**Topic: Backtracking**

**Theory:** In many applications of the backtrack method, the desired solution is expressible as an n-tuple *(x1,...,Xn),* where the x*i* are chosen from some finite set Si. Often the problem to be solved calls for finding one vector that maximizes (or minimizes or satisfies) a *criterion function P(x1,…..* . , *xn). Sometime*s it seeks all vectors that satisfy *P.* For example, sorting the array of integers in. *a[1* : n] is a problem whose solution is expressible by an *n- tuple, w*here x*i* is the index in *a* of the ith smallest element. The criterion function P is the inequality *a[xi]* ≤ *a[xi+1]* for 1 ≤ i < *n.* The set *Si* is finite and includes the integers 1 through *n.* Though sorting is not usually one of the problems solved by backtracking, it is one example of a familiar problem whose solution can be formulated as an n-tuple.

**Control abstraction**:

void Backtrack( int k )

// This is a schema that describes the backtracking process //using recursion. On entering, the first k-1 values x[1], x[2], //…., x[k-1] of the solution vector x[1:n] have been //assigned. x[] and n are global.

{

for (each x[k] such that x[k] Є T(x[1], …, x[k-1])

{

if (Bk (x[1], x[2], …, x[k]))

{

if (x[1], x[2], …, x[k] is a path to an answer node)

output x[1:k];

if (k < n) Backtrack(k+1);

}

}

}



| **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):**

**#include <bits/stdc++.h>**

**using namespace std;**

**void func(int i, int n, vector<int> &v, int k, vector<int> &temp)**

**{**

**if(k < 0)return;**

**if(k == 0)**

**{**

**cout << "Elements are: \n";**

**for(auto &i: temp)cout << i << " ";**

**cout << "\n";**

**return;**

**}**

**if(i == n)return;**

**func(i + 1, n, v, k, temp);**

**temp.push\_back(v[i]);**

**func(i + 1, n, v, k - v[i], temp);**

**temp.pop\_back();**

**}**

**int main()**

**{**

**int n;**

**cout << "Enter the size of the array\n";**

**cin >> n;**

**vector<int> v(n);**

**cout << "Enters the elements\n";**

**for(auto &i: v)cin >> i;**

**cout << "Enter the target element\n";**

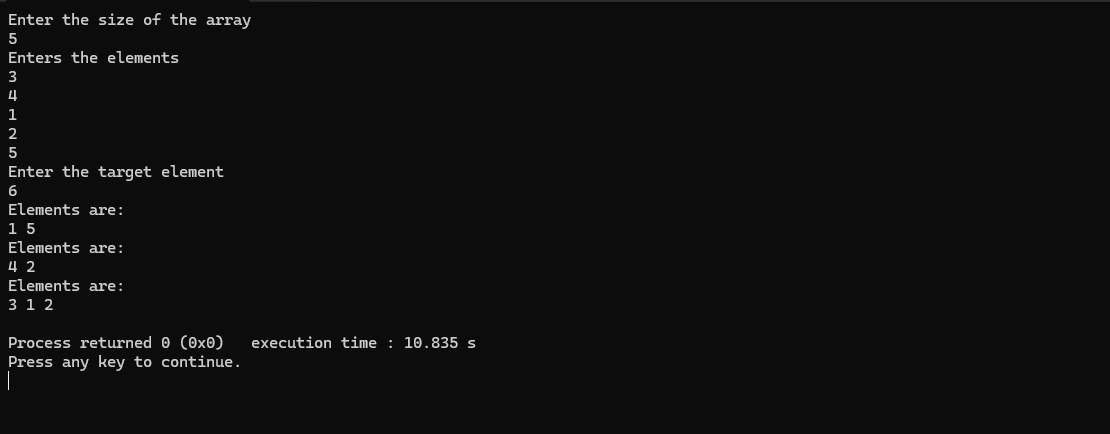
**int k; cin >> k;**

**vector<int> temp;**

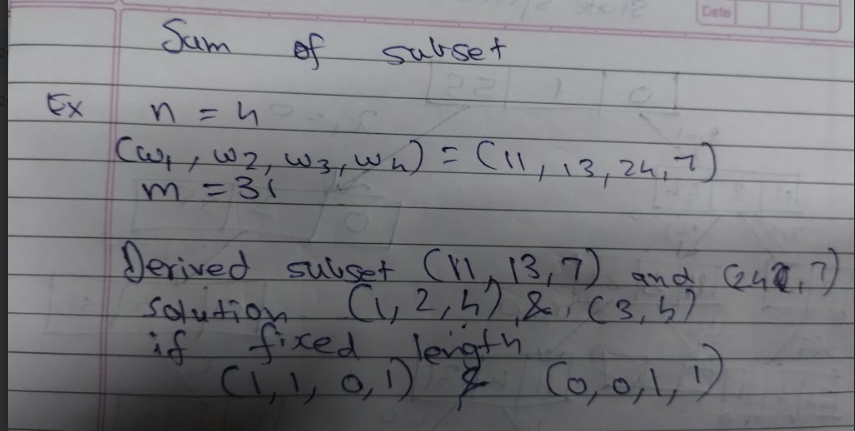
**func(0, n, v, k, temp);**

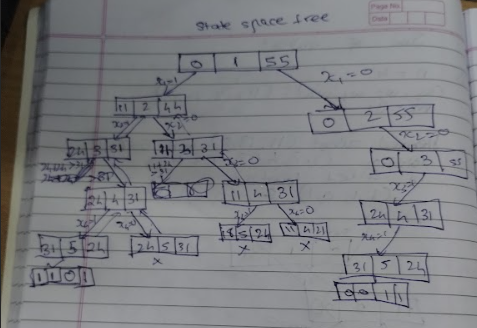
**}**

**Output:**

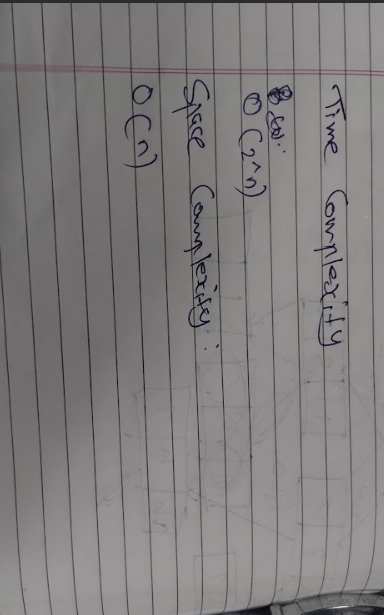


**Example sum of subset Problem along with state space tree:**

****

****

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

****

**Conclusion:**

**We have learned about and analysed the subset sum problem using backtracking.**