SUM OF SUBSET PROBLEM

A general paper on the sum of subset with possible solutions
Presented by Poojita Suri and Neelanja Chaturvedi
All data presented has been borrowed from various sources on the internet, and the references have been mentioned accordingly.

1.0 INTRODUCTION

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. Subset Sum is a well-known hard problem in computing that can be informally defined as given a set of positive integers and a target value, determine whether some subset has a sum equal to the target. In different versions of the problem, the input set may or may not contain duplicate values, and the problem can also be expressed as an optimization problem.

Definition: The Subset Sum problem is defined as follows: given a set of positive integers S and an integer t, determine whether there is a set S 0 such that S 0 ⊆ S and Σ(S 0) = t. This problem is particularly interesting because, depending on what parameter is used to characterize the size of the problem instance, it can be shown to have polynomial, sub-exponential, or strongly exponential worst-case time complexity. Let n represent the size of the input set S, and let m be the maximum value in the set [1].

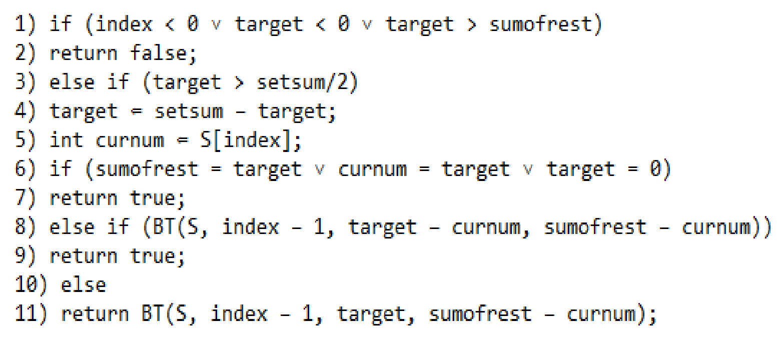
2.0 STRATEGIES FOR SOLVING SUM OF SUBSET PROBLEM

There are various strategies and algorithms that can be applied in order to solve the sum of subset problem. These strategies differ on their approach, their space-time complexity, and application.

2.1 BACKTRACKING ALGORITHM

Backtracking is a standard approach to solving Subset Sum. It has a simple recursive formulation, and with the proper bounding conditions, it is competitive with any other exact algorithm. The logic is simply to branch on the numbers in the set S. For any element y of S, if there is a subset S 0 with sum t, it either contains y or it doesn't. If S 0 contains y, we can put y in the subset we get by recursive call on S − {y} and t − y. Otherwise, we skip y and find S 0 by recursive call on S − {y} and t. The BT algorithm, a simple version of backtracking, is shown in Figure 1. The input set S is represented as an array (indexed 0 to n − 1), and an index parameter identifies which element of the array is the current branching value.[2]

A BT computation can be modeled as a binary tree where each node represents a single activation of the recursive code. Each activation processes one element of S, and it makes at most two recursive calls. So the total number of recursive calls cannot exceed the number of nodes in a full binary tree of depth n, and the worst-case time complexity is O(2n).

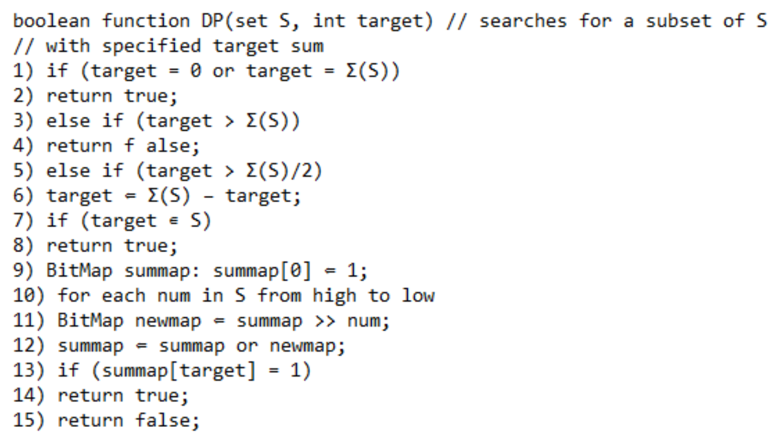


```
if (index < 0 ∨ target < 0 ∨ target > sumofrest)
return false;
else if (target > setsum/2)
target = setsum - target;
int curnum = S[index];
if (sumofrest = target ∨ curnum = target ∨ target = 0)
return true;
else if (BT(S, index - 1, target - curnum, sumofrest - curnum))
return true;
else
return BT(S, index - 1, target, sumofrest - curnum);
```

Figure 1: A backtracking algorithm (BT) for Subset Sum

2.2 DYNAMIC PROGRAMMING

In defining DP, an implementation of dynamic programming, we set up a bit array T of length t to represent the sums that can be produced by subsets of S, initially all zeros except T[0], which is set to 1 (representing the sum of the empty subset of S). When it is determined that some subset has sum r, T[r] is set to one. The sums of subsets are recorded while processing one element of S at a time. To process a number y, T is traversed from high index to low, and for each T[i] = 1, T [i + y] is set to 1. After processing all n elements of S, the array T indicates all sums of subsets of S. [2]



```
boolean function DP(set S, int target) // searches for a subset of S
// with specified target sum
if (target = 0 or target = Σ(S))
return true;
else if (target > Σ(S))
return f alse;
else if (target > Σ(S)/2)
target = Σ(S) - target;
if (target ∊ S)
return true;
BitMap summap: summap[0] = 1;
for each num in S from high to low
BitMap newmap = summap >> num;
summap = summap or newmap;
if (summap[target] = 1)
return true;
return false;
```

Figure 2: A dynamic programming (DP) algorithm for Subset Sum

2.3 GENETIC ALGORITHMS

The proposed algorithm was capable of finding an optimal solution for the subset sum problem. Also, it can be seen that the proposed genetic algorithm is problem independent and can be used to solve other combinatorial optimization problems. Let Np denote the population size and Nc denote the current number of children generated. First Nc was set to zero. Then (Np−Nc)/2 pairs of parent chromosomes were randomly selected, and the difference degree for every pair of parent chromosomes were calculated. The difference-degree (di) of ith parent pair is defined as follows: Di =Nd/ Ng Where Ng is the size of chromosome, and Nd is the number of different genes between the two parent chromosomes. [1]

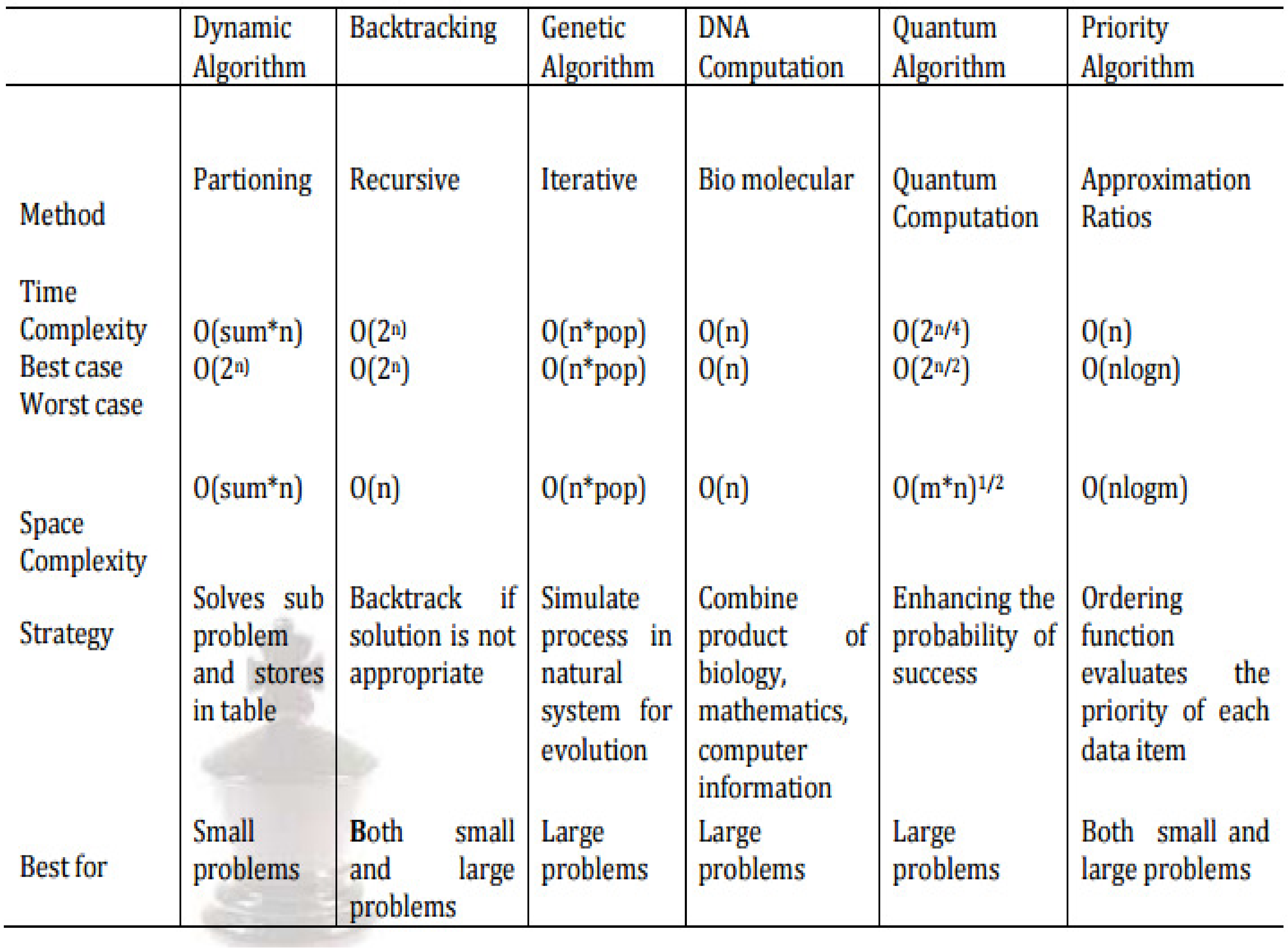
2.4 DNA ALGORITHM

A DNA procedure for solving the sum of subset problem in the Adleman Lipton model. The algorithm works in O (n) steps for the subset sum problem of an undirected graph with n vertices. The innovation of the procedure is the ingenious choice of the vertices strands' length, which can get the solution of the problem in proper length range and simultaneity simplify the complexity of the computation. However, to solve some complex problems using the new method can help us understand more about the characteristic of problem and promote the development of DNA computing research, for those DNA-based computers may be a good choice for performing massively parallel computations. Up to now, there are still many unsolved mathematical NP-complete problems because they are difficult to support basic biological experiment operations.

2.5 PRIORITY ALGORITHM

Priority algorithms for the Subset-Sum Problem with the main focus on their approximation ratios. Different classes of priority algorithms are analyzed and corresponding lower bounds are proved. Although we are unable to close the two gaps we have for fixed and adaptive revocable priority algorithms, this work gives us some basic understandings of the relative power the Priority Model, and how revocable acceptances increase the algorithm's approximation ability. The work starts with irrevocable priority algorithms, shows tight approximation ratio for this class, regardless of whether it is fixed or adaptive. In most cases, our algorithms runs time O (nlogn); the only exception is the algorithm ADAPTIVE, which is quadratic due to its checking of the terminal condition.

3.0 COMPARATIVE STUDY OF ALGORITHMS



	Dynamic Algorithm	Backtracking	Genetic Algorithm	DNA Computation	Quantum Algorithm	Priority Algorithm
Method	Partioning	Recursive	Iterative	Bio molecular	Quantum Computation	Approximation Ratios
Time Complexity Best case	O(sum*n)	$O(2^n)$	O(n*pop)	O(n)	$O(2^{n/4})$	O(n)
Worst case	$O(2^n)$	$O(2^n)$	O(n*pop)	O(n)	$O(2^{n/2})$	O(nlogn)
Space Complexity	O(sum*n)	O(n)	O(n*pop)	O(n)	$O(m*n)^{1/2}$	O(nlogm)
Strategy	Solves sub problem and stores in table	Backtrack if solution is not appropriate	Simulate process in natural system for evolution	Combine product of biology, mathematics, computer information	Enhancing the probability of success	Ordering function evaluates the priority of each data item
Best for	Small problems	Both small and large problems	Large problems	Large problems	Large problems	Both small and large problems

4.0 APPLICATIONS

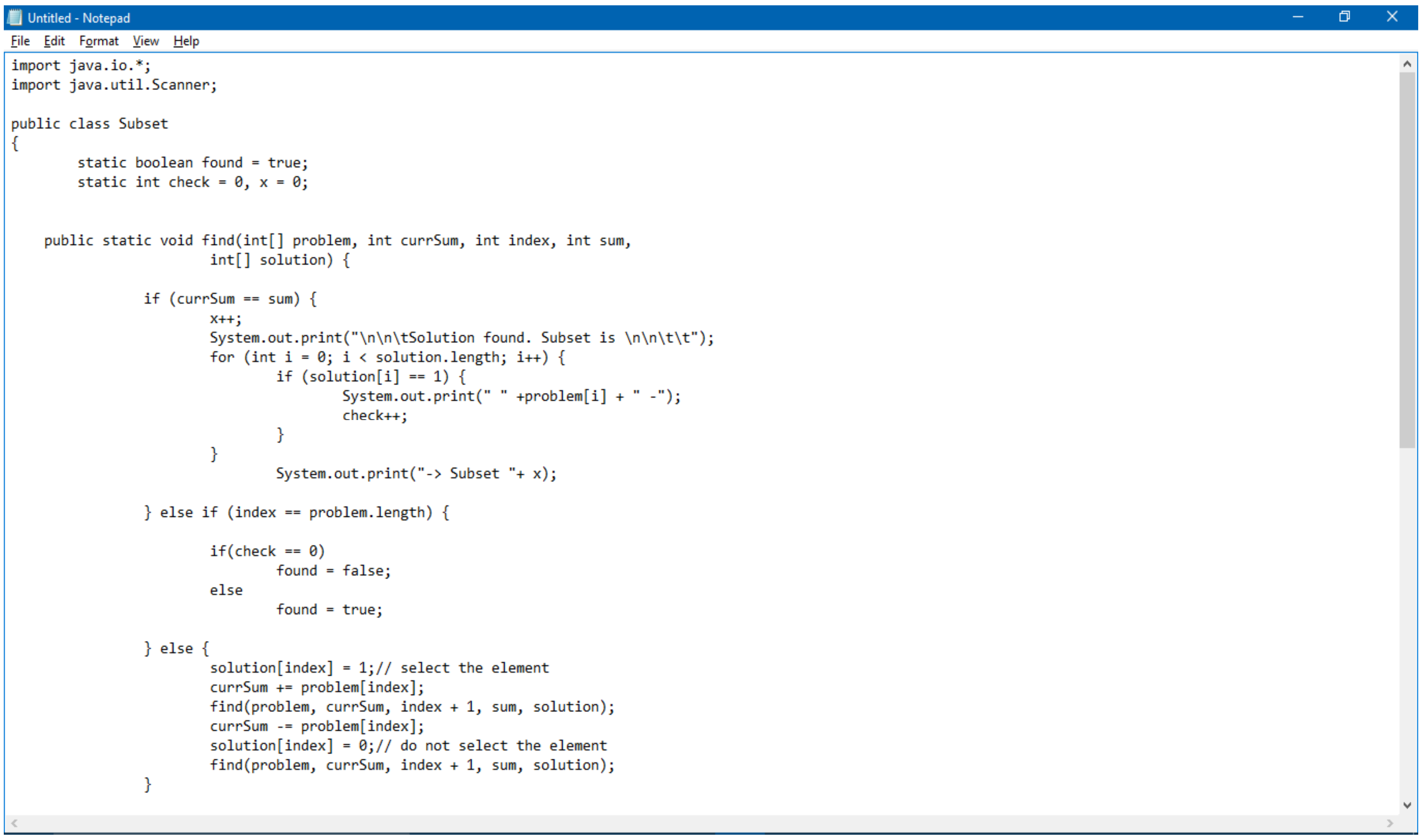
Subset-Sum problem is an important algorithm which has its applications in wide range of domains like complexity theory, cryptography and cooperative voting games. There are security problems other than public-key codes for which subset-sum problems are useful.

5.0 CONCLUSION

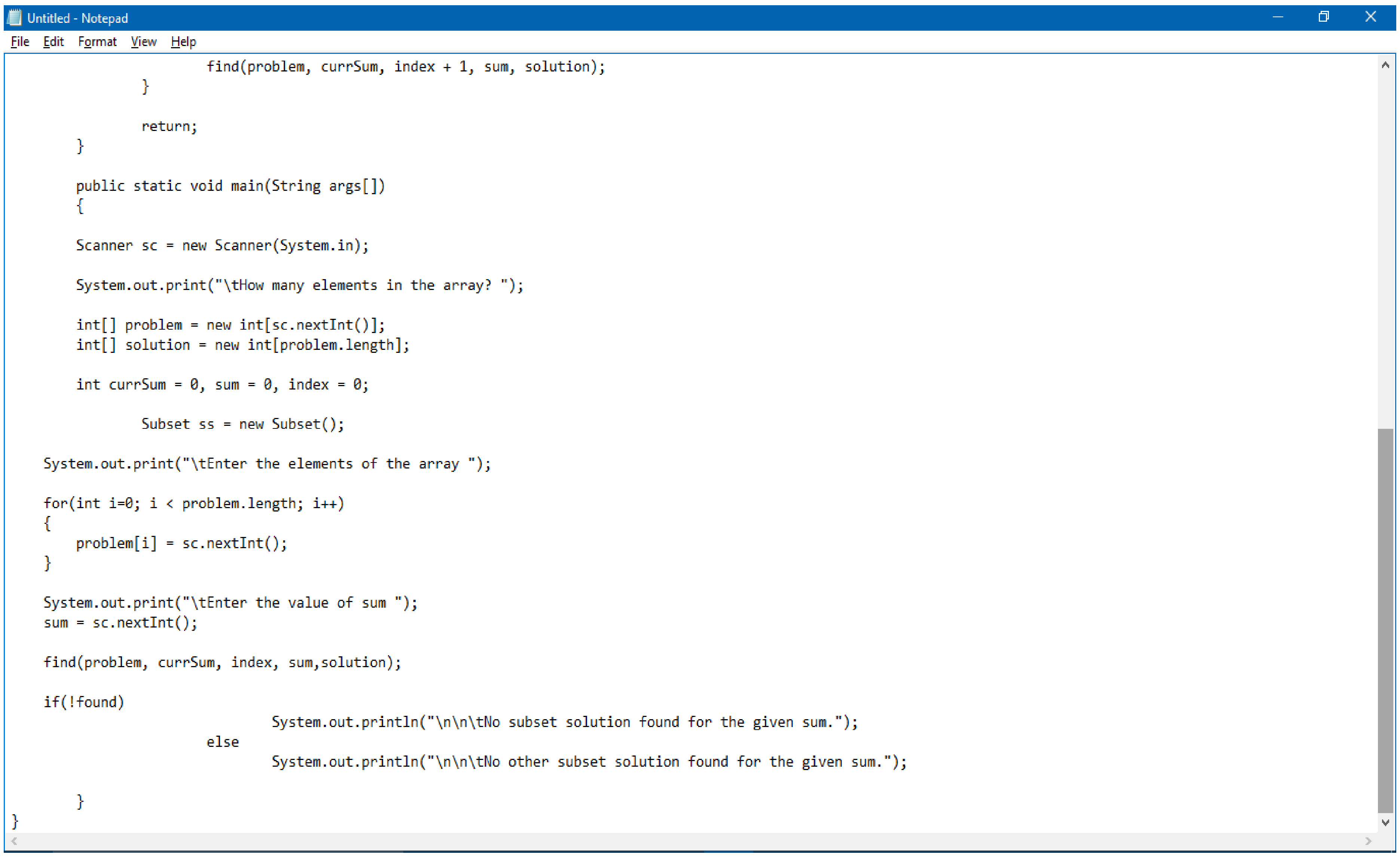
From this general research paper we can conclude that there are various techniques to solve sum of subset problem. Here we have shown the results of an empirical study that illustrates the comparative characteristics of the three algorithms mentioned above. Various experiments are designed to test the analytical complexity results, to find what density defines the critical region of the problem space, to determine what target sums are most difficult to find, and to compare. We have implemented this using backtracking and recursive function.

6.0 IMPLEMENTATION

For our purpose, we have used the backtracking approach using a recursive function to implement the sum of subset problem. We have done so on the java platform. The code for the following is as follows:

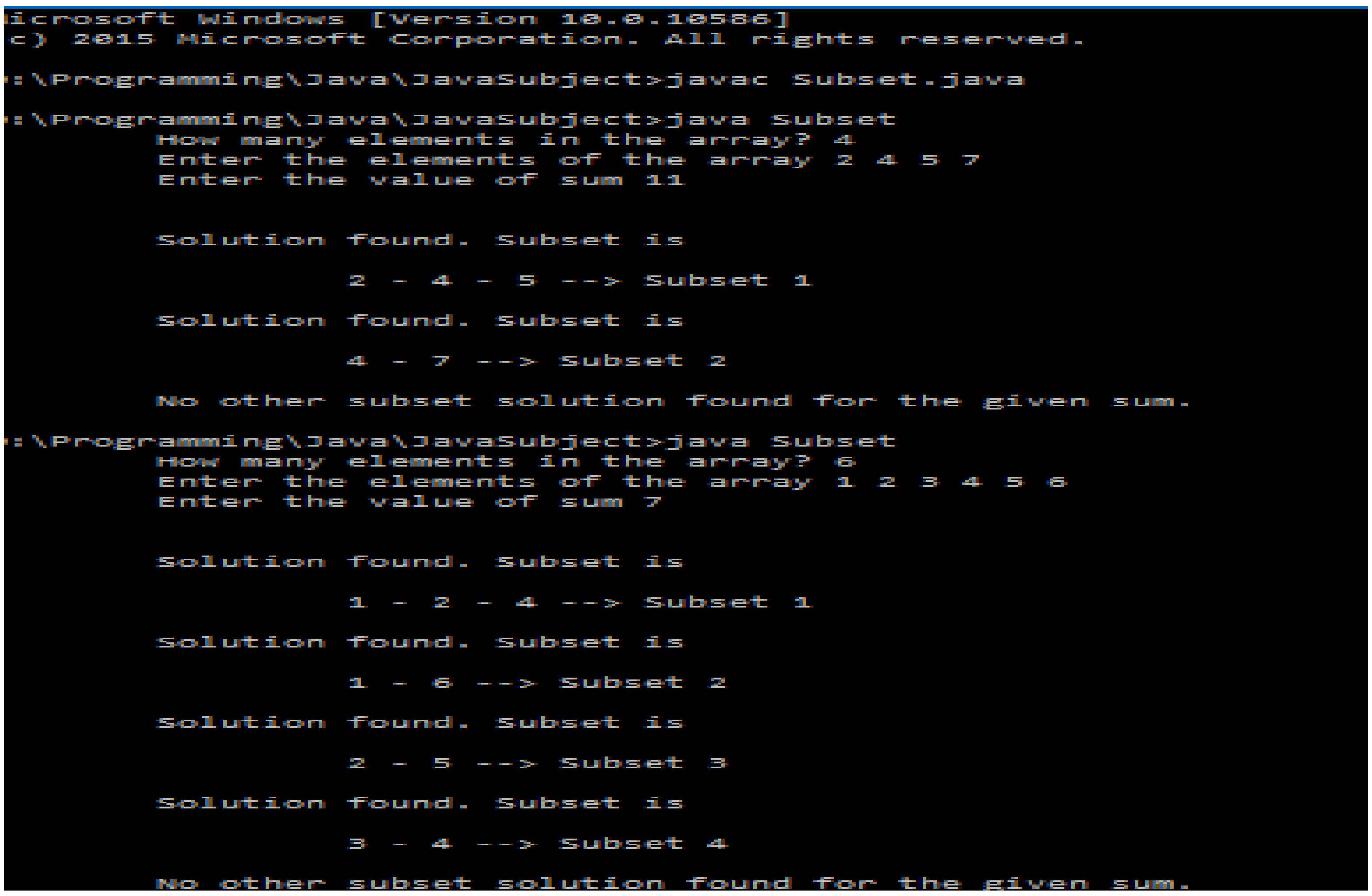


```
import java.io.*;
import java.util.Scanner;

public class Subset
{
        static boolean found = true;
        static int check = 0, x = 0;


    public static void find(int[] problem, int currSum, int index, int sum,
                        int[] solution) {

                if (currSum == sum) {
                        x++;
                        System.out.print("\n\n\tSolution found. Subset is \n\n\t\t");
                        for (int i = 0; i < solution.length; i++) {
                                if (solution[i] == 1) {
                                        System.out.print(" " +problem[i] + " -");
                                        check++;
                                }
                        }
                                System.out.print("-> Subset "+ x);

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

                        if(check == 0)
                                found = false;
                        else
                                found = true;

                } else {
                        solution[index] = 1;// select the element
                        currSum += problem[index];
                        find(problem, currSum, index + 1, sum, solution);
                        currSum -= problem[index];
                        solution[index] = 0;// do not select the element
                        find(problem, currSum, index + 1, sum, solution);
                }
```



```
                        find(problem, currSum, index + 1, sum, solution);
                }

                return;
        }

        public static void main(String args[])
        {

        Scanner sc = new Scanner(System.in);

        System.out.print("\tHow many elements in the array? ");

        int[] problem = new int[sc.nextInt()];
        int[] solution = new int[problem.length];

        int currSum = 0, sum = 0, index = 0;

                Subset ss = new Subset();

    System.out.print("\tEnter the elements of the array ");

    for(int i=0; i < problem.length; i++)
    {
        problem[i] = sc.nextInt();
    }

    System.out.print("\tEnter the value of sum ");
    sum = sc.nextInt();

    find(problem, currSum, index, sum,solution);

    if(!found)
                                System.out.println("\n\n\tNo subset solution found for the given sum.");
                        else
                                System.out.println("\n\n\tNo other subset solution found for the given sum.");

        }
}
```

OUTPUT:



```
icrosoft Windows [Version 10.0.10586]
c) 2015 Microsoft Corporation. All rights reserved.

:\Programming\Java\JavaSubject>javac Subset.java

:\Programming\Java\JavaSubject>java Subset
        How many elements in the array? 4
        Enter the elements of the array 2 4 5 7
        Enter the value of sum 11


        Solution found. Subset is

                 2 - 4 - 5 --> Subset 1

        Solution found. Subset is

                 4 - 7 --> Subset 2

        No other subset solution found for the given sum.
:\Programming\Java\JavaSubject>java Subset
        How many elements in the array? 6
        Enter the elements of the array 1 2 3 4 5 6
        Enter the value of sum 7


        Solution found. Subset is

                 1 - 2 - 4 --> Subset 1

        Solution found. Subset is

                 1 - 6 --> Subset 2

        Solution found. Subset is

                 2 - 5 --> Subset 3

        Solution found. Subset is

                 3 - 4 --> Subset 4

        No other subset solution found for the given sum.
```

REFERENCES

[1] "An empirical study on sum of subset problem" by Thomas E. O'Neil

[2] "An Analysis of Different Types of Algorithms for Sum of Subset Problem" International Journal of Advance Foundation and Research in Science & Engineering (IJAFRSE) Volume 1, Special Issue, ICCICT 2015.

[3]http://www.geeksforgeeks.org/backttracking-set-4-subset-sum/