

|  |
| --- |
| **Title: Implementation of Backtracking Algorithm** |



**Objective:** To learn the Backtracking strategy of problem solving for SUM OF SUBSET 

**CO to be achieved:**

|  |  |
| --- | --- |
| Sr. No | Objective |
| CO 1 | Compare and demonstrate the efficiency of algorithms using asymptotic complexity notations. |
| CO 2 | Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies. |
| CO 3 | Analyze and solve problems for   different string matching algorithms. |



**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**
3. **http://www.math.utah.edu/~alfeld/queens/queens.html**
4. [**http://www-isl.ece.arizona.edu/ece175/assignments275/assignment4a/Solving%208%20queen%20problem.pdf**](http://www-isl.ece.arizona.edu/ece175/assignments275/assignment4a/Solving%208%20queen%20problem.pdf)
5. [**http://www.slideshare.net/Tech\_MX/8-queens-problem-using-back-tracking**](http://www.slideshare.net/Tech_MX/8-queens-problem-using-back-tracking)
6. [**http://www.mathcs.emory.edu/~cheung/Courses/170.2010/Syllabus/Backtracking/8queens.html**](http://www.mathcs.emory.edu/~cheung/Courses/170.2010/Syllabus/Backtracking/8queens.html)
7. [**http://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/**](http://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/)
8. **http://www.hbmeyer.de/backtrack/achtdamen/eight.htm**



**Pre Lab/ Prior Concepts:**

Data structures, Concepts of algorithm analysis



**Historical Profile:**

In computer science, the subset sum problem is an important problem in complexity theory and cryptography. The problem is this: given a set (or multiset) of integers, is there a non-empty subset whose sum is zero? For example, given the set {−7, −3, −2, 5, 8}, the answer is *yes* because the subset {−3, −2, 5} sums to zero. The problem is NP-complete, meaning roughly that while it is easy to confirm whether a proposed solution is valid, it may inherently be prohibitively difficult to determine in the first place whether any solution exists.



**New Concepts to be learned:**

Application of algorithmic design strategy to any problem, Backtracking method of problem solving Vs other methods of problem solving,SUM OF SUBSET problem and its applications.

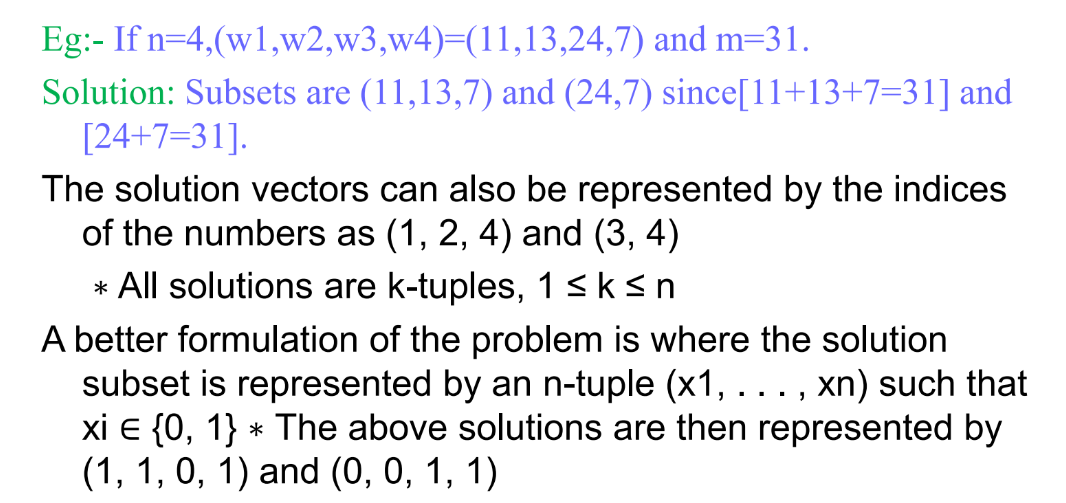


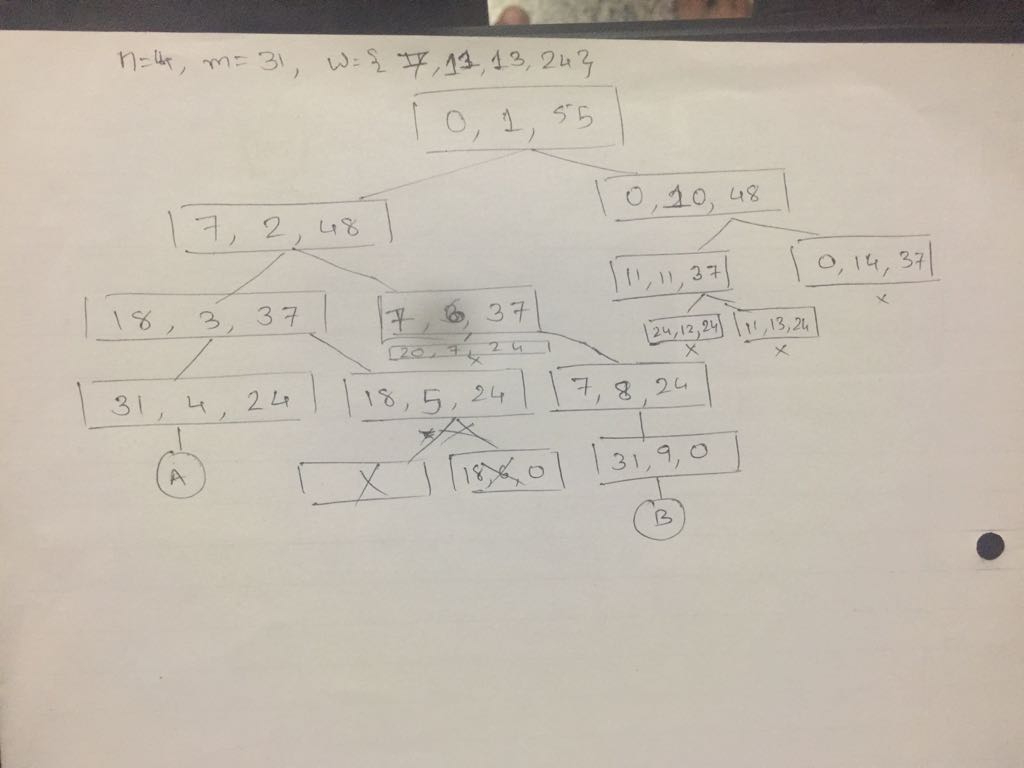
**Algorithm SUM OF SUBSET Problem:-**

The algorithm for the approximate subset sum problem is as follows:

initialize a list *S* to contain one element 0.  
 for each *i* from 1 to *N* do  
 let *T* be a list consisting of *xi* + *y*, for all *y* in *S*  
 let *U* be the union of *T* and *S*  
 sort *U*  
 make *S* empty   
 let *y* be the smallest element of *U*   
 add *y* to *S*   
 for each element *z* of *U* in increasing order do  
 //trim the list by eliminating numbers close to one another  
 //and throw out elements greater than *s*  
 if *y* + *cs*/*N* < *z* ≤ *s*, set *y* = *z* and add *z* to *S*   
 if *S* contains a number between (1 − *c*)*s* and *s*, output *yes*, otherwise *no*

**Example sum of subset Problem:**

****



**Code:**

import java.util.\*;

public class SumOfSubset

{

static int[] w;

static int[] x;

static int sum;

public static void subset(int s, int k, int r) {

int i = 0;

x[k] = 1;

if (s + w[k] == sum) {

System.out.println("\t"+s+"\t"+k+"\t"+r);

System.out.println("Ans:");

for (i = 1; i <= k; i++) {

System.out.print("\t" + x[i]);

}

System.out.println();

}

else if ((s + w[k] + w[k + 1]) <= sum)

{

System.out.println("\t" +s+"\t" +k+"\t" +r);

subset(s + w[k], k + 1, r - w[k]);

}

if ((s + r - w[k]) >= sum && (s + w[k + 1]) <= sum)

{

System.out.println("\t" +s+"\t" +k+"\t" +r);

x[k] = 0;

subset(s, k + 1, r - w[k]);

}

}

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.print("Enter the number of elements:");

int n = sc.nextInt();

w = new int[n + 1];

x = new int[n + 1];

int total = 0;

System.out.println("Enter " + n + " Elements :");

for (int i = 1; i < n + 1; i++) {

w[i] = sc.nextInt();

total += w[i];

}

Arrays.sort(w);

for (int i = 1; i <= n; i++) {

System.out.print( w[i]+"\t" );

}

System.out.println("\n"+"Enter the sum to be obtained: ");

sum = sc.nextInt();

if (total < sum) {

System.out.println("Not possible to obtain the subset");

System.exit(1);

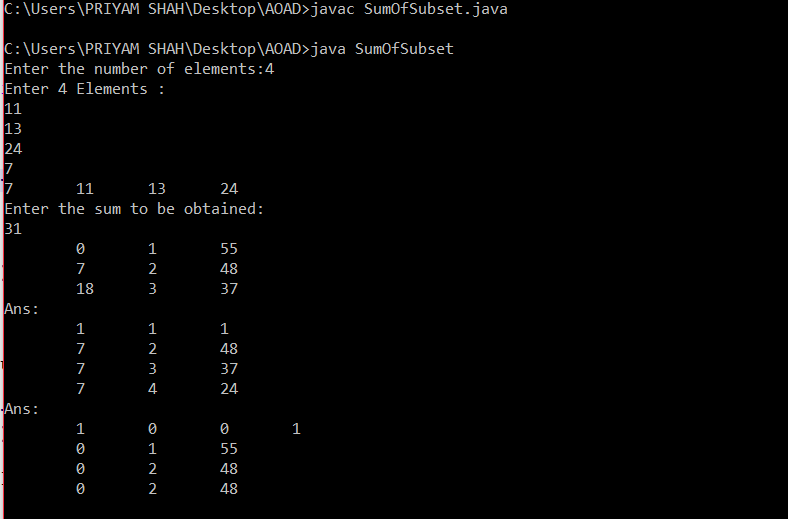
}

SumOfSubset.subset(0, 1, total);

}

}

OUTPUT:



**Time Complexity Analysis :**

The recurrence T(i,n)=T(i−1,n−1)+log2(i)

In order to check whether some subset of SS sums to nn, go over all i∈Si∈S and check whether some subset of S∖iS∖i sums to n−in−i.

The actual algorithm uses memoization and a few other optimizations. Its complexity is exponential in the size of SS (probably roughly 2|S|2|S| in the worst case).

**Conclusion:** **Thus the** **Implementation of sum of subset using Backtracking Algorithm strategy was done successfully.**