Median of two sorted arrays

BRUTE FORCE :

Copy both arrays to new array and sort them. If size is odd find mid elem and return if it is even find both middle elem and divide by 2 to get median.

```
double median(vector<int>& a, vector<int>& b) {
  // Write your code here.
  vector<int> c;
  for(int i=0;i<a.size();i++)
  {
    c.push_back(a[i]);
  }
  for(int j=0;j<b.size();j++)
  {
    c.push_back(b[j]);
  }
  sort(c.begin(),c.end());
  double median;
  int n=c.size();
  if(n%2==0)
  {
    median= (c[n/2-1]+c[n/2])/2.0;
  }
  else{
    median=c[n/2];
  }
  return median;
}
```

- Time Complexity: O(N+M log(N+M))
- Space Complexity : O(N+M)

Optimal Approach :

We use binary search to determine what is the right amount of elements that must be taken from each array in left half and right half. We use two variables to determine where we need to partition the elements of the array (cut1 and cut 2). We maintain 4 variables namely l1,l2,r1,r2 where l1 is the end of left partition of array1 and l2 is the end of partition of array2 just before cut1 and cut2.

r1 is the starting of array1 partition(cut1 index) and r1 is the starting of array2 partition (cut2index).

Now we know that both arrays are sorted but when we partition the array we need to make sure that the last element of array1 is less than the starting element of array2 because all the elements on the left must be less than the elements on the right . Already left of arr1 is less than right of arr1 but comparison is done only for arr1 and arr2 for both l1≤r2 and l2≤r2 when this condition is met then we check whether length is even or odd. For even length we need 2 elements which needs to be max element of left half and min element of right half. add both and divide by 2.

for odd length just find max of left half.

Binary Search is performed only on the array which is of smaller length.

Code :

```
double median(vector<int>& a, vector<int>& b) {
  // Write your code here.

  if(a.size()>b.size())
    median(b,a);
  int n1=a.size();
  int n2=b.size();
  int low=0,high=n1-1;
  while(low<=high)
  {
    int cut1=low+(high-low)/2;
    int cut2=((n1+n2+1)/2)-cut1;
    int l1=cut1==0?INT_MIN:a[cut1-1];
    int l2=cut2==0?INT_MIN:b[cut2-1];
    int r1=cut1==n1?INT_MAX:a[cut1];
    int r2=cut2==n2?INT_MAX:b[cut2];
    if(l1<=r2 && l2<=r1)
    {
      if((n1+n2)%2==0)
      {
        return (max(l1,l2)+min(r1,r2))/2.0;
      }
      else
      {
        return max(l1,l2);
      }
    }
    else if(l1>r2)
    {
      high=cut1-1;
    }
```

```
    else{
      low=cut1+1;
    }
  }
  return 0.0;
}
```

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