BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Name	Balla Mahadev Shrikrishna
UID no.	2023300010
Experiment No.	8

AIM:	To implement a Backtracking algorithm.

Program 1

PROBLEM STATEMENT :

Details – We are given n distinct positive numbers and find all combinations of these numbers whose sum are m using a Backtracking algorithm.
Input – Enter the weights of n number of elements and the integer m
Output – Print all possible subsets so that the sum is m.
Submission –
1) C/C++ source code of implementation
2) Verified output for the written source code with multiple inputs 3) One page report of Exp. 8

ALGORITHM:

➔

PROGRAM (subset.c):

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

int flag = 0;

void printsubset(int subset[], int size) {
  printf("[");
  for (int i = 0; i < size; i++)
    printf("%d ", subset[i]);
  printf("]\n");
}

void subsetsum(int index, int n, int set[], int targetsum, int subset[],
               int subsetsize) {
  if (targetsum == 0) {
    flag = 1;
    printsubset(subset, subsetsize);
```

```
    return;
  }

  if (index == n)
    return;

  if (set[index] <= targetsum) {
    subset[subsetsize] = set[index];
    subsetsum(index + 1, n, set, targetsum - set[index], subset,
              subsetsize + 1);
  }
  subsetsum(index + 1, n, set, targetsum, subset, subsetsize);
}

int main() {
  int n, targetsum;

  printf("Enter the number of elements in the set: ");
  scanf("%d", &n);

  int *set = (int *)malloc(n * sizeof(int));

  printf("Enter the elements of the set: ");
  for (int i = 0; i < n; i++) {
    scanf("%d", &set[i]);
  }

  printf("Enter the target sum: ");
  scanf("%d", &targetsum);

  int *subset = (int *)malloc(n * sizeof(int));

  printf("Subsets with sum %d:\n", targetsum);

  subsetsum(0, n, set, targetsum, subset, 0);

  return 0;
}
```

RESULT:

```
PS C:\Mahadev\SE\Sem4\DAA\Lab\Lab Sessions\exp8> gcc subset.c
PS C:\Mahadev\SE\Sem4\DAA\Lab\Lab Sessions\exp8> ./a.exe
Enter the number of elements in the set: 5
Enter the elements of the set: 1 2 3 4 5
Enter the target sum: 7
Subsets with sum 7:
[1 2 4 ]
[2 5 ]
[3 4 ]
PS C:\Mahadev\SE\Sem4\DAA\Lab\Lab Sessions\exp8> ./a.exe
Enter the number of elements in the set: 3
Enter the elements of the set: 1 2 3
Enter the target sum: 10
Subsets with sum 10:
PS C:\Mahadev\SE\Sem4\DAA\Lab\Lab Sessions\exp8> ./a.exe
Enter the number of elements in the set: 3
Enter the elements of the set: 1 2 3
Enter the target sum: 0
Subsets with sum 0:
[]
PS C:\Mahadev\SE\Sem4\DAA\Lab\Lab Sessions\exp8> ./a.exe
Enter the number of elements in the set: 4
Enter the elements of the set: 1 2 3 4
Enter the target sum: 10
Subsets with sum 10:
[1 2 3 4 ]
```

ANALYSIS:

Algorithm -

Step 1: Input

- ➔ Read the number of elements n.
- ➔ Allocate and read n integers into an array set.
- ➔ Read the targetsum.

Step 2: Initialize

- ➔ Allocate an array subset of size n to store temporary subsets.
- ➔ Set global flag = 0 to track if any subset has been found.

Step 3: Call subsetsum()

- ➔ subsetsum(0, n, set, targetsum, subset, 0);
- ➔ Start from index 0 with full target sum and empty subset.

Step 4: Inside subsetsum()

For each call:

- ➔ Base Case 1: If targetsum == 0 → valid subset found
    - ◆ Print subset

<table>
<tr>
<td></td>
<td>
Set flag = 1
Return
➔ Base Case 2: If index == n → End of array reached
Return
➔ Recursive Step:
If set[index] <= targetsum, include set[index] in subset, and recurse:
subsetsum(index + 1, ..., targetsum - set[index], ..., subsetsize + 1)
Exclude set[index], recurse:
subsetsum(index + 1, ..., targetsum, ..., subsetsize)
Step 5: Output
➔ All subsets that add up to targetsum are printed.
Time Complexity Analysis -
Worst Case:
➔ Every element can be included or excluded → 2^n combinations.
➔ Each valid subset might take up to O(n) time to print.
➔ So, the total worst-case time complexity is: O(n * (2^n))
Best Case:
➔ If targetsum is very small or elements are large → few paths are explored due to pruning.
Space Complexity Analysis -
➔ subset[]: size n → O(n)
➔ Recursive call stack: max depth n → O(n)
</td>
</tr>
<tr>
<td>CONCLUSION:</td>
<td>In this experiment, I implemented a recursive backtracking solution to the Subset Sum Problem, where I explored all subsets of a set that sum to a target value. Through this, I learned how backtracking helps systematically explore combinations, but also realized the time complexity can grow exponentially with larger inputs. This reinforced the importance of optimizing recursive algorithms for efficiency.</td>
</tr>
</table>