**Batch: B-1              Roll No.: 16010122104**

**Experiment No. 8**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title: Implementation of sum of subset Algorithm** |

**Objective:** To learn the Backtracking strategy of problem solving for Sum of subset

**CO to be achieved:**

|  |  |
| --- | --- |
| CO 2 | Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies. |

**Books/ Journals/ Websites referred:**

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran,” Fundamentals of computer algorithm”, University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein,” Introduction to algortihtms”,2nd Edition ,MIT press/McGraw Hill,2001**

**Pre Lab/ Prior Concepts:**

Data structures, Concepts of algorithm analysis

**Historical Profile:**

Subset sum problem is to find 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).

One way to find subsets that sum to K is to consider all possible subsets. A [power set](http://en.wikipedia.org/wiki/Power_set) contains all those subsets generated from a given set. The size of such a power set is 2N.

***Input:***

A vector X={x1,x2… xn} for all n elements in the set where Xi=0 (element not added) or xi=1 (element added in the solution tuple).

***Output:***

Summation of the chosen numbers must be equal to given number M and one number can be used only once.

BACKTRACKING CONDITION

Diagram, letter, schematic

Description automatically generated

**New Concepts to be learned:**

Application of algorithmic design strategy to any problem, Backtracking method of problem solving Vs other methods of problem solving problem  sum of subset and its applications.

**Algorithm:**

Algorithm sumOfSub(s, k, r)

{//It is assumed w[1]<=m and Sigma(i=1 to m)w[i]>=m

//generate the left child. Note: s+w(k)<=M since Bk-1 is true.

X{k]=1;

if (S+W[k]=m) then write(X[1:k]);  //Subset found. there is no recursive call here

as W[j]>0,1<=j<=n.

else if (S+W[k]+W[k+1]<=m) then sumOfSub(S+W[k], k+1,r- W[k]); //moving to

next sub-problem.

Similarly, assume the array is presorted and we found one subset. We can generate

next node excluding the present node only when inclusion of next

node satisfies the constraints.

if ((S+ r- W[k]>=m)and (S+ W[k+1]<=m)) then//generate right {

//child and those satisfying 2 bounding functions

X{k]=0;

sumOfSub (S, k+1, r- W[k]);

}}

**Implementation(Code):**

package Backtracking;

import java.util.\*;

public class sumofsubset {

public static Boolean sumOfSubset(int[] arr, int[] res, int sum, int index){

if(sum==0){

for (int i : res) {

if(i!=-1 && i!=0)

System.out.print(i+" ");

}

System.out.println();

return true;

}else if(sum<0){

return false;

}else if(arr.length==0 && sum!=0){

return false;

}else if(index==arr.length){

return false;

}else{

int temp = arr[index];

res[index] = temp;

Boolean b = sumOfSubset(arr, res, sum-temp, index+1);

res[index] = -1;

b = sumOfSubset(arr, res, sum, index+1);

return b;

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the size of the array: ");

int size = sc.nextInt();

int arr[] = new int[size];

for (int i=0; i<size; i++) {

System.out.print("Enter element "+(i+1)+": ");

arr[i] = sc.nextInt();

}

System.out.print("Enter the target sum: ");

int sum = sc.nextInt();

// int arr[] = {2,3,4,5};

// int sum = 9;

// int res[] = new int[arr.length];

int res[] = new int[size];

int index = 0;

sumOfSubset(arr, res, sum, index);

sc.close();

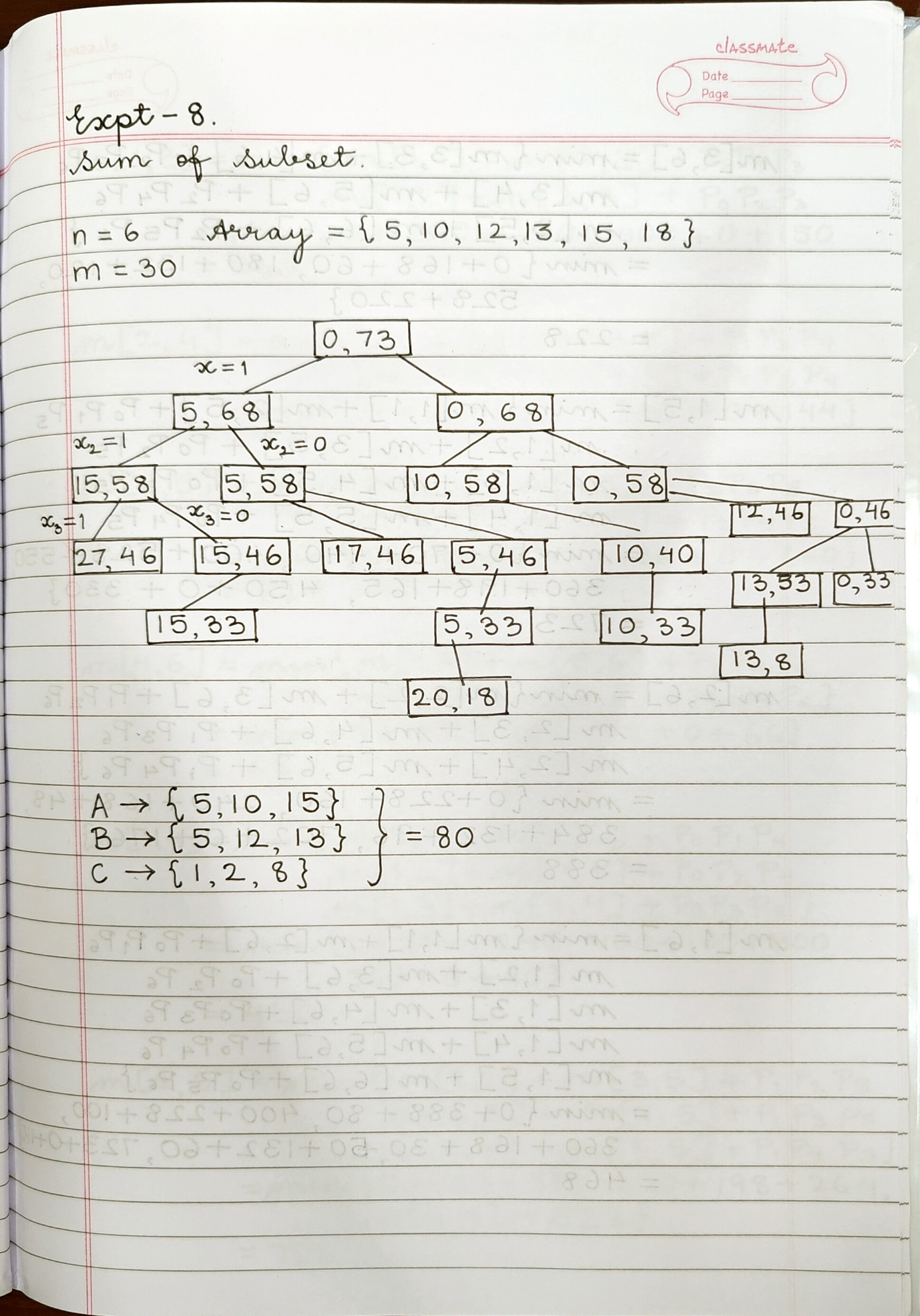
}

}

**Output:**



**Example sum of subset Problem along with state space tree:**



**Analysis of Backtracking solution for sum of subset Problem:**

Time Complexity: The above approach may try all the possible subsets of a given array in the worst case. Therefore the time complexity of the above approach is exponential i.e. O(2n ). 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 + 2 2 + 2 3 + .. 2 n = 2n+1 – 1 = O(2 n ) →Space Complexity: O(n) for the recursion stack

**Conclusion:**

Through this experiment we implemented sum of subset problem using backtracking strategy. Subset sum problem is to find 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.