Pairs with given sum II

Problem Description

Given a sorted array of integers (not necessarily distinct) **A** and an integer **B**, find and return how many pair of integers (A[i], A[j]) such that i != j have sum equal to B.
Since the number of such pairs can be very large, return number of such pairs modulo (10^9 + 7).

Problem Constraints

1 <= |A| <= 100000
1 <= A[i] <= 10^9
1 <= B <= 10^9

N = 10^5 $O(N^2)$ $(10^5)^2 \rightarrow 10^{10}$ iterations

1 sec $\rightarrow$ 10^8 it, on an average

Input Format

The first argument given is the integer array A.
The second argument given is integer B.

Output Format

Return the number of pairs for which sum is equal to B modulo (10^9+7).

1+6 $\rightarrow$ 0 & 5 6+1 $\rightarrow$ 5 & 0

arr[7] = {1, 2, 2, 4, 5, 6, 7}, B = 7 (0 1 2 3 4 5 6)

1 + 6
2 + 5
2 + 5

} 3

2 elements

idea 1: 2 loops

count = 0, MOD = $10^9 + 7$

```
for ( i = 0 ; i < N ; i++ ) {
    for ( j = i+1 ; j < N ; j++ ) {
        B - arr[i]
        if (arr[j] == B - arr[i]) {
            count = (count + 1) % MOD
        }
    }
}
return count
```

TC: $O(N^2)$

SC: $O(1)$

arr[7] = {1, 2, 2, 4, 5, 6, 7} (0 1 2 3 4 5 6)

i = 0 j = 1 2 3 4 5 6 0,5 ✓

i = 1 j = 2 3 4 5 6

i = 2 j = 3 4 5 6

i = 3 j = 4 5 6

i = 4 j = 5 6

i = 5 j = 6

i = 6 j ✗

{1, 2, 2, 4, 5, 5, 7}, B = 7 (0 1 2 3 4 5 6)

frequency hashmap

	B-arr[i]		Integer ele	Integer freq
1 →	6	✗	1	1
2 →	{5 5}	2	2	2
			4	1
2 →	{5 5}	2	5	2
4 →	3	✗	7	1

```
for (i=0  i<N  i++) {
    if (map.containsKey
        (B-arr[i]) {
        count += map.get(B-arr[i])
    }
}
return count/2
```

5 → { 2 2 } 2

5 → { 2 2 } 2

2	5
2	5
2	5
2	5

5	2
5	2
5	2
5	2

7 → 0 X

B = 7

{ 1 2 5 6 }

```
for (i=0  i<N  i++) {
    if (map.containsKey
        (B-ar[i]) {
        count += map.get(B-ar[i])
    }
}
return count/2
```

TC: O(N)
SC: O(N)

1	1
2	1
5	1
6	1

c = ~~1~~ ~~2~~ ~~3~~ 4

count/2

1 6
2 5
5 2
6 1

0 0 ✓

Edge case:

0	1	2	3	4	5
2	2	2	2	2	3

4 + 3 + 2 + 1 = 10 pairs

B = 4

ar[i] + ar[j] == B

same

2	~~3~~ ~~4~~ 5
3	1

		B-ar[i]	count
i = 0	2	2	4
i = 1	2	2	4
i = 2	2	2	4
i = 3	2	2	4
i = 4	2	2	4
i = 5	3	1 X	

20

20/2 = 10

Implementing this code :-

```
for (i=0  i<N  i++) {

    if (map.containsKey
            (B-ar[i]) {
        if (ar[i] != B - ar[i]) {
            count += map.get(B-ar[i])
        } else {
            c = map.get (B - ar[i])
            map.put (B - ar[i], c - 1)
            count += map.get(B-ar[i])
            map.put (B - ar[i], c + 1)
        }
    }
}
return count/2
```

TC : O(N)

SC : O(N)

Search in Bitonic Array!

Problem Description

Given a bitonic sequence **A** of **N** distinct elements, write a program to find a given element **B** in the bitonic sequence in **O(logN)** time.

NOTE:

- A Bitonic Sequence is a sequence of numbers which is first strictly increasing then after a point strictly decreasing.

Problem Constraints

$3 <= N <= 10^5$

$1 <= A[i], B <= 10^8$

Given array always contain a bitonic point.

Array A always contain distinct elements.

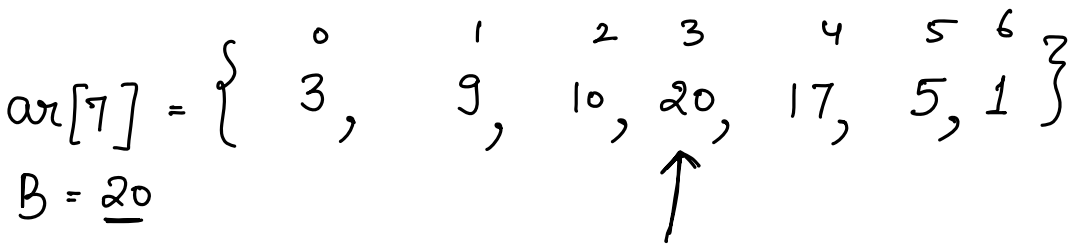
Input Format

First argument is an integer array **A** denoting the bitonic sequence.

Second argument is an integer **B**.

Output Format

Return a single integer denoting the position (0 index based) of the element **B** in the array **A** if **B** doesn't exist in **A** return **-1**.



$\Rightarrow$ 3

B = 15

$\Rightarrow$ -1

bitonic point/index

Overview :

if(B == ar[bp])

return bp

else {

0 ← bp-1

bp+1 → N-1

}

0 — bp-1 : sorted (asc)

bp bp+1 — N-1 : sorted (descending)

```java
// finding bitonic point
public int findBitonicPoint(int[] arr, int n, int l, int r) {
    int mid;
    mid = (r + l) / 2;
    if (arr[mid] > arr[mid - 1] && arr[mid] > arr[mid + 1]) {
        return mid;
    } else if (arr[mid] > arr[mid - 1] && arr[mid] < arr[mid + 1]) {
        return findBitonicPoint(arr, n, mid, r);
    } else if (arr[mid] < arr[mid - 1] && arr[mid] > arr[mid + 1]) {
        return findBitonicPoint(arr, n, l, mid);
    }
    return -1;
}
```