Kth element

BRUTE FORCE :

Use merge sort technique of merging arrays and return the kth element. Can be optimised by using counter variable and two pointer whenever counter reaches k return that element.

- Time Complexity : O(k)
- Space Complexity :O(1)

Optimal Approach :

Same as median we can partition the array to find the exact point at which the arrays can be merged.

Some edge cases to keep in mind while solving this problem:

What is the min and maximum elements that can be picked in median question :

low=0 and high =min(m,n)

But here we have kth element what if the value of k is less than min(m,n) that is considering

1 2 7 8

3 4 10 12

k=2

the maximum element we can pick from 1st array is 2 instead of m because when we take 0 elements from arr2 we are left only with arr1 and if we consider to take 4 elements it is not valid because left half should contain k elements only and m>k so high=min(k,m)

so high should be min(k,m)

2nd edge case

7 12 14 15

1 2 3 4 9 11

k=7

max elements that can be picked from arr1 is high=4 and min element low=0

but in left half we must have 7 elements and arr2 contains only 6 even if we take all elements from arr2 we still need 1 element to complete left half which changes low=1 that is min 1 element nees to be taken from arr1. So low is max(0,k-n).

```
#include <bits/stdc++.h>
int ninjaAndLadoos(vector<int> &arr1, vector<int> &arr2, int m, int n, int k) {
    // Write your code here.
    if(m > n) {
        return ninjaAndLadoos(arr2, arr1, n, m, k);
    }

    int low =max(0,k-n), high = min(k,m);

    while(low <= high) {
        int cut1 = (low + high) >> 1;
        int cut2 = k - cut1;
        int l1 = cut1 == 0 ? INT_MIN : arr1[cut1 - 1];
        int l2 = cut2 == 0 ? INT_MIN : arr2[cut2 - 1];
        int r1 = cut1 == m ? INT_MAX : arr1[cut1];
        int r2 = cut2 == n ? INT_MAX : arr2[cut2];

        if(l1 <= r2 && l2 <= r1) {
            return max(l1, l2);
        }
        else if (l1 > r2) {
            high = cut1 - 1;
        }
        else {
            low = cut1 + 1;
        }
    }
    return 1;
}
```

- Time Complexity : O(log(min(m,n)))
- Space Complexity : O(1)