Term Project
Algorithm Design
1021. Sacrament of the Sum
Difficulty : 141

By:

5838428 Harpreet Singh Arora

5748106 Serhii Bielik

Problem Description (1)

Background

— The Brother of mine, the Head of Monastic Order wants to know tomorrow about the results long-term researches. He wants to see neither more nor less than the Summering Machine! Even moreover, he wants our Machine — only a machine — to demonstrate its comprehension of the Sacrament of the Sum as deeply as it is possible. He wants our Machine to find two numbers that give the sum equal to the Sacred Number 10 000
— Tsh-sh-sh! This is madness that borders on blasphemy! How can the Machine calculate the Sacred Number? Twenty-seven years we work on it, but we've could teach it to tell if the sum of two introduced numbers greater or lower than 10 000. Can an ordinary mortal find two numbers that there sum will be equal to 10 000?
— But we'll have to do it with the help of our Machine, even if it is not capable. Otherwise we'll have… let's say, big problems, if it is possible to call boiling oil like this. However, I have an idea. Do you remember, last week we've entered two numbers -7 and 13 into the Machine, and it answered that their sum is lower than 10 000. I don't know how to check this, but nothing's left for us than to believe to the fruit of our work. Let's enter now a greater number than -7 and start up the Machine again. We'll do like this again and again until we find a number that being added to 13 will give us 10 000. The only thing we are to do is to prepare an ascending list of numbers.
— I don't believe in this… Let's start with the sum that is obviously greater than the Sacred Number and we'll decrease one of the summand. So, we have more chances to avoid boilin… big problems.

Haven't come to an agreement, the Brothers went away to their cells. By next day everyone of them has prepared a list of numbers that, to his opinion, could save them… Can both of the lists save them together?

Problem Description (2)

Problem

- Your program should decide, if it is possible to choose from two lists of integers such two numbers that their sum would be equal to 10 000.

Input

- You are given both of these lists one by one. Format of each of these lists is as follows: in the first line of the list the quantity of numbers N_i of the i-th list is written. Further there is an i-th list of numbers each number in its line (N_i lines). The following conditions are satisfied: $1 \le N_i \le 50\,000$, each element of the lists lays in the range from -32768 to 32767. The first list is ascending and the second one is descending.

Output

- You should write "YES" to the standard output if it is possible to choose from the two lists of integers such two numbers that their sum would be equal to 10 000. Otherwise you should write "NO".

Problem Description (3)

Sample

Input	output
4 -175 19 19 10424 3 8951 -424 -788	YES

Problem Description (4)

Time limit: 1.0 second

Memory limit: 64 MB

Problem Author: Leonid Volkov & Alexander Petrov
Problem Source: Ural State University Internal Contest October'2000 Junior Session

Difficulty: 141

Solution #1

The solution is peaty straightforward.

The first obvious solution will be to go directly through arrays and compare values one by one in the nested loops.

However, the complexity for this solution will be $\mathbf{O}(n^2)$. And that is far too slow to be able to pass all test cases.

```python
n = int(input())
a = [int(input()) for x in range(n)]
m = int(input())
b = [int(input()) for x in range(m)]

for i in range(n):
    for j in range(m):
        if a[i] + b[j] == 10000:
            print("YES")
            exit(0)
print("NO")
```

Program’s Code #1

ID	Date	Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
7877382	23:33:53 9 May 2018	Serhii Bielik	1021. Sacrament of the Sum	Python 3.6	Time limit exceeded	4	1.029	1 612 KB

Solution #2

Thus, we have to find more efficient way to solve this problem. We know that input array are sorted so this is a perfect situation for applying Binary Search which has great performance **O(Log n)**.

Binary Search: Search a sorted array by repeatedly dividing the search interval in half.

Program’s Code #2

```python
n = int(input())
a = [int(input()) for x in range(n)]
m = int(input())
b = [int(input()) for x in range(m)]


def check(x):
    global n, m

    left = 0
    right = n - 1

    while left <= right:
        mid = int((left + right) / 2)
        if a[mid] + x == 10000:
            return True
        if a[mid] + x > 10000:
            right = mid - 1
        else:
            left = mid + 1

    return False


for i in range(m):
    if check(b[i]):
        print("YES")
        exit(0)
print("NO")
```

Therefore, overall we got the **O(n Log n)** complexity because for each element in array b we run search. In the function check() we are splitting second array to halves until the suitable pair number will be found or until array is ended.

This solution is fast enough to pass all test cases:

ID	Date	Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
7877191	19:44:41 9 May 2018	Harpreet Singh Arora	1021. Sacrament of the Sum	Python 3.6	Accepted		0.998	2 344 KB

Author: Serhii Bielik • Problem: Sacrament of the Sum

ID	Date	Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
7877150	18:44:35 9 May 2018	Serhii Bielik	1021. Sacrament of the Sum	Python 3.6	Accepted		0.982	2 348 KB
7877148	18:43:49 9 May 2018	Serhii Bielik	1021. Sacrament of the Sum	Python 3.6	Wrong answer	1	0.046	716 KB

Timus Online Judge Submission Result

Solution #3

Finally, we can improve the solution to make logic more clear.

This version runs slightly faster, probably because of reducing arithmetical operations.

Program’s Code #3

```python
n = int(input())
a = [int(input()) for x in range(n)]
m = int(input())
b = [int(input()) for x in range(m)]


def find(x):
    global n, m

    left = 0
    right = n - 1

    while left <= right:
        mid = int((left + right) / 2)
        if a[mid] == x:
            return True
        if a[mid] > x:
            right = mid - 1
        else:
            left = mid + 1

    return False


for i in range(m):
    if find(10000 - b[i]):
        print("YES")
        exit(0)
print("NO")
```

ID	Date	Author	Problem	Language	Judgement result	Test #	Execution time	Memory used
7877390	23:51:48 9 May 2018	Serhii Bielik	1021. Sacrament of the Sum	Python 3.6	Accepted		0.873	3 144 KB

```python
n = int(input())
a = [int(input()) for x in range(n)]
m = int(input())
b = [int(input()) for x in range(m)]


def find(a, left, right, x):
    global n, m

    while left <= right:
        mid = int((left + right) / 2)
        if a[mid] == x:
            return True
        if a[mid] > x:
            return find(a, left, mid - 1, x)
        else:
            return find(a, mid + 1, right, x)

    return False


for i in range(m):
    if find(a, 0, n-1, 10000 - b[i]):
        print("YES")
        exit(0)
print("NO")
```

Program's Code #4

The recursive implementation of Binary Search is slower than iterative version

7877574	07:57:55 10 May 2018	Serhii Bielik	1021. Sacrament of the Sum	Python 3.6	Time limit exceeded	13	1.014	2 580 KB

Thank You ☺