**Name:** Vishal Shashikant Salvi.

**UID:** 2019230069

**Class:** SE Comps

**Batch:** C

**Experiment No 3**

**Aim**: Sum of subset problem using backtracking.

**Theory**:

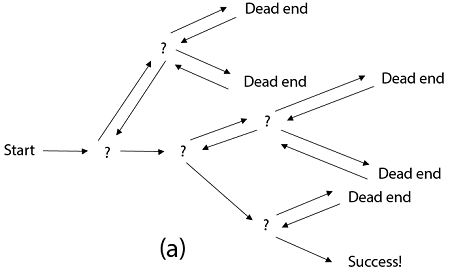
**Introduction of Backtracking**

The Backtracking is an algorithmic-method to solve a problem with an additional way. It uses a recursive approach to explain the problems. We can say that the backtracking is needed to find all possible combination to solve an optimization problem.

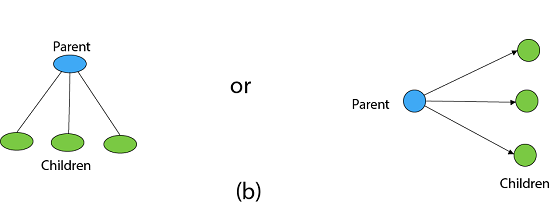
**Backtracking** is a systematic way of trying out different sequences of decisions until we find one that "works."

In the following Figure:

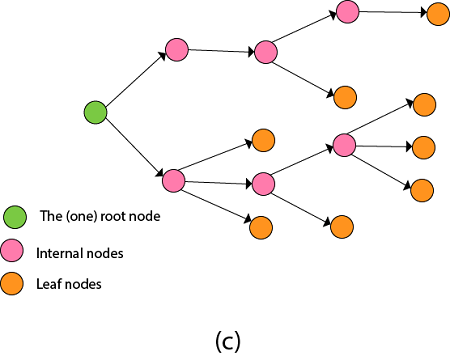
* Each non-leaf node in a tree is a parent of one or more other nodes (its children)
* Each node in the tree, other than the root, has exactly one parent



Generally, however, we draw our trees downward, with the root at the top.



A tree is composed of nodes.



**Backtracking can understand of as searching a tree for a particular "goal" leaf node.**

Backtracking is undoubtedly quite simple - we "explore" each node, as follows:

**To "explore" node N:**

1. If N is a goal node, return "success"

2. If N is a leaf node, return "failure"

3. For each child C of N,

Explore C

If C was successful, return "success"

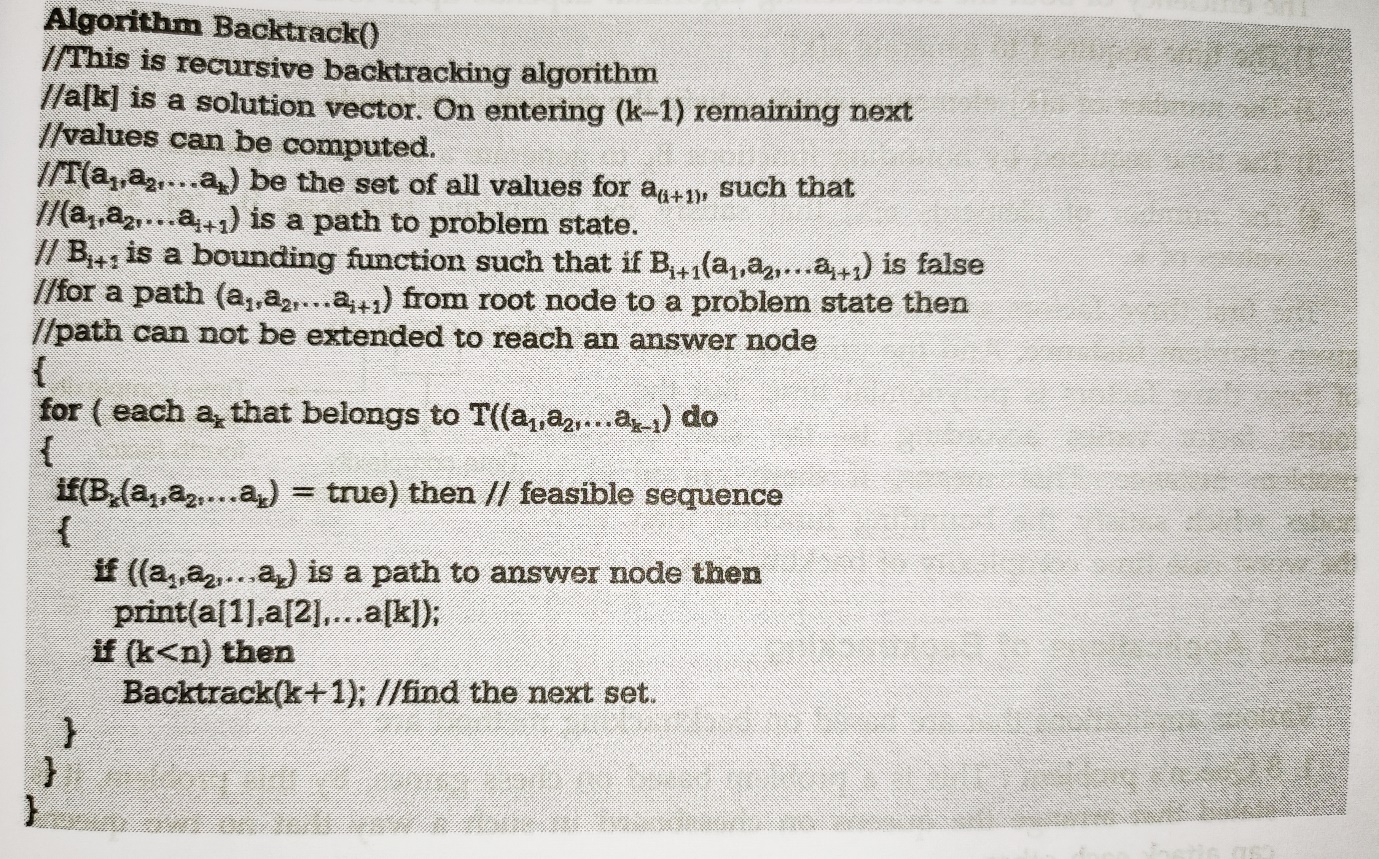
4. Return "failure"

Backtracking algorithm determines the solution by systematically searching the solution space for the given problem. Backtracking is a **depth-first search** with any bounding function. All solution using backtracking is needed to satisfy a complex set of constraints. The constraints may be explicit or implicit.

**Explicit Constraint** is ruled, which restrict each vector element to be chosen from the given set.

**Implicit Constraint** is ruled, which determine which each of the tuples in the solution space, actually satisfy the criterion function.

**Algorithm:**



**Sum of subset Problem**

The Subset-Sum Problem is to find a subset's' of the given set S = (S1 S2 S3...Sn) where the elements of the set S are n positive integers in such a manner that s'∈S and sum of the elements of subset's' is equal to some positive integer 'X.'

The Subset-Sum Problem can be solved by using the backtracking approach. In this implicit tree is a binary tree. The root of the tree is selected in such a way that represents that no decision is yet taken on any input. We assume that the elements of the given set are arranged in increasing order:

S1 ≤ S2 ≤ S3... ≤ Sn

The left child of the root node indicated that we have to include 'S1' from the set 'S' and the right child of the root indicates that we have to execute 'S1'. Each node stores the total of the partial solution elements. If at any stage the sum equals to 'X' then the search is successful and terminates.

The dead end in the tree appears only when either of the two inequalities exists:

* The sum of s' is too large i.e.

s'+ Si + 1 > X

* The sum of s' is too small i.e.

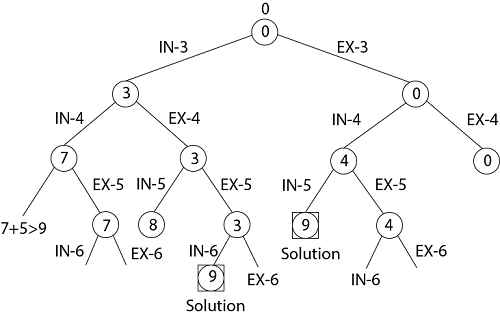
Subset-Sum Problem

**Example:** Given a set S = (3, 4, 5, 6) and X =9. Obtain the subset sum using Backtracking approach.

**Solution:**

1. Initially S = (3, 4, 5, 6) and X =9.
2. S'= (∅)

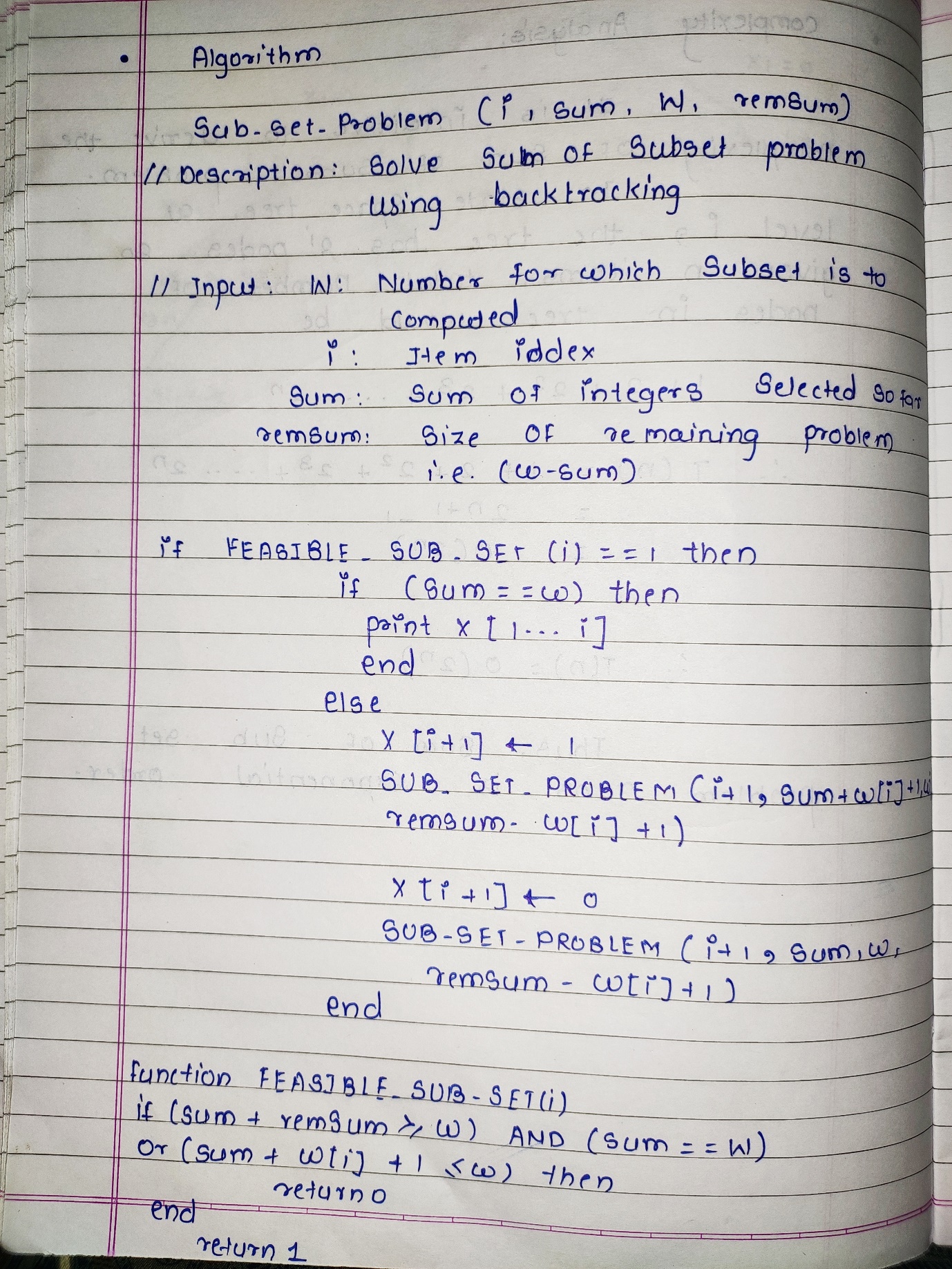
The implicit binary tree for the subset sum problem is shown as fig:



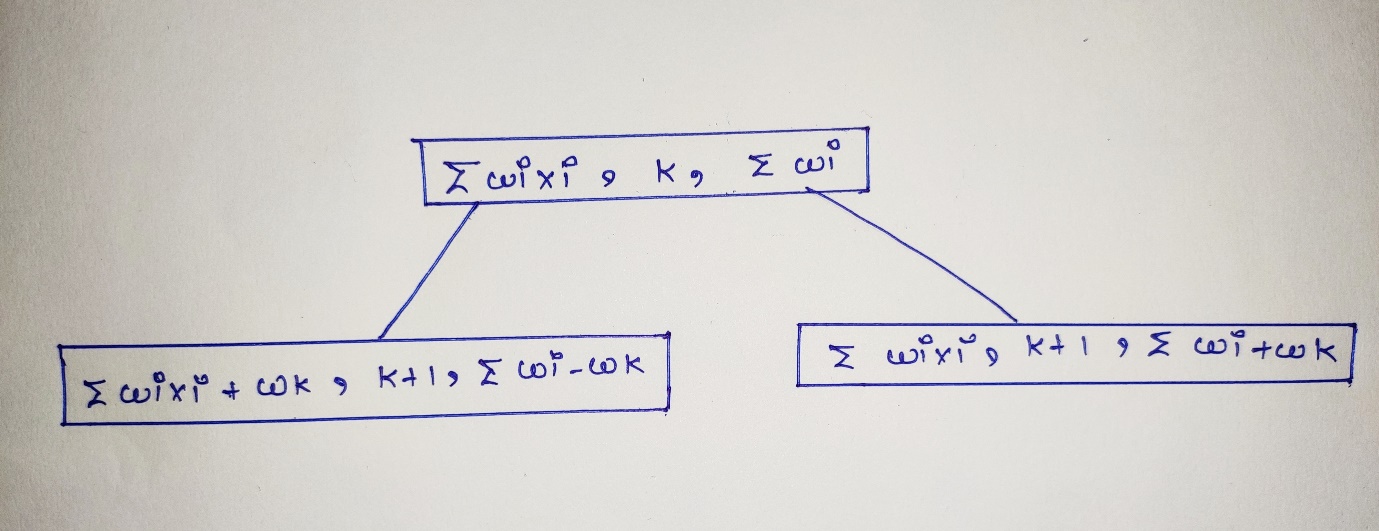
The number inside a node is the sum of the partial solution elements at a particular level.

Thus, if our partial solution elements sum is equal to the positive integer 'X' then at that time search will terminate, or it continues if all the possible solution needs to be obtained.

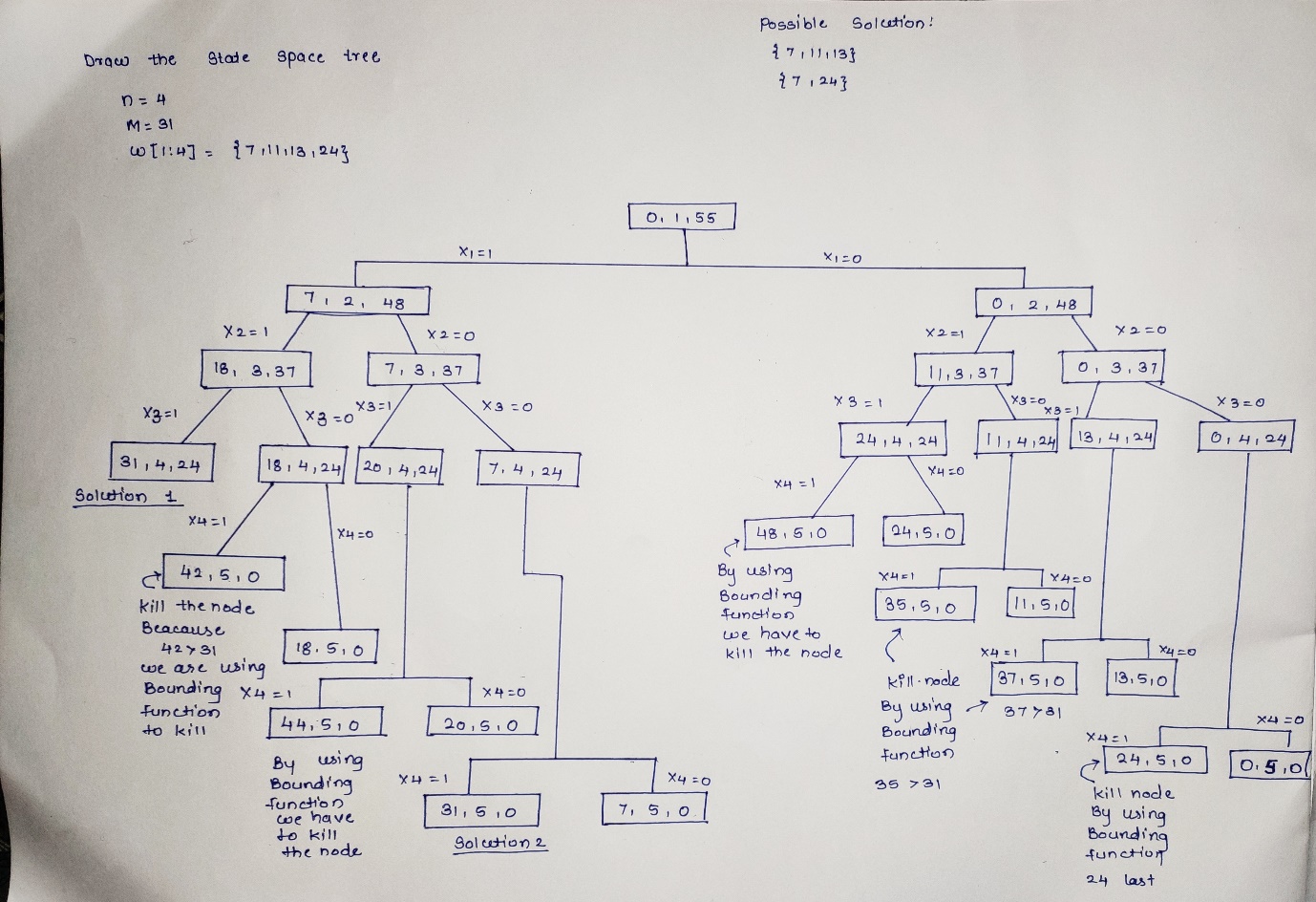
**Algorithm:**



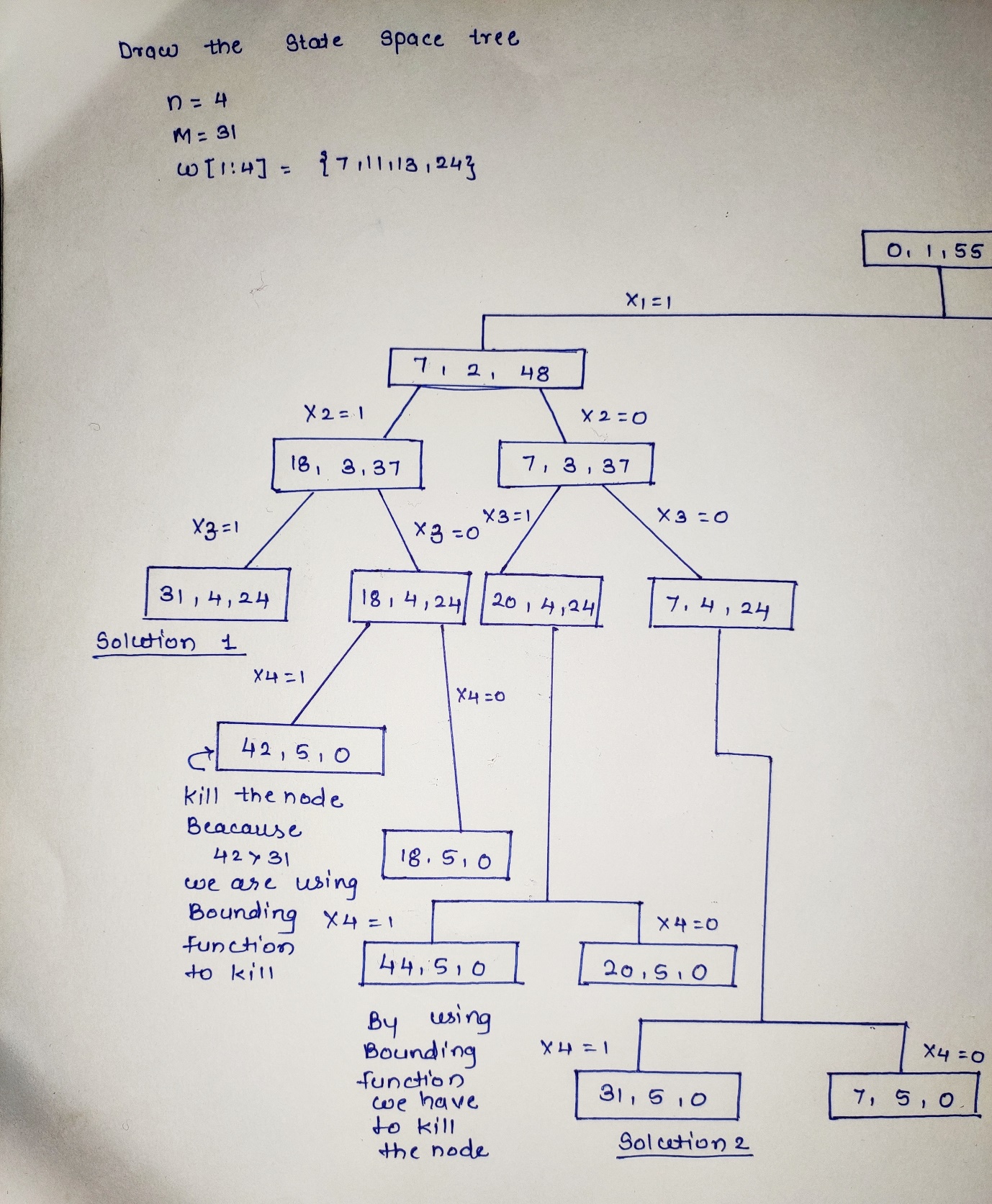
**Mathematical Derivation:**



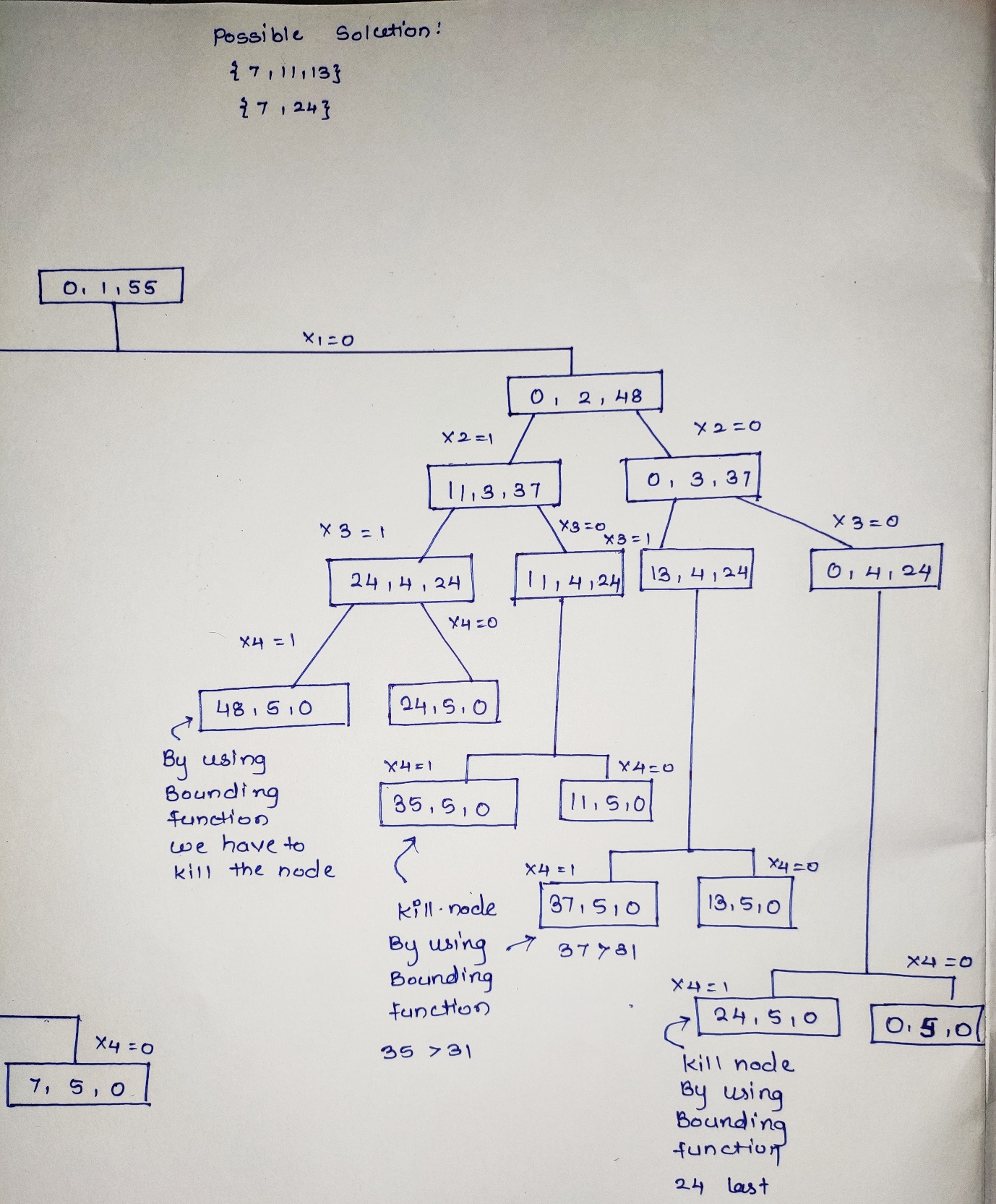
**Problem State space tree:**



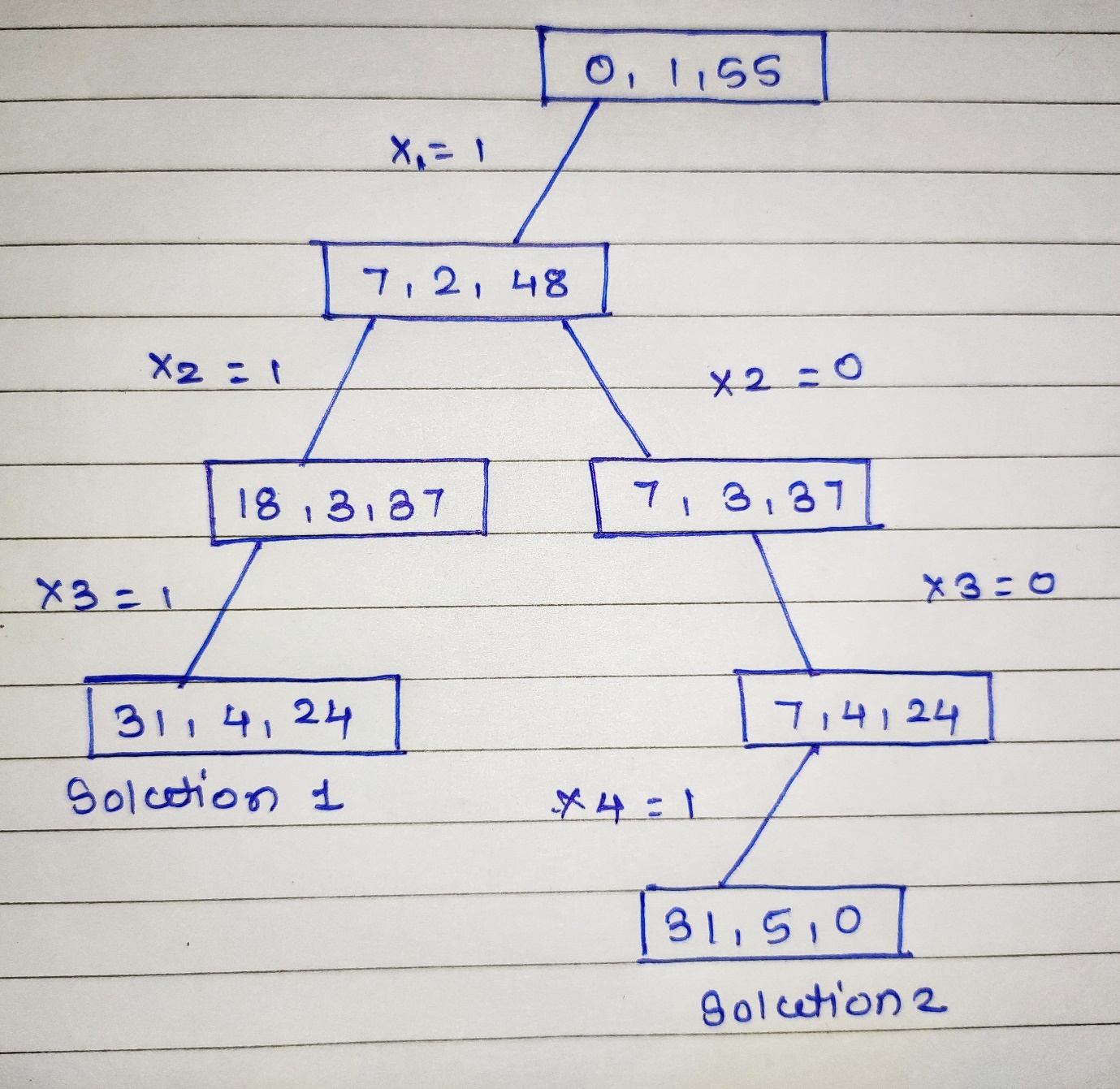
**Right State space tree:**



**Left State space tree:**

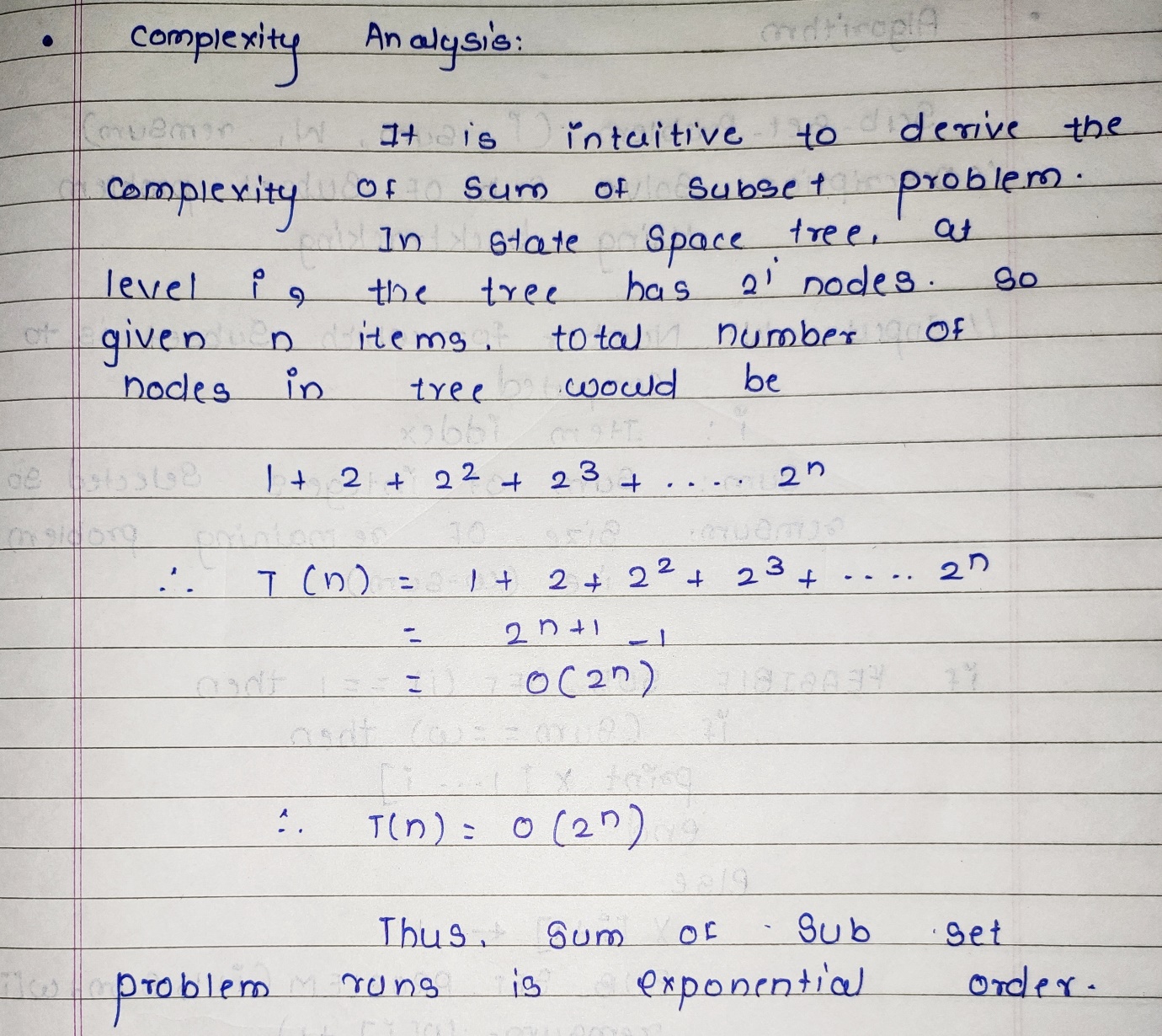


**Corresponding tree:**



**Complexity Analysis:**

The time complexity of Sum of Subset problem is O(2n).



**Code:**

#include<stdio.h>

#define TRUE 1

#define FALSE 0

int inc[100],w[100],sum,n;

int feasible\_sub\_set(int i,int wt,int total) {

return(((wt+total)>=sum)&&((wt==sum)||(wt+w[i+1]<=sum)));

}

void subsetproblem(int i,int wt,int total) {

int j;

if(feasible\_sub\_set(i,wt,total)) {

if(wt==sum) {

printf("\n{");

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

if(inc[j])

printf("%d ",w[j]);

printf("}\n");

} else {

inc[i+1]=TRUE;

subsetproblem(i+1,wt+w[i+1],total-w[i+1]);

inc[i+1]=FALSE;

subsetproblem(i+1,wt,total-w[i+1]);

}

}

}

int main() {

int i,j,n,temp,total=0;

printf("\nEnter Number of Elements in set:\n");

scanf("%d",&n);

printf("\nEnter %d numbers:\n",n);

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

scanf("%d",&w[i]);

total+=w[i];

}

printf("\nEnter the sum:\n");

scanf("%d",&sum);

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

for (j=0;j<n-1;j++)

if(w[j]>w[j+1]) {

temp=w[j];

w[j]=w[j+1];

w[j+1]=temp;

}

printf("\nThe given numbers in ascending order:\n");

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

printf("%d \t",w[i]);

if((total<sum))

printf("\nSubset are not possible");

else {

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

inc[i]=0;

printf("\n All Possible solutions for sum of subset:\n");

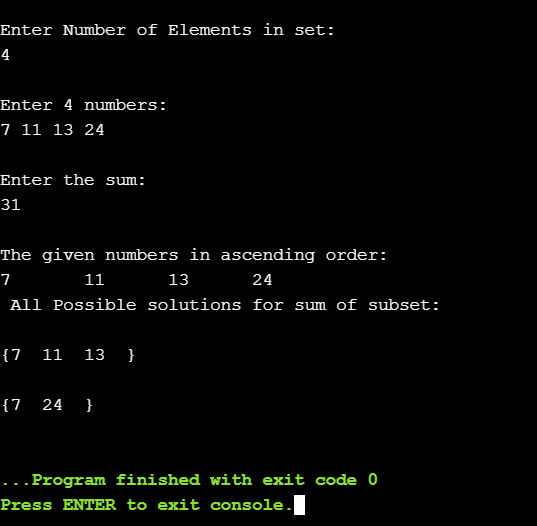
subsetproblem(-1,0,total);

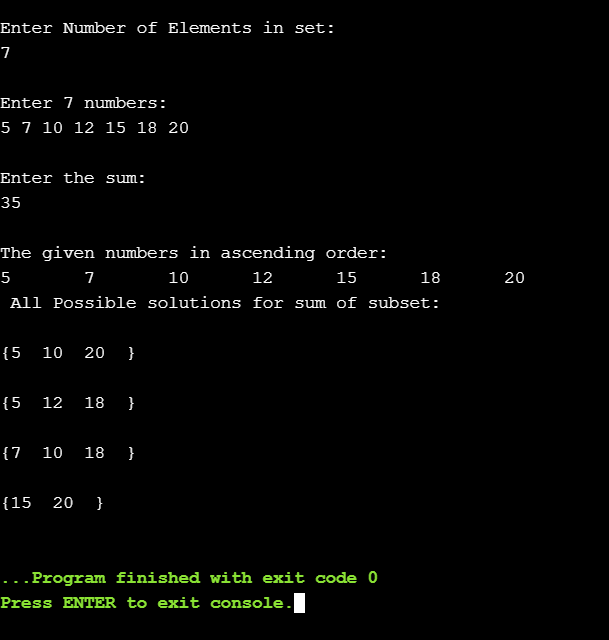
}

return 0;

}

**Output:**





## **Conclusion**:

Backtracking is an algorithm for capturing some or all solutions to given computational issues, especially for constraint satisfaction issues. The algorithm can only be used for problems which can accept the concept of a “partial candidate solution” and allows a quick test to see if the candidate solution can be a complete solution. Backtracking is considered an important technique to solve constraint satisfaction issues and puzzles. It is also considered a great technique for parsing and also forms the basis of many logic programming languages. We solve sum of subset by using backtracking algorithm.