**Binary Search:**

* Optimised way to search
* Used For sorted arrays(Ascending or descending)

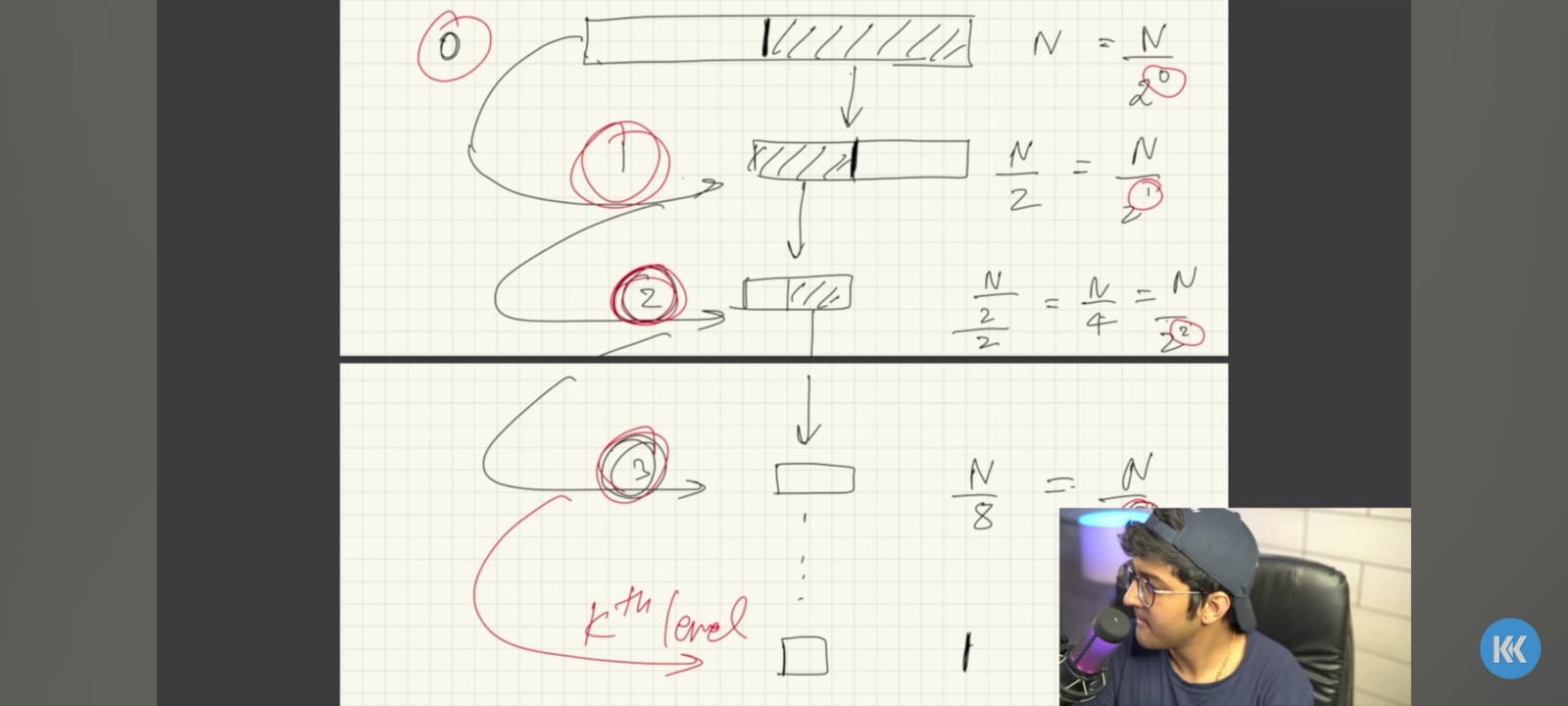
**Steps in Binary Search:**

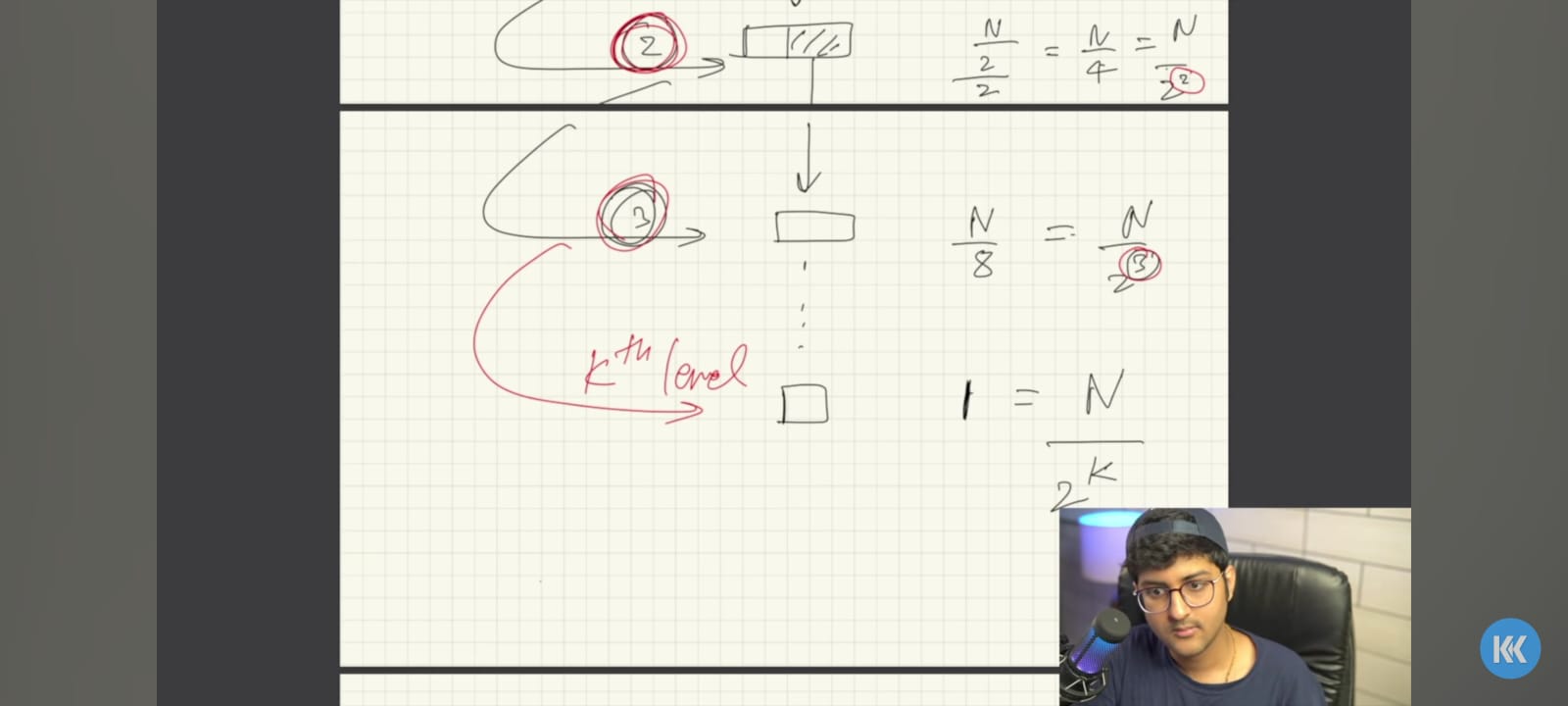
1. Sort the array(asc)
2. Start index=0 and end index=array.length-1
3. Find the middle element.
4. If middle element==target, ans found.
5. If target>middle element🡪search in the right hand side of the middle element by making start=middle +1 other wise check in the left hand side by making end=middle-1;
6. Repeat the process until start index >end index
7. If start>end element not found.

**Time Complexity:**

**Best Case:** When the element is at middle index. O(1).

**Worst Case:**





N/2^K=1🡺N=2^K

Log(N)=log(2^K)🡺log(N)=Klog(2)🡺k=log(N)/log(2)

K=log2(N)

Total Comparisons in worst case=logN

If we comapare with linear search, consider an array of size one million. In linear search the worst case comparisons would be one million but in binary search the worst case comparisons are only 19.

**How to know whether the array is sorted in ascending or descending..?**

If(arr[0]>arr[arr.length-1])🡺descending

If(arr[0]<arr[arr.length-1])🡺ascending

**Binary Search:**

**package** BinarySearch;

**public** **class** main {

**public** **static** **void** main(String[] args) {

**int**[] arr= {-34,-23,-16,-7,7};

**int** target=-34;

**int** ans=*binarysearch*(arr,target);

System.***out***.println(ans);

}

//return the index

//return -1 if does not exist

**static** **int** binarysearch(**int**[] arr, **int** target) {

**int** start=0;

**int** end=arr.length-1;

**while**(start<=end) {

//find the middle element

//int mid=(start+end)/2;//might be possible that (start+end)may exceeds the range of integer in java;

**int** mid=start+(end-start)/2;//s+(e-s)/2==>(2s+e-s)/2==>(s+e)/2;

// System.out.println(mid);

**if**(target<arr[mid]) {

end=mid-1;

}

**else** **if**(target>arr[mid]) {

start=mid+1;

}

**else** {

**return** mid;

}

}

**return** -1;

}

**static** **int** binarysearchDesc(**int**[] arr, **int** target) {

**int** start=0;

**int** end=arr.length-1;

**while**(start<=end) {

//find the middle element

//int mid=(start+end)/2;//might be possible that (start+end)may exceeds the range of integer in java;

**int** mid=start+(end-start)/2;//s+(e-s)/2==>(2s+e-s)/2==>(s+e)/2;

// System.out.println(mid);

**if**(target<arr[mid]) {

start=mid+1;

}

**else** **if**(target>arr[mid]) {

end=mid-1;

}

**else** {

**return** mid;

}

}

**return** -1;

}

}

**Order agnostic binary search:**

**package** BinarySearch;

**public** **class** OrderAgnosticBinarySearch {

**public** **static** **void** main(String[] args) {

// int[] arr= {-34,-23,-16,-7,7,8,67,87,99};

**int**[] arr= {99,80,75,22,11,10,5,2,-3};

**int** target=22;

**int** ans=*orderagnosticBs*(arr,target);

System.***out***.println(ans);

}

**static** **int** orderagnosticBs(**int**[] arr, **int** target) {

**int** start=0;

**int** end=arr.length-1;

//find whether the array is sorted in asc or desc

**boolean** isAsc=arr[start]<arr[end];

**while**(start<=end) {

//find the middle element

//int mid=(start+end)/2;//might be possible that (start+end)may exceeds the range of integer in java;

**int** mid=start+(end-start)/2;//s+(e-s)/2==>(2s+e-s)/2==>(s+e)/2;

**if**(arr[mid]==target) {

**return** mid;

}

**if**(isAsc) {

**if**(target>arr[mid]) {

start=mid+1;

}

**else** {

end=mid-1;

}

}

**else** {

**if**(target>arr[mid]) {

end=mid-1;

}

**else** {

start=mid+1;

}

}

}

**return** -1;

}

}