Experiment No. : 7

Title: Sum of Subset problem using Backtracking

Batch: B1 **Roll No.:16010420133** **Experiment No.: 7**

Aim: To Implement Sum of Subset problem using Backtracking.

Algorithm of Sum of Subset problem:

Algorithm SUB_SET_PROBLEM(i, sum, W, remSum)
Description : Solve sub of subset problem using backtracking
Input : W: Number for which subset is to be computed
i: Item index
sum : Sum of integers selected so far
remSum : Size of remaining problem i.e. (W – sum)

Output : Solution tuple X

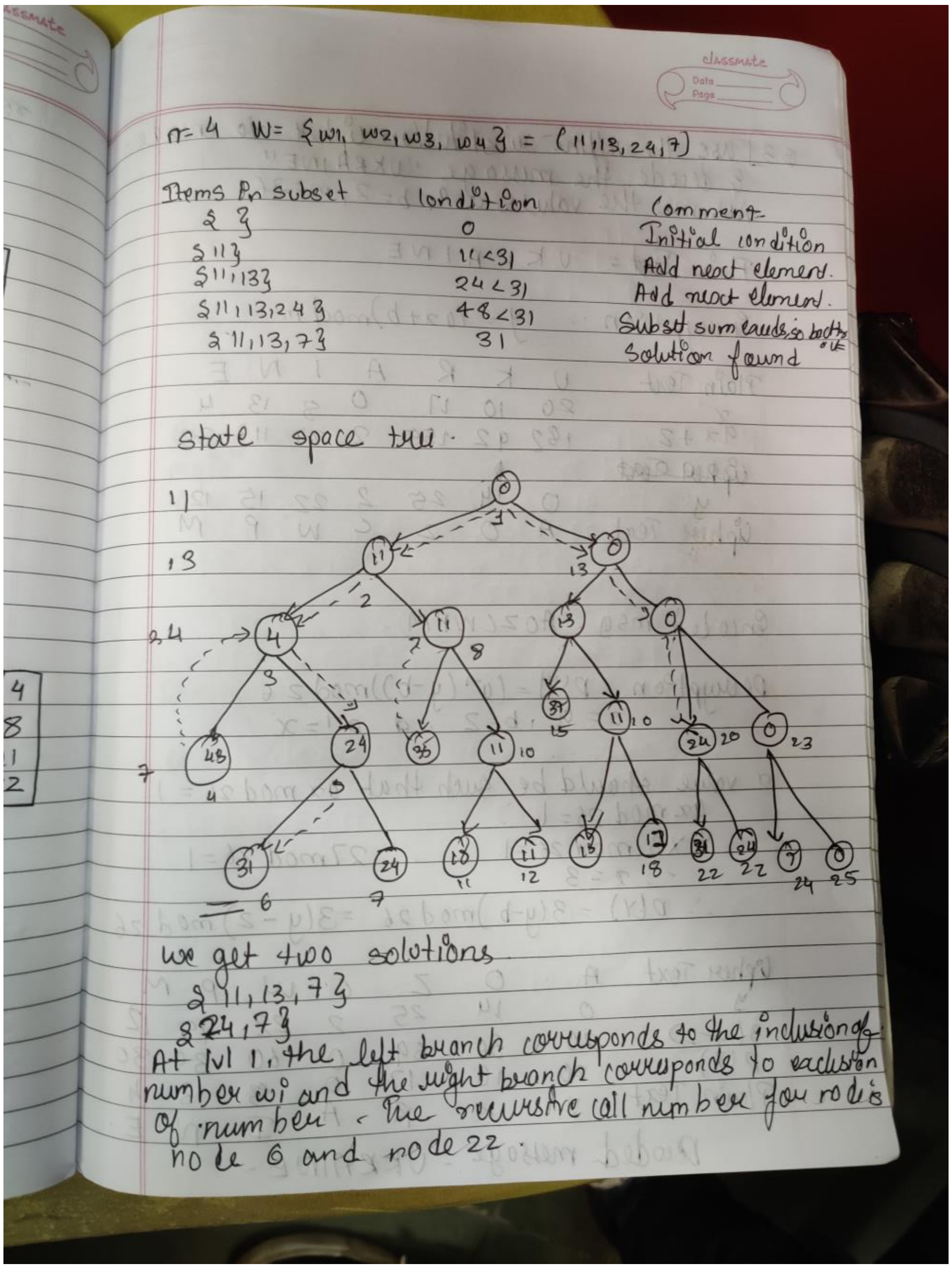
```
1.  if FEASIBLE_SUB_SET(i) == 1 then
2.  if (sum == W) then
3.  print X[1…i]
4.  end
5.  else
6.  X[i + 1] ← 1
7.  SUB_SET_PROBLEM(i + 1, sum + w[i] + 1, W, remSum – w[i] + 1 )
8.  X[i + 1] ← 0  // Exclude the ith item
9.  SUB_SET_PROBLEM(i + 1, sum, W, remSum – w[i] + 1 )
10. end

11. function FEASIBLE_SUB_SET(i)
12. if (sum + remSum ≥ W) AND (sum == W) or (sum + w[i] + 1 ≤ W) then
13. return 0
14. end
15. return 1
```

Working of Sum of Subset problem:

Problem Statement

Problem: Consider the sum-of-subset problem, n = 4 W =(w1, w2, w3, w4) = (11, 13, 24, 7) and M = 31. Find a solution to the problem using backtracking. Show the state-space tree leading to the solution. Also, number the nodes in the tree in the order of recursion calls.

Solution



n= 4 W= {w1, w2, w3, w4} = (11,13,24,7)

Items in subset	condition	Comment
{ }	0	Initial condition
{11}	11<31	Add next element.
{11,13}	24<31	Add next element.
{11,13,24}	48<31	Subset sum exceeds, so backtrack
{11,13,7}	31	Solution found

state space tree.

0
11
0
4
11
13
0
48
24
35
11
31
11
24
0
31
24
18
11
13
17
31
24
7
0

we get two solutions

{11,13,7}

{24,7}

At lvl 1, the left branch corresponds to the inclusion of number wi and the right branch corresponds to exclusion of number. The recursive call number for node is node 6 and node 22.

Derivation of Sum of Subset problem:

Time complexity Analysis

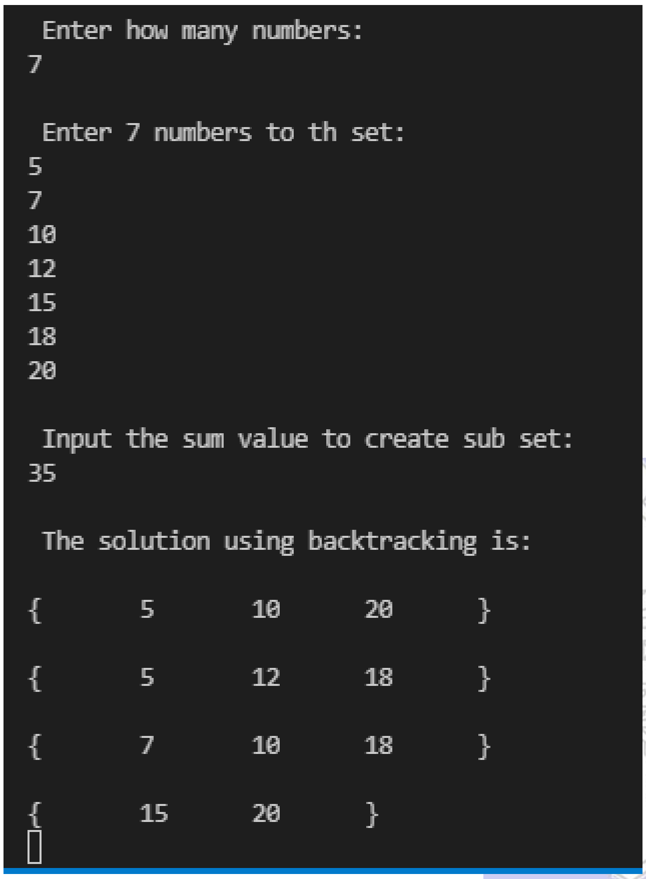
It is intuitive to derive the complexity of sum of the subset problem. In the state-space tree, at level i, the tree has 2i nodes. So, given n items, the total number of nodes in the tree would be 1 + 2 + 22 + 23 + .. 2n.

T(n) = 1 + 2 + 22 + 23 + .. 2n = 2n+1 – 1 = O(2n)

Program(s) of Sum of Subset problem:

```
#include<stdio.h>
#include<conio.h>
#define TRUE 1
#define FALSE 0
int inc[50],w[50],sum,n;
void sub_Set_problem(int i,int wt,int total);
int feasible_sub_set(int i,int wt,int total) {
    return(((wt+total)>=sum)&&((wt==sum)||(wt+w[i+1]<=sum)));
}


void main() {
    int i,j,n,temp,total=0;
    printf("\n Enter how many numbers:\n");
    scanf("%d",&n);
    printf("\n Enter %d numbers to th set:\n",n);
    for (i=0;i<n;i++) {
        scanf("%d",&w[i]);
        total+=w[i];
    }
    printf("\n Input the sum value to create sub set:\n");
    scanf("%d",&sum);


    if((total<sum))
      printf("\n Subset construction is not possible"); else {
        for (i=0;i<n;i++)
           inc[i]=0;
        printf("\n The solution using backtracking is:\n");
        sub_Set_problem(-1,0,total);
    }
    getch();
}
void sub_Set_problem(int i,int wt,int total) {
    int j;
    if(feasible_sub_set(i,wt,total)) {
        if(wt==sum) {
            printf("\n{\t");
            for (j=0;j<=i;j++)
                if(inc[j])
                 printf("%d\t",w[j]);
            printf("}\n");
        } else {
            inc[i+1]=TRUE;
            sub_Set_problem(i+1,wt+w[i+1],total-w[i+1]);
            inc[i+1]=FALSE;
            sub_Set_problem(i+1,wt,total-w[i+1]);
        }
    }
}
```

Output(o) of Sum of Subset problem:



```
 Enter how many numbers:
7

 Enter 7 numbers to th set:
5
7
10
12
15
18
20

 Input the sum value to create sub set:
35

 The solution using backtracking is:

{       5       10      20      }

{       5       12      18      }

{       7       10      18      }

{       15      20      }
```

Post Lab Questions:- Explain the process of backtracking ? What are the advantages of backtracking as against brute force algorithms ?

Ans: Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree).
Advantages:

1. Simple to implement.
2. State changes are stored in stack, meaning we do not need to concern ourselves about them.
3. Intuitive approach of trial and error.
4. Code size is usually small.

Conclusion: (Based on the observations):

CO4: Understand Backtracking and Branch-and-bound algorithms.

Outcome:

Thus, we learnt the sum of subsets using backtracking and implemented the same.

References:

1. Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones & Bartlett Students Edition
2. Harsh Bhasin , " Algorithms : Design & Analysis", 1st Edition 2013, Oxford Higher education, India
3. T.H. Coreman ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication
4. Jon Kleinberg, Eva Tardos, " Algorithm Design", 10th Edition 2013, Pearson India Education Services Pvt. Ltd.