Problems in Binary Search.

1) Index of first & Index of last Occurence.

I/P: arr[] = {15, 15, 20}
x = 15

O/P: 0

Naive → linear search direct.
worst → when ele not there → O(n) time.
& O(1) space

Eff → O(logn) time
- If ele is same, ele mid might not be first ele.

{5, 10, 10, 15, 20, 20, 20} → {20, 20, 20}

So we compare the mid with mid - 1.

- if (arr[mid] == x) { return mid; }

↳ changed to put it last in else.{
if (mid == 0 || arr[mid-1] != arr[mid])
{return mid;}
else { return firstOcc/BSearch(arr, low, mid-1, x);}

}

Same for iterative search. → high = mid - 1

② Count Occurrences in a Sorted Array.

I/p arr[] = {10, 20, 20, 20, 30, 30}

x = 20

O/p = 3.

→ Naive → Linear Search → O(n)

Eff → Use 2 Binary Searches.

first Occ

& Last Occ. (In last pg)

↳ Not done, very similar to first Occ

```
int countOcc (int arr[], int n, int x)
{
    int first = firstOcc(arr, n, x);
    if(first == -1)
    { return 0; }
    else { return (lastOcc(arr, n, x) - first + 1); }
}
```

---

③ Square Root:

I/p: x = 4 I/p: x = 14

O/p = 2 O/p: 3.

In → sqrt()
Java is there in Math class

```
int sqRootFloor (int x)
{ int i = 1;
  while (i*i <= x)
    { i++; }
  return (i-1);
}
```

O(√x)

O(log x)

Eff sol.

```
int SqRootFloor(int x)
{
    int low = 1, high = x, ans = -1;
    while(low <= high)
    {
        int mid = (low+high)/2
        int mSq = mid * mid;
        if(mSq == x)
            { return mid; }
        else if(mSq > x)
            { high = mid-1; }
        else {
            low = mid+1;
            ans = mid;
        }
    }
    return ans;
}
```

Search in an infinite ~~sorted~~ Array.

① Linear Search ⟶ `if(arr[i] == x) return i;`
`if(arr[i] > x) return -1;`

② Reverse Binary Search

```
int Search(int arr[], int x)
{
    if(arr[0] == x) return 0;
    int i = 1;
    while(arr[i] < x)
        { i = i*2; }
    if(arr[i] == x) return i; }
    return BinarySearch(arr, x, i/2+1, i-1);
}
```

Search in a Sorted Roteted Array.

⇒ I/p: arr[] = {10, 20, 30, 40, 50, 8, 9} (x = 30.)

O/p: 2.

I/p: {100, 200, 300, 10, 20} (x = 40.)

O/p: -1.

* Naive → Linear Search and return. O(n)

Naive (2) → Rotate the array back and apply binary Search.
O(k * logn).

→ Sort and Binary Search
O(nlogn)

Split array from where rotation started and apply 2 binary Searches (O(k) + O(logn)).

Eff.

⇒ If array is sorted and rotated :-

* At least half of array must be sorted.
* which side is sorted can be found out by comparing mid with corner elem.

Code:-

```
int Search (int arr[], int n, int x)
{ int low=0  high = n-1;
while (low<=high)
{ mid=(low+high)/2;
  if (arr[mid]==x) return mid;

if(arr[low] < arr[mid])
  { if(x>= arr[low] && arr[mid] >x)
    { high = mid-1; }
    else{ low = mid+1; }
  }

else
  { if(x> arr[mid] && x<=arr[high])
    { low = mid+1; }
    else{ high = mid-1; }
  }
} return -1;

}
```