Competitive Programming

Two Pointers

B.Bhuvaneswaran, AP (SG) / CSE

9791519152

bhuvaneswaran@rajalakshmi.edu.in

RAJALAKSHMI
ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

Two pointers

- Two-pointers is an extremely common technique used to solve array and string problems.
- It involves having two integer variables that both move along an iterable.
- This means we will have two integers, usually named something like i and j, or left and right which each represent an index of the array or string.

Two pointers

- Generally, the 2 Pointers approach is a good choice in those cases where:
  - The Array(s) is/are sorted
  - We are searching for a pair of numbers, or a difference etc.

Intersection of 2 Sorted Arrays

- Find the intersection of two sorted arrays or in other words, given 2 sorted arrays, find all the elements which occur in both the arrays.
- Input Format
  - The first line contains T, the number of test cases. Following T lines contain:
    - Line 1 contains N1, followed by N1 integers of the first array
    - Line 2 contains N2, followed by N2 integers of the second array
- Output Format
  - The intersection of the arrays in a single line

Sample input and output

Input:

```
1
3 10 17 57
6 2 7 10 15 57 246
```

Output:

```
10 57
```

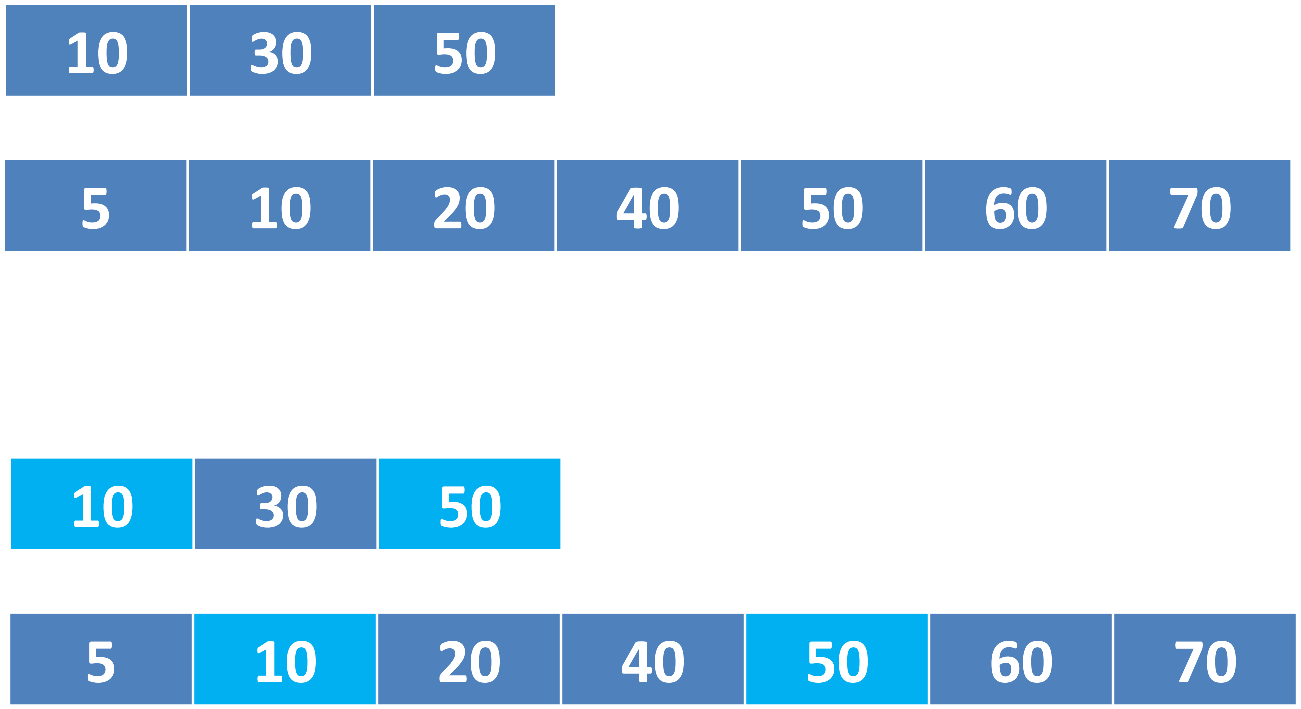
Input:

```
1
6 1 2 3 3 4 5 6
2 1 6
```

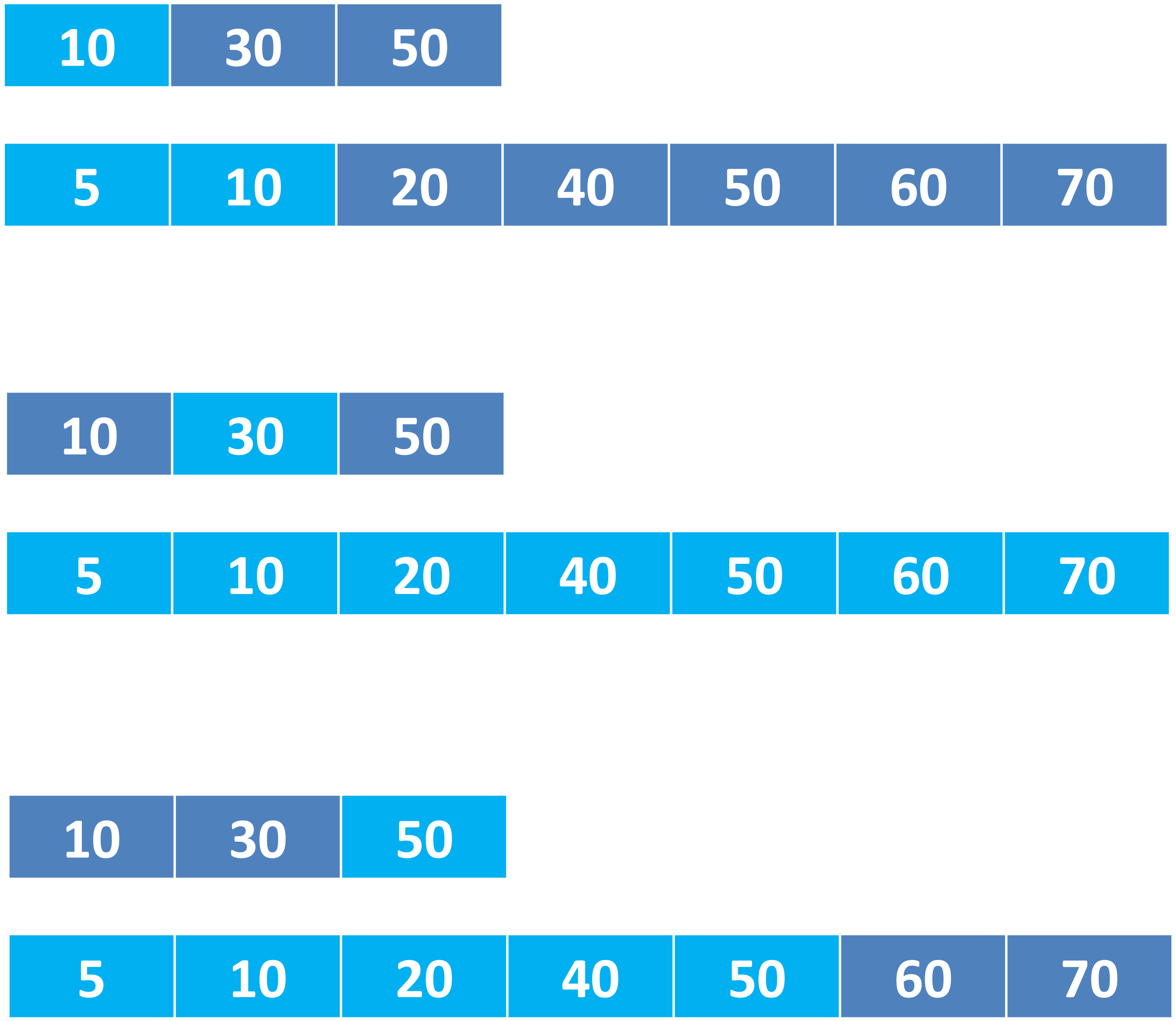
Output:

```
1 6
```

Example



Brute Force Approach



Brute Force Approach

- First Array: m
- Second Array: n
- Time Complexity: O(m * n)

Two Pointers Approach

10	30	50
↑		

5	10	20	40	50	60	70
↑						

Two Pointers Approach

10	30	50
↑		

5	10	20	40	50	60	70
	↑					

Two Pointers Approach

10	30	50
↑		

5	10	20	40	50	60	70
	↑					

Two Pointers Approach

10	30	50
	↑	

5	10	20	40	50	60	70
		↑				

Two Pointers Approach

10	30	50
	↑	

5	10	20	40	50	60	70
			↑			

Two Pointers Approach

10	30	50
		↑

5	10	20	40	50	60	70
			↑			

Two Pointers Approach

10	30	50
		↑

5	10	20	40	50	60	70
				↑		

Two Pointers Approach

10	30	50
		↑

5	10	20	40	50	60	70
				↑		

Two Pointers Approach

- First Array: m
- Second Array: n
- Time Complexity: O(m + n)

Pseudo Code

```
i = 0 // i for first array
j = 0 // j for first array
intersectionList = []
while i < A.length - 1 && j < B.length - 1:
        if A[i] < B[j]:
                i++
        else if A[i] > B[j]:
                j++
        else if A[i] == B[j]:
                intersectionList.add(A[i])
                i++
                j++
```

Note

- Arrays are sorted
- Take care of edge cases

Check pair with difference k

- Given an array A of sorted integers and another non negative integer k, find if there exists 2 indices i and j such that A[i] - A[j] = k, i != j.
- Input Format
  - First line is number of test cases T. Following T lines contain:
    - N, followed by N integers of the array
    - The non-negative integer k
- Output format
  - Print 1 if such a pair exists and 0 if it doesn’t.

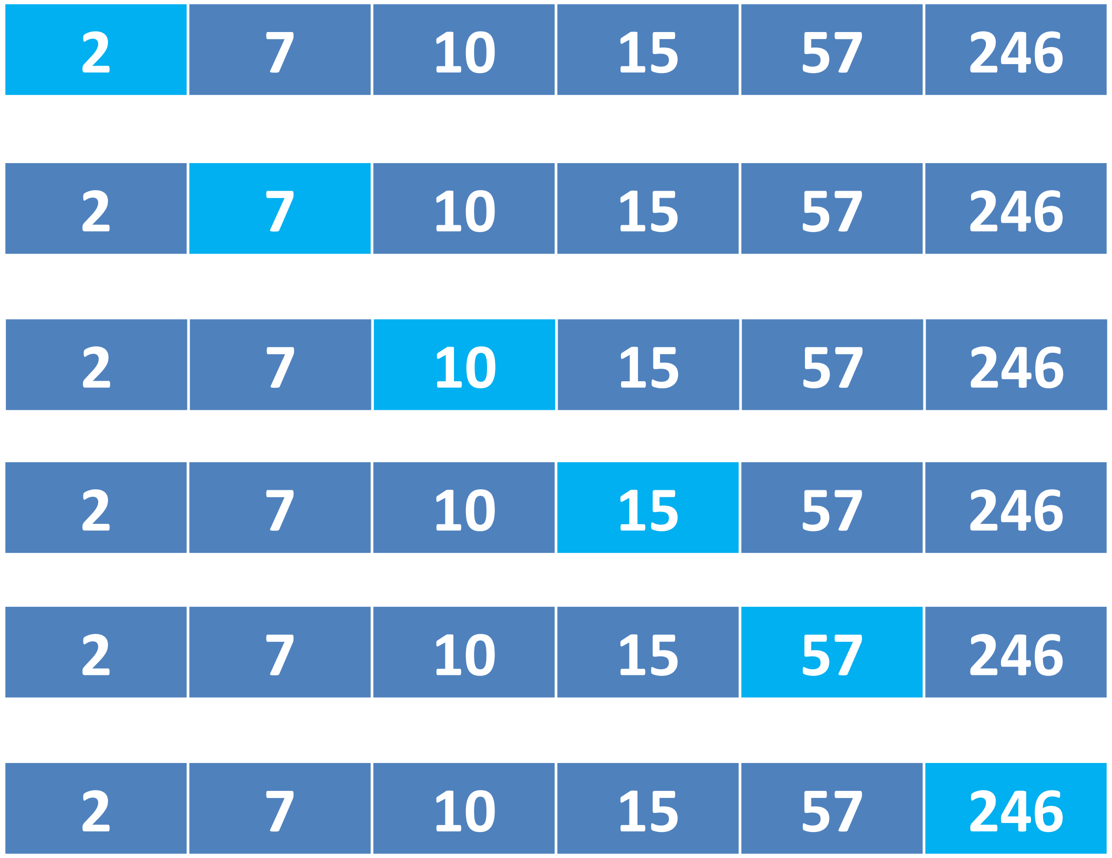
Example

- Find a pair with difference k = 47

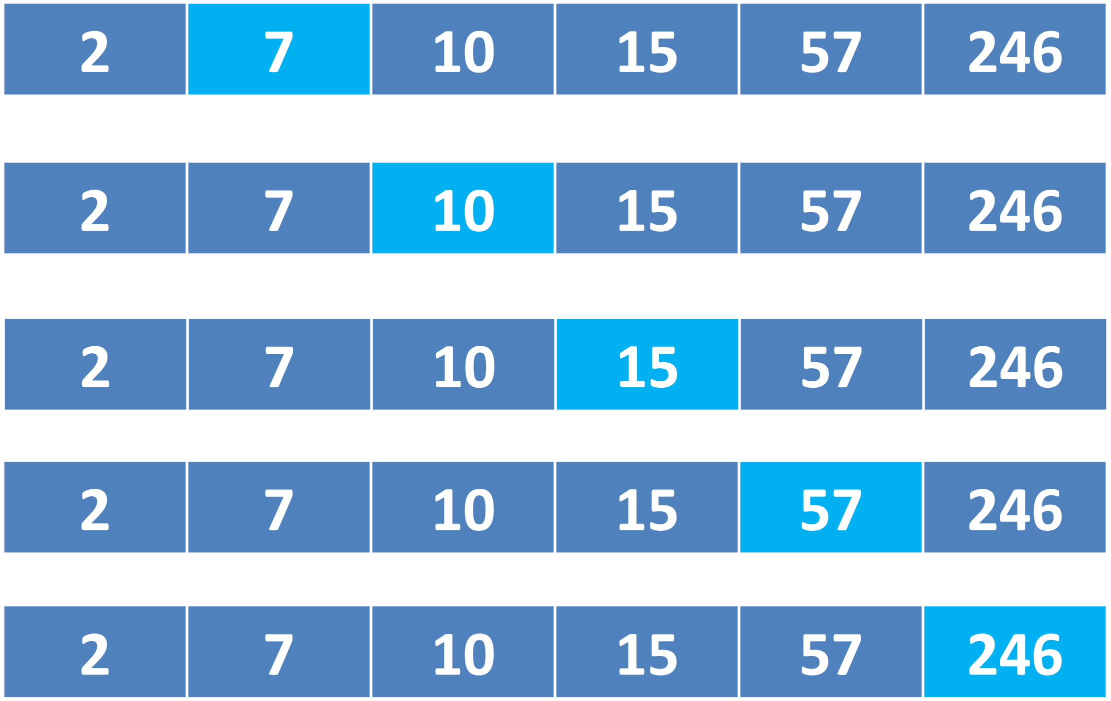
2	7	10	15	57	246

2	7	10	15	57	246

Brute Force Approach



Brute Force Approach



Brute Force Approach

2	7	**10**	15	57	246

2	7	10	**15**	57	246

2	7	10	15	**57**	246

Brute Force Approach

- Time Complexity: $O(n^2)$

Two Pointers Approach

2	7	10	15	57	246
↑ left	↑ right				

Rule

- Increment the right pointer to increase the difference.
- Increment the left pointer to decrease the difference.

Two Pointers Approach

2	7	10	15	57	246
↑ left	↑ right				

Two Pointers Approach

2	7	10	15	57	246
↑ left		↑ right			

Two Pointers Approach

2	7	10	15	57	246
↑ left			↑ right		

Two Pointers Approach

2	7	10	15	57	246
↑ left				↑ right	

Two Pointers Approach

2	7	10	15	57	246
	↑ left			↑ right	

Two Pointers Approach

2	7	10	15	57	246
		↑ left		↑ right	

Two Pointers Approach

- Time Complexity: O(n)

Queries?

Thank You…!