classic problem it states that given a finite set S of positive integers and a integer target T>0 , we check whether there exists a subset S" ,sum of whose elements is T.

A new approach to address Subset Sum Problem.

IMPLEMENTATION

ALGORITHM 1: DYNAMIC PROGRAMMING

Code:

//Code implementation technique 1 using memorization and dynamic programming and recursion.....

```
#include <bits/stdc++.h>
Using namespace std;
// dp[i][j] is going to store true if sum j is
// possible with array elements from 0 to i. bool dp[100][100];
void display(const vector<int>& v)
{
for (int i = 0; i < v.size(); ++i)
printf("%d ", v[i]); printf("\
n");
}
// A recursive function to print all subsets with the
// help of dp[][]. Vector p[] stores current subset.
void printSubsetsRec(int arr[], int i, int sum, vector<int>& p)
{
// If sum becomes 0 if(sum == 0)
{
display(p); return;
}
If (i<=0 || sum<0) return;
// If given sum can be achieved after ignoring
// current element. if (dp[i-1][sum])
{
// Create a new vector to store path
vector<int> b = p;
```

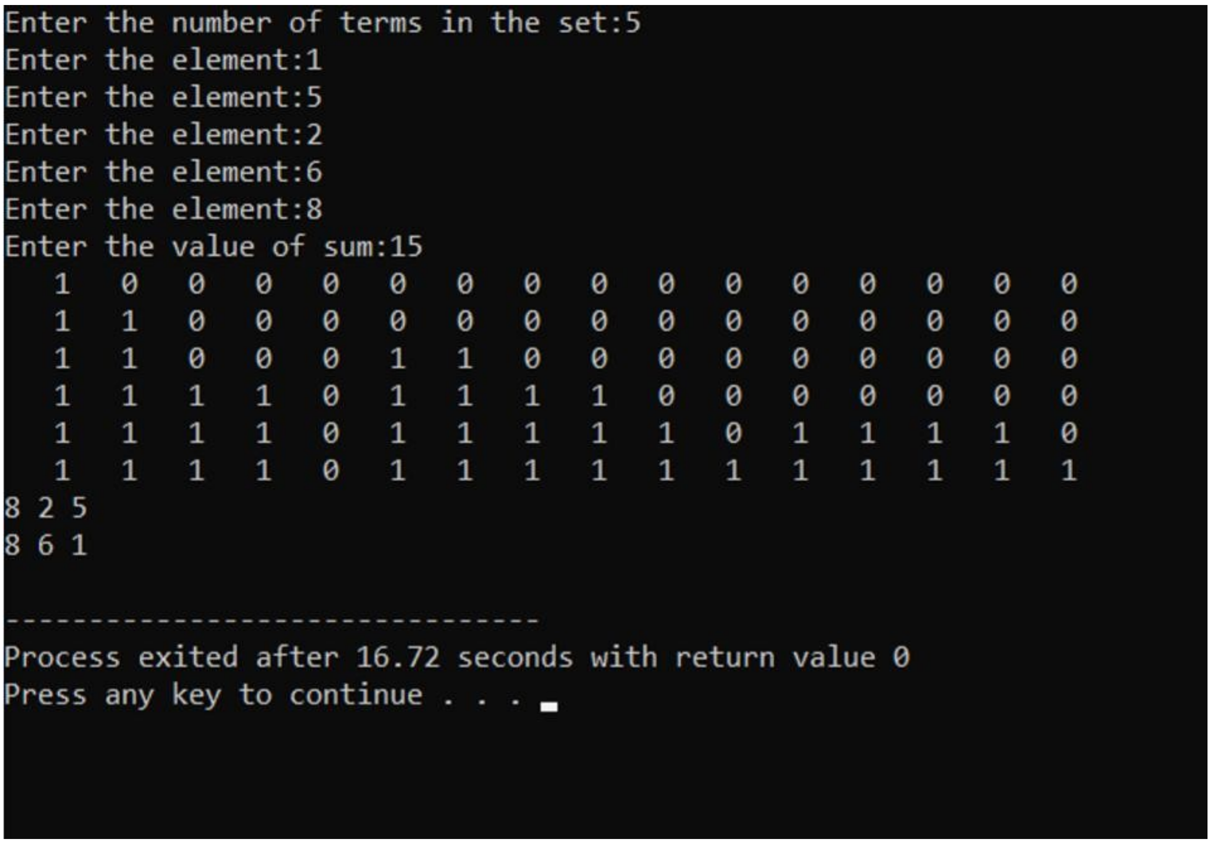
```
printSubsetsRec(arr, i-1, sum, p);
}
// If given sum can be achieved after considering
// current element.
if (sum >= arr[i-1] && dp[i-1][sum-arr[i-1]])
{
p.push_back(arr[i-1]);
printSubsetsRec(arr, i-1, sum-arr[i-1], p);
p.pop_back();
}
}
// Prints all subsets of arr[0..n-1] with sum 0.
 void printAllSubsets(int arr[], int n, int sum)
{
if (n == 0 || sum < 0) return;
// If sum is 0, then answer is true
for (int i = 0; i <= n; i++) dp[i]
[0] = true;
// If sum is not 0 and set is empty, then answer is false
for (int i = 1; i <= sum; i++)
dp[0][i] = false;
// Fill the subset table in botton up manner
for (int i = 1; i <= n; i++)
{
for (int j = 1; j <= sum; j++)
```

```
{
if(j<arr[i-1]) dp[i][j] = dp[i-1][j];
if (j >= arr[i-1])
dp[i][j] = dp[i-1][j] ||
dp[i - 1][j-arr[i-1]];
}
}
for (int i = 0; i <= n; i++)
{
for (int j = 0; j <= sum; j++)
{
printf ("%4d",dp[i][j]);
printf("\n");
}
if (dp[n][sum] == false)
{
printf("There are no subsets with sum %d\n", sum);
return;
}
// Now recursively traverse dp[][] to find all
// paths from dp[n-1][sum] vector<int> p;
printSubsetsRec(arr, n, sum, p);
}

int main()
```

```
{
int n,sum;
printf("Enter the number of terms in the set:");
scanf("%d",&n);
int arr[n];
for(int i=0;i<n;i++)
{
printf("Enter the element:");
scanf("%d",&arr[i]);
}
printf("Enter the value of sum:");
scanf("%d",&sum);
printAllSubsets(arr, n, sum);
return 0;
}
```

OUTPUT:



```
Enter the number of terms in the set:5
Enter the element:1
Enter the element:5
Enter the element:2
Enter the element:6
Enter the element:8
Enter the value of sum:15
   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   1   1   0   0   0   1   1   0   0   0   0   0   0   0   0   0
   1   1   1   1   0   1   1   1   1   0   0   0   0   0   0   0
   1   1   1   1   0   1   1   1   1   1   0   1   1   1   1   0
   1   1   1   1   0   1   1   1   1   1   1   1   1   1   1   1
8 2 5
8 6 1

--------------------------------
Process exited after 16.72 seconds with return value 0
Press any key to continue . . .
```

ALGORITHM 2: BACKTRACKING

CODE :

```
#include <stdio.h> #include <stdlib.h>
#define ARRAYSIZE(a) (sizeof(a))/(sizeof(a[0]))
static int total_nodes;
// prints subset found
void printSubset(int A[], int size)
{
for(int i = 0; i < size; i++)
{
printf("%*d", 5, A[i]);
}
printf("\n");
}
// inputs
// s   - set vector
// t   - tuplet vector
// s_size    - set size
// t_size    - tuplet size so far
// sum       - sum so far
// ite - nodes count
// target_sum - sum to be found
void subset_sum(int s[], int t[],
int s_size, int t_size, int sum, int ite,
int const target_sum)
{
total_nodes++;
if( target_sum == sum )
{
// We found subset
printSubset(t, t_size);
// Exclude previously added item and consider next candidate
subset_sum(s, t, s_size, t_size-1, sum - s[ite], ite + 1, target_sum);
return;
}
Else
```

```
{
// generate nodes along the breadth for( int i = ite; i < s_size; i++ )
{
t[t_size] = s[i];
// consider next level node (along depth)
subset_sum(s, t, s_size, t_size + 1, sum + s[i], i + 1, target_sum);
}
}
}
// Wrapper to print subsets that sum to target_sum
// input is weights vector and target_sum
void generateSubsets(int s[], int size, int target_sum)
{
int *tuplet_vector = (int *)malloc(size * sizeof(int));
subset_sum(s, tuplet_vector, size, 0, 0, 0, target_sum);
free(tuplet_vector);
}
int main()
{
int n;
printf("Enter the number of elements:"); scanf("%d",&n);
int weights[n]; for(int i=0;i<n;i++)
{
printf("enter the weight:"); scanf("%d",&weights[i]);
}
```

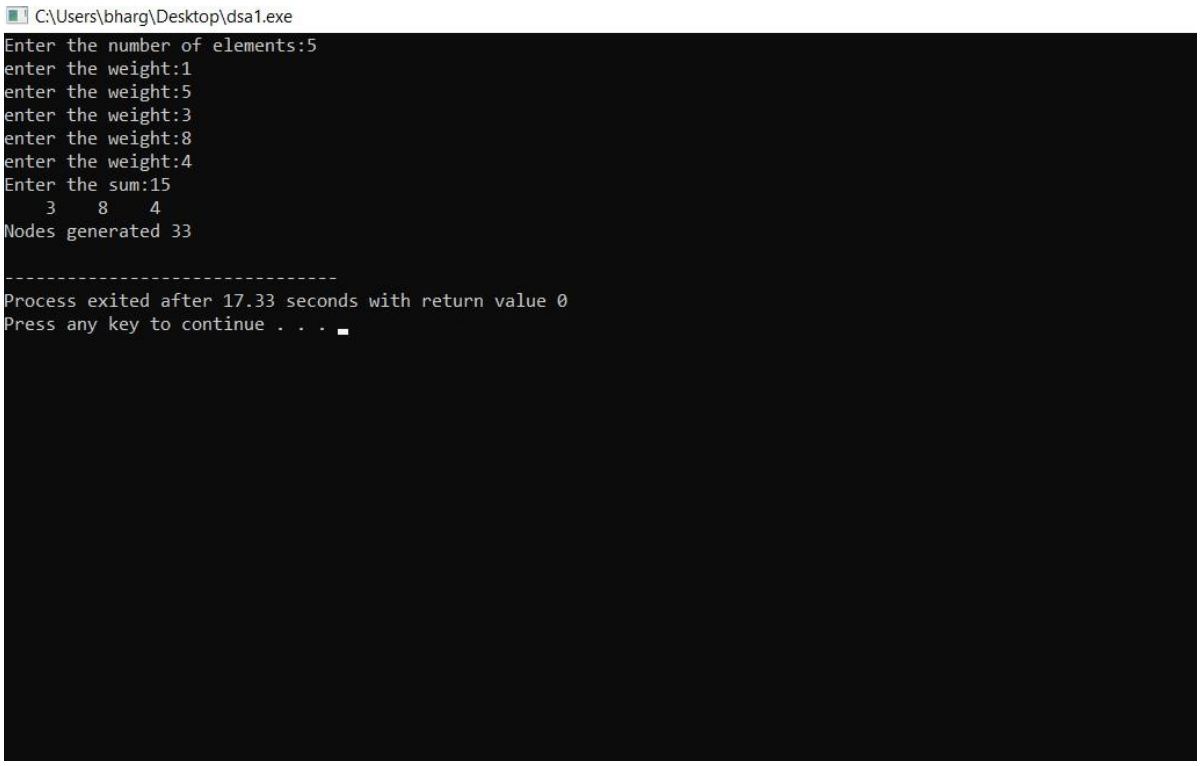
```
int size = ARRAYSIZE(weights); int sum;

printf("Enter the sum:"); scanf("%d",&sum);
generateSubsets(weights, size, sum);

printf("Nodes generated %d\n", total_nodes); return 0;

}
```

OUTPUT :



```
C:\Users\bharg\Desktop\dsa1.exe
Enter the number of elements:5
enter the weight:1
enter the weight:5
enter the weight:3
enter the weight:8
enter the weight:4
Enter the sum:15
    3    8    4
Nodes generated 33

--------------------------------
Process exited after 17.33 seconds with return value 0
Press any key to continue . . .
```

<u>**COMPARISION**</u> :

The time complexity of the dynamic programming algorithm is O(sum*n) where n is the input size and sum is the input that we have entered to check whether any combination of the set elements (subset) can generate the input.

The time complexity of backtracking algorithm is O(2^n) where n is the input size.

<u>GRAPH</u> :

X-AXIS: input size n or number of elements in the set Y-AXIS: time taken

<u>DYNAMIC PROGRAMMING ALGORITHM :</u>

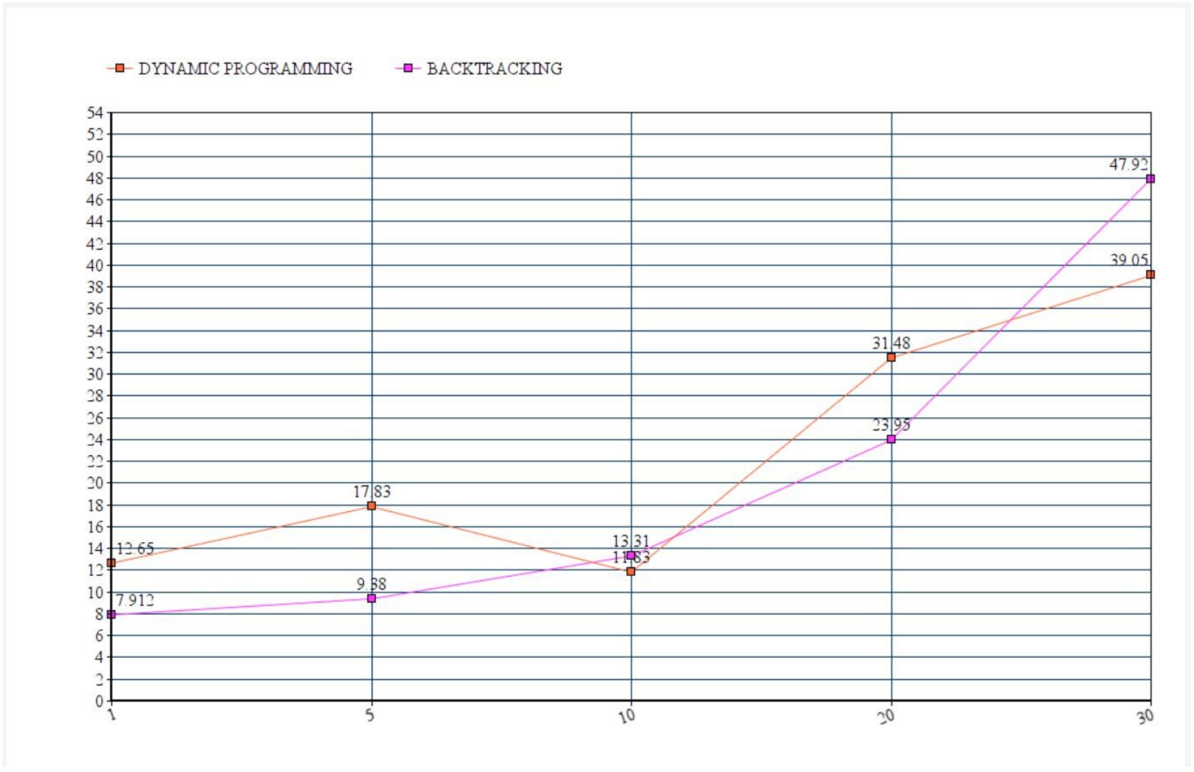
SIZE OF INPUT(n)	**TIME TAKEN (secs)**
1	**12.65**
5	**17.83**
10	**11.83**
20	**31.48**

30	**39.05**

BACKTRACKING ALGORITHM:

SIZE OF INPUT(n)	**TIME TAKEN(secs)**
1	**7.912**
5	**9.38**
10	**13.31**
20	**23.95**
30	**47.92**

GRAPH:


DYNAMIC PROGRAMMING
BACKTRACKING
12.65
17.83
11.83
31.48
39.05
7.912
9.38
13.31
23.95
47.92
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
52
54
1
5
10
20
30