

**CHANDIGARH UNIVERSITY**  
**UNIVERSITY INSTITUTE OF ENGINEERING**  
**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**



<b>Submitted By: SANSKAR AGRAWAL</b> <b>Submitted To: Er. RAHUL BHANDARI</b>	
<b>Subject Name</b>	Competitive Coding
<b>Subject Code</b>	20CSP-314
<b>Branch</b>	CSE
<b>Semester</b>	5th

## LAB INDEX

**NAME:** SANSKAR AGRAWAL

UID: 20BCS5914

**SECTION:20BCS\_MM\_806\_B**

**SUBJECT NAME:** Competitive Coding Lab

**SUBJECT CODE: 20CSP-314**

[illegible]

# Experiment – 2

## **Problem Statement 2.1 Equal Stacks**

You have three stacks of cylinders where each cylinder has the same diameter, but they may vary in height. You can change the height of a stack by removing and discarding its topmost cylinder any number of times.

Find the maximum possible height of the stacks such that all of the stacks are exactly the same height. This means you must remove zero or more cylinders from the top of zero or more of the three stacks until they are all the same height, then return the height.

### **Example**

$h1 = [1, 2, 1, 1]$

$h2 = [1, 1, 2]$

$h3 = [1, 1]$

There are **4**, **3** and **2** cylinders in the three stacks, with their heights in the three arrays. Remove the top 2 cylinders from  **$h1$**  (heights = [1, 2]) and from  **$h2$**  (heights = [1, 1]) so that the three stacks all are 2 units tall. Return **2** as the answer.

**Note:** An empty stack is still a stack.

### **Function Description**

Complete the equalStacks function in the editor below.

equalStacks has the following parameters:

- `int h1[n1]`: the first array of heights
- `int h2[n2]`: the second array of heights
- `int h3[n3]`: the third array of heights

### **Returns**

- `int`: the height of the stacks when they are equalized

### **Input Format**

The first line contains three space-separated integers,  **$n1$** ,  **$n2$** , and  **$n3$** , the numbers of cylinders in stacks **1**, **2**, and **3**. The subsequent lines describe the respective heights of each cylinder in a stack from top to bottom:

- The second line contains  **$n1$**  space-separated integers, the cylinder heights in stack **1**. The first element is the top cylinder of the stack.
- The third line contains  **$n2$**  space-separated integers, the cylinder heights in stack **2**. The first element is the top cylinder of the stack.
- The fourth line contains  **$n3$**  space-separated integers, the cylinder heights in stack **3**. The first element is the top cylinder of the stack.

### **Constraints**

- $0 < n1, n2, n3 \leq 10^5$
- $0 < \text{height of any cylinder} \leq 100$

### Sample Input

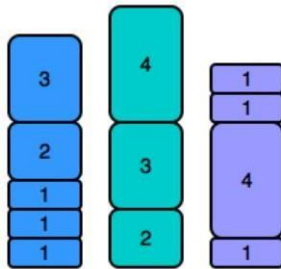
STDIN	Function
5 3 4	h1[] size n1 = 5, h2[] size n2 = 3, h3[] size n3 = 4
3 2 1 1 1	h1 = [3, 2, 1, 1, 1]
4 3 2	h2 = [4, 3, 2]
1 1 4 1	h3 = [1, 1, 4, 1]

### Sample Output

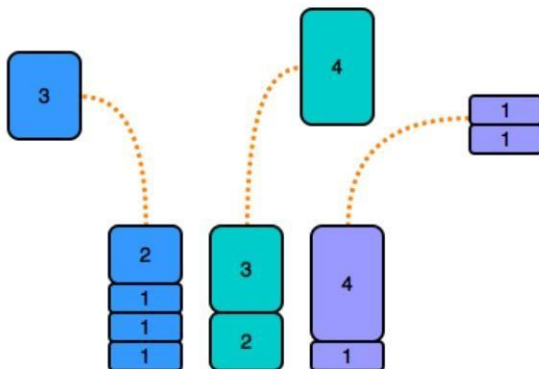
5

### Explanation

Initially, the stacks look like this:



To equalize their heights, remove the first cylinder from stacks 1 and 2, and then remove the top two cylinders from stack 3 (shown below).



The stack heights are reduced as follows:

1.  $8 - 3 = 5$
2.  $9 - 4 = 5$
3.  $7 - 1 - 1 = 5$


All three stacks now have *height* = 5, the value to return.


## Solution:


```
/*
 * Complete the 'equalStacks' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 * 1. INTEGER_ARRAY h1
 * 2. INTEGER_ARRAY h2
 * 3. INTEGER_ARRAY h3
 */
int equalStacks(vector<int> h1, vector<int> h2, vector<int> h3) {
    int n=h1.size();    int m=h2.size();    int p=h3.size();
    stack<int>a,b,c; int sum1=0,sum2=0,sum3=0; for(int i=n-1;i>=0;i--)
    ({    sum1+=h1[i];
        a.push(sum1); }
    for(int i=m-1;i>=0;i--){
        sum2+=h2[i];
        b.push(sum2); }
    for(int i=p-1;i>=0;i--){
        sum3+=h3[i];
        c.push(sum3); } while(1){
    if(a.empty()||b.empty()||c.empty()){
        return 0;
    }
    if(a.top()==b.top() && a.top()==c.top()){
        return a.top();
    }
    else if(a.top()>=b.top()&&a.top()>=c.top()){
        a.pop();
    }
    else if(b.top()>=a.top()&&b.top()>=c.top()){
        b.pop();
    }
    else {
        c.pop();
    }
}
}
```


## Output:


✓ **Test case 0**


✓ Test case 1 

✓ Test case 2 

✓ Test case 3 

✓ Test case 4 

✓ Test case 5 

✓ Test case 6 

Compiler Message

Success