AOA ASSIGNMENT-2

Subset Sum Problem

(using Dynamic Programming)

NAME: **Shawn Louis**

ROLL NO: **31**

BATCH: B

GROUP MEMBER: **Kevin Johnson**

ROLL NO: **25**

BATCH: **B**

PROBLEM STATEMENT:

Subset sum problem is to find if there exists a subset of elements that are selected from a given set whose sum adds up to a given number K. We are considering the set contains non-negative values. It is assumed that the input set is unique (no duplicates are presented)

OBJECTIVE:

The objective is to find if there exists any subsets in a given set that add up to the given sum.

OUTCOME:

The sum subset problem is solved and also verification if subsets are **present or not** is also done**.**

STRATEGY USED: DYNAMIC PROGRAMMING

DESCRIPTION OF STRATEGY:

1) Dynamic Programming is the most powerful design technique for solving optimization problems.

2) Dynamic Programming is used when the sub-problems are not independent, e.g. when they share the same sub-problems. In this case, divide and conquer may do more work than necessary, because it solves the same sub problem multiple times.

3) Dynamic Programming solves each sub-problems just once and stores the result in a table so that it can be repeatedly retrieved if needed again.

4) Dynamic Programming is a **Bottom-up approach-** we solve all possible small problems and then combine to obtain solutions for bigger problems.
5) Dynamic Programming is a paradigm of algorithm design in which an optimization problem is solved by a combination of achieving sub-problem solutions and appearing to the "**principle of optimality**".
6) The idea is to simply store the results of sub-problems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial

ALGORITHM:

Let isSubSetSum(int set[], int n, int sum) be the function to find whether there is a subset of set[] with sum equal to *sum*. n is the number of elements in set[]. The isSubsetSum problem can be divided into two sub-problems

…a) Include the last element, recur for n = n-1, sum = sum – set[n-1]

…b) Exclude the last element, recur for n = n-1.

If any of the above the above sub-problems return true, then return true

CODE:

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

int SubsetSum(int set[], int n, int sum);

int main()
{
int i, n, sum;
int set[10];
clrscr();

printf("\n\nEnter the number of elements : ");
scanf("%d", &n);
printf("\nEnter the elements in set : ");
for(i = 0; i < n; i++)
  scanf("%d", &set[i]);

printf("\nEnter the value of sum : ");
scanf("%d", &sum);

if (SubsetSum(set, n, sum) == 1)
      printf("\nFound a subset with given sum!!!");
else
```

```
      printf("\nNo subset with given sum!!!");
getch();
return 0;
}

int SubsetSum(int set[], int n, int sum)
{
      int subset[10][10];
      int i, j;

      for (i = 0; i <= n; i++)
      subset[i][0] = 1;

      for (i = 1; i <= sum; i++)
      subset[0][i] = 0;

      for (i = 1; i <= n; i++)
      {
      for (j = 1; j <= sum; j++)
      {
            if(j<set[i-1])
                  subset[i][j] = subset[i-1][j];
            if (j >= set[i-1])
                  subset[i][j] = subset[i-1][j] || subset[i - 1][j-
set[i-1]];
      }
      }
```

```
	printf("\nDP Table : \n")
	for (i = 0; i <= n; i++)
	{
		for (j = 0; j <= sum; j++)
		printf ("%4d", subset[i][j]);
	printf("\n");
	}

	return subset[n][sum];
}
```

Output Screenshot :

```
Enter the number of elements : 6

Enter the elements in set : 3 34 4 12 5 2

Enter the value of sum : 9

DP Table :
   1   0   0   0   0   0   0   0   0   0
   1   0   0   1   0   0   0   0   0   0
   1   0   0   1   0   0   0   0   0   0
   1   0   0   1   1   0   0   1   0   0
   1   0   0   1   1   0   0   1   0   0
   1   0   0   1   1   1   0   1   1   1
   1   0   1   1   1   1   1   1   1   1

Found a subset with given sum!!!
```