**Experiment 2.2**

**Student Name: Sahul Kumar Parida UID: 20BCS4919**

**Branch: CSE Section/Group: WM-904/B**

**Semester: 5th Date of Performance: 27/09/2022**

**Subject Name: Design and Analysis Algorithm Lab**

**Subject Code: 20CSP-312**

1. **Aim/Overview of the practical:**

To implement a subset-sum problem using dynamic programming.

1. **Task to be done/which logistics used:**

Using Dynamic Programming

Dynamic Programming is a method for solving a complex problem by breaking it down into a collection of simpler sub problems, solving each of those sub problems just once, and storing their solutions using a memory-based data structure (array, map, etc).

1. **Algorithm/Flowchart:**

We will create a 2D array of size (arr.size() + 1) \* (target + 1) of type **boolean**. The state DP[i][j] will be **true** if there exists a subset of elements from A[0….i] with **sum value = ‘j’.** The approach for the problem is:

if (A[i-1] > j)

DP[i][j] = DP[i-1][j]

else

DP[i][j] = DP[i-1][j] OR DP[i-1][j-A[i-1]]

1. This means that if current element has value greater than ‘current sum value’ we will copy the answer for previous cases.
2. And if the current sum value is greater than the ‘ith’ element we will see if any of previous states have already experienced the **sum=’j’ OR any previous states experienced a value ‘j – A[i]’** which will solve our purpose.

1. **Steps for experiment/practical/Code:**

#include<iostream>

using namespace std;

bool subset\_sum(int a[], int n, int sum)

{

bool dp[n+1][sum+1];

int i,j;

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

dp[i][0] = true;

for(j=1;j<=sum;j++)

dp[0][j]=false;

for(i=1;i<=n;i++)

{

for(j=1;j<=sum;j++)

{

if(dp[i-1][j]==true)

dp[i][j]=true;

else

{

if(a[i-1]>j)

dp[i][j]=false;

else

dp[i][j]=dp[i-1][j-a[i-1]];

}

}

}

return dp[n][sum];

}

int main()

{

int i;

int n;

int sum;

cout<<"20BCS4919"<<endl;

cout<<"Sahul"<<endl;

cout<<"Enter the value of sum:"<<endl;

cin>>sum;

cout<<"Enter the number of elements in the set:"<<endl;

cin>>n;

int a[n];

cout<<"Enter the values:"<<endl;

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

cin>>a[i];

bool result=subset\_sum(a,n,sum);

if(result==true)

cout<<"subset with the given sum found";

else

cout<<"no required subset found";

cout<<endl;

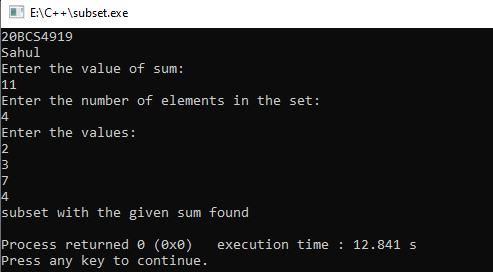
return 0;

}

1. **Observations/Discussions/ Complexity Analysis:**

**Time Complexity:** O(sum\*n), where sum is the ‘target sum’ and ‘n’ is the size of array.  
**Auxiliary Space:** O(sum\*n), as the size of 2-D array is sum\*n. + O(n) for recursive stack space

1. **Output:**



**Learning outcomes (What I have learnt):**

1. Dynamic Programming
2. To implement problems based on different algorithm design techniques.
3. To learn the importance of designing an algorithm in an effective way by considering space and time complexity.