INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact Factor: 6.078
(Volume 7, Issue 2 - V7I2-1406)
Available online at: https://www.ijariit.com

Proof of P=NP

Sanket Kulkarni
sanketskulkarni96@gmail.com
Independent Researcher

ABSTRACT

The aim of this work is to find subsets of array whose sum is K in polynomial time and hence to provide proof of P=NP

Keywords: *P vs NP, Subset Sum, Algorithm*

1. INTRODUCTION

Subset sum problem: Given an array of integers.(both positive and negative) Is there exists polynomial time algorithm to find subsets of array whose sum is k?

Consider Example: int a={3,-4, 2, 9}, k=5

Our answer should return {-4, 9} since the sum of -4 and 9 is equal to 5.

By using following algorithm we can find all subsets in polynomial time which proves P=NP.

2. JAVA PROGRAM

```
import java.util.*;

public class SubsetSum {

   public static void main(String[] args) {
      int a[] = {1,2,3,4,5,6,7,8,9,10};
      int target = 10;
      int n = a.length;
      int b[];
      HashSet<Integer> ans = new HashSet<>();
      b = a.clone();
      Arrays.sort(b);
      int min = b[0];
      int add = 0;
      if (min <= 0) {
         add = 1 - min;
         for (int i = 0; i < b.length; i++) {
            b[i] = b[i] + add;
         }
         for (int i = n - 1; i >= 0; i--) {
            for (int j = 1; j <= n; j++) {
               ans.add(b[i] - add);
               solution(b, add, j, 1, i - 1, b[i], target + (add * j), ans);
               ans.remove(b[i] - add);
            }
         }
      } else {
         for (int i = n - 1; i >= 0; i--) {
```



```
            ans.add(b[i]);
            solution(b, add, -1, 1, i - 1, b[i], target, ans);
            ans.remove(b[i]);
        }
    }
}

private static void solution(int a[], int add, int take, int taken, int ei, int mysum, int target, HashSet<Integer> ans) {
    if ((mysum == target && take == taken) || (mysum == target && take == -1)) {
        System.out.println(ans);
        return;
    }
    if (mysum > target) return;
    if (ei < 0) return;
    if (taken - take == 1) return;

    int eleindex = nearestSmaller(a, ei, target - mysum);
    for (int i = eleindex; i >= 0; i--) {
        ans.add(a[i] - add);
        solution(a, add, take, taken + 1, i - 1, mysum + a[i], target, ans);
        ans.remove(a[i] - add);
    }
}

private static int nearestSmaller(int[] arr, int ei, int target) {
    int start = 0, end = ei, index = 0;
    while (start <= end) {
        int mid = (start + end) / 2;

        if (target < arr[mid]) {
            end = mid - 1;
        } else {
            index = mid;
            start = mid + 1;
        }
    }
    return index;
}
}
```

3. EXPLANATION

Let us assume array contains only positive integers and we are not allowed to modify original array. Hence we will clone original array and sort it in increasing order. We will include each element of array in our answer to see if (target-element) or smaller number exists in array or not.

Suppose we found that number at some index 'eleindex'. Then we will loop from (0 to eleindex) and will include each element in array. We will repeat this procedure for all elements of array. We are using binary search technique to search element and HashSet to prevent duplicates. Hence time complexity will be **$O(n^3 \log n)$** for only positive integers.

If there are negative numbers in array, we will convert problem to positive integers.

To do this, we need to add some value to each element of array such that first element of array becomes 1 and all other elements will be positive.

Example: a={-4, -3, 2, 7}, target=4

In above example if we add 5 to each element, our problem will be converted to only positive integers.

We will solve this problem as mentioned above. But we need one extra for loop to check how much elements we are including when searching for target element. Hence time complexity becomes **$O(n^4 \log n)$.**

4. SUMMARY

For only positive Integers, Time Complexity: **$O(n^3 \log n)$**

For only positive Integers, Space Complexity: **$O(n)$**

For both positive and negative Integers, Time Complexity: **$O(n^4 \log n)$**

For both positive and negative Integers, Space Complexity: **$O(n)$**

