August 3, 2025

BINARY SEARCH FOUNDATIONS

* Syllabus completed so far :
  - → Maths (Combinatorics | Number Theory)
  - → STL Basics
  - → Contribution Technique
  - → Greedy (intro)
  - → Prefix / Partial sums
  - → STL Applications

FOUNDATION

* Topics to be covereds in next 8 weeks :
  - → Binary Search (This week)
  - → Two pointers — 1 week
  - → Recursion — 1 "
  - → Backtracking — 1 "
  - → Graph — 3 "
  - → Trees — 1

* Future Topics :
  - → Bit Manipulation — 1 week
  - → Dynamic Programming — 3 weeks
  - → Seg — 1 week
  - → String & Trie — 1 week
  - → Linked List | Binary Tree — 2 weeks

* Basics of Binary Search

Given a sorted array, find the position of element x.

Sorted array :

0	1	2	3	4	5	6	7	8
1	3	5	9	11	13	14	19	20

X = 14

Approach : 1. We calculate mid to create a partition.
[lo = 0, hi = 8] $mid = \frac{lo + hi}{2}$.

2. The value at mid can tell us whether x will be to the left of mid or right. So, if our $x = 14$ in above example. Here, the mid will come as $mid = \frac{0+8}{2} = 4$

$a[4] = 11$.. Since we have a sorted array, we know that 14 will never lie before 11 and hence we should look towards the right of 11.

→ Shift lo/hi accordingly

3. Reiterate over new range until x is found.

Time Complexity = $O(\log N)$

* Fundamental problem of Binary Search

Note: Every Binary Search Problem can be reduced to this.

Given a sorted array of 0's and 1's where 0's is followed by 1's. Find the First 1. (index of first 1)

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	1	1	1

Ans = 6

Approach:

1. Search Space: Feasible range of my answer.

ANS → index → [0 _ _ _ _ _ _ N-1] range

2. Use of Binary Search:

Variables that we will use -

a. lo - hi

It tells me what range to search in. Initial values here will be lo = 0, hi = N-1

b. Ans

It stores best answer (index) that we found till now or Default Answer

Ans = N (n here indicates the end of search space, meaning the index of the value doesn't exist)

default answer

c. $Mid = lo + \frac{(hi - lo)}{2} \approx \frac{lo + hi}{2}$

Better to write this way in interviews because of integer overflow.

Suppose "INT → [0 - - - - 10]

lo = 8, hi = 10

$Mid = \frac{8 + 10}{2} = \frac{18}{2}$ → Overflow (int range is from 0 to 10)

$Mid = 8 + \frac{(10-8)}{2} = 8 + \frac{2}{2} = 8 + 1 = 9$ ✓

3. Runthrough with an example:

Idx :	0	1	2	3	4	5	6	7	8	9
a :	0	0	0	0	0	0	1	1	1	1

Ans	lo	hi	mid
N	0	9	4
N	5	9	7
7	5	6	5
7	6	6	6
6	6	5	Terminate

Iteration 1: a[4] = 0, Shift lo towards right

Iteration 2: a[7] = 1, possible ans. Shift hi for better ans.

Iteration 3: a[5] = 0, shift lo.

Iteration 4: a[6] = 1. Possible ans. Shift hi

Here lo > hi, which means there's no better range to search. Ans = 6.

4. Pseudocode

```
lo = 0 ;
hi = N-1 ;
ans = N ;

while( lo <= hi )
{
    mid = lo + (hi-lo)/2 ;
    if ( a[mid] == 1 )      // possible answer
    {
        ans = mid ;         // update ans
        hi = mid-1 ;        // search for better answer
    } else lo = mid+1 ;     // shift to Right if there is 0.
}
```

5. What if we have to find last 0?

From the above logic, we can easily find the index of first 1. So, idx-1 will be the index of last 0 if idx is index of first 1.

Ques 1. Implement lowerbound & Upperbound from scratch in O(log N).

	0	1	2	3	4	5	6	7
a :	1	2	4	8	8	8	20	25

For x = 9, lowerbound(x) = 6 [index of element which is greater than equal to x] in this case i.e. 9

a[6] = 20 >= 9

→ Given the value of x, we will now create a new array consisting of only 1's and 0's using a conditional check.

	0	1	2	3	4	5	6	7
a:	1	2	4	8	8	8	20	25

if (a[i] ≥ x) then m[i] = 1 else m[i] = 0

	0	1	2	3	4	5	6	7
m:	0	0	0	0	0	0	1	1

Let's see how we converted a → m.

For i=0, a[i] = 1.

if (1 ≥ 9) No, m[0] = 0

For i=6, a[i] = 20

is (20 ≥ 9), Yes then m[6] = 1.

Now, it shall be clear that if we just convert a → m, then our problem reduces itself to finding the first 1 in m.

→ But, above we are creating a O(N) array (M) which isn't optimised.

- We will not create check(x) unless needed. We will only retrieve the value when we are at the idx.

Runthrough of our above approach:

[1, 2, 4, 8, 8, 8, 20, 25]

↑ 0 ↑ 3 ↑ 7

- mid = (0+7)/2 = 3

check(int x) { if (a[i] ≥ x) return 1 else return 0 }

↳ check(mid) = check(3) = 0 (a[3] ≥ 9, No) (shift lo = mid+1)

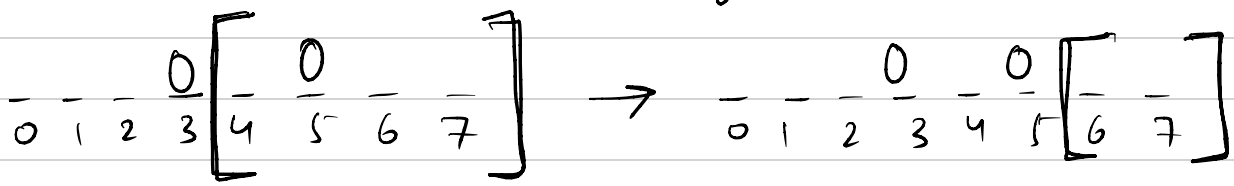
[0 1 2 3 | 4 5 6 7] ← New search space

As you can see, instead of creating the whole m[], we are getting values at particular index.

- lo = 3
  hi = 7
  mid = $\frac{3+7}{2}$ = 5

8 ↑ (a[5] ≥ 9) ↓ No

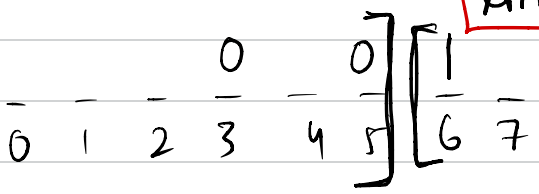
check(5) = 0 → Shift lo = mid + 1



- lo = 6
  hi = 7
  mid = $\frac{6+7}{2} = \frac{13}{2} = 6$, a[6] ≥ 9, Yes. So

check(6) = 1.

Ans = 1 → shift hi = mid − 1



Terminate

```
int n;
int arr[100100];
int x;

int check(int mid){ // only checks at mid instead of creating the whole 0,1 array
    if(arr[mid]>=x) return 1;
    else return 0;
}

void solve(){
    cin>>n;
    for(int i=0;i<n;i++)cin>>arr[i];
    cin>>x;

    // lower_bound
    int lo = 0;
    int hi = n-1;
    int ans = n; // default answer

    while(lo<=hi){
        int mid = (lo+hi)/2; // lo + (hi-lo)/2;
        if(check(mid)==1){// check mid = 1;
            ans = mid;
            hi = mid-1;
        }else{
            lo = mid+1;
        }
    }
    cout<<ans<<endl;
```

Time complexity :

Generally,

$O(\log(lo - hi + 1)) + O(check(x))$

$\Rightarrow O(\log(lo - hi + 1)) + O(1)$

$\Rightarrow O(\log N)$

Ques 2. Rotated Array.

Given all distinct elements, rotations of an array is defined as:

$[1,3,5,8,12] \xrightarrow{1} [12,1,3,5,8] \xrightarrow{2} [8,12,1,3,5] \xrightarrow{3} [5,8,12,1,3]$

Find out how many times the original sorted array was rotated to get the given array. Find in $O(\log N)$

Example:

```
  0  1  2  3  4
[ 5, 8, 12, 1, 3 ]      Ans = 3  (index of the smallest element)

  0  1  2  3  4
[ 1, 2, 3, 4, 5 ]       Ans = 0

  0  1  2  3  4
[ 2, 3, 4, 5, 1 ]       Ans = 4
```

From above examples, we can gather that the index of the smallest element returns the no. of times that array has been rotated. We can use this to design our CHECK fxn and reduce this problem to finding the first 1.

⇒ check : $a[mid] \leq a[N-1]$

```
int check(int mid){
    if(arr[mid]<=arr[n-1]) return 1;
    else return 0;
}
```

Simply use our previous written BinSearch logic to find the first 1.

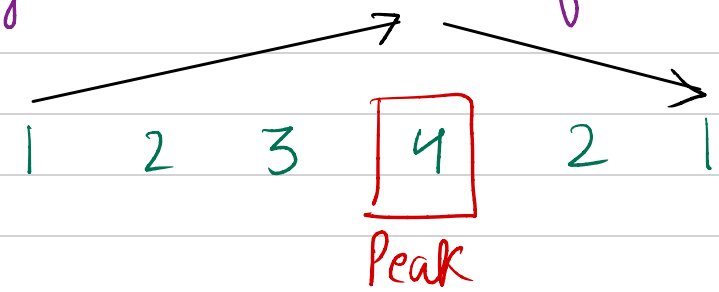
```
[ 5, 8, 12, 1, 3 ]
After check : [ 0, 0, 0, (1), 1 ]

[ 1, 2, 3, 4, 5 ]
After check : [ (1), 1, 1, 1, 1 ]

[ 2, 3, 4, 5, 1 ]
After check : [ 0, 0, 0, 0, (1) ]
```

Ques 3. Peak finding in a Bitonic Array

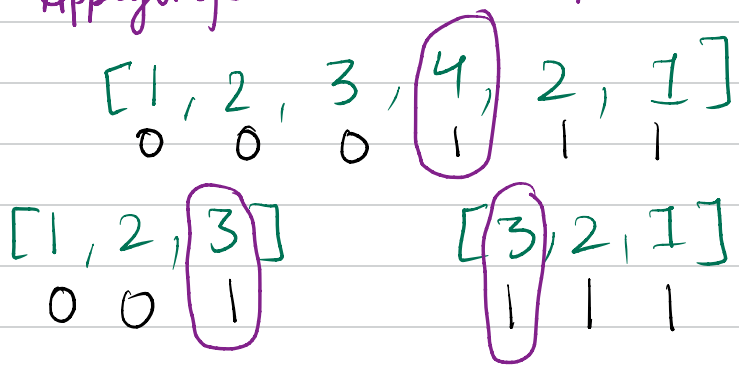
Bitonic arrays : where the value first increases & then decreases.



Similarly like previous solutions, here we will also aim to convert the above array into the problem of finding the first 1 using the check function. The rest of the logic will remain same.

```
int check(int mid){
    if(mid==n-1)return 1;
    if(arr[mid]>arr[mid+1]) return 1;
    else return 0;
}
```

Applying check on examples:



* Types of Binary Search

- Find ans in log N (Simply design check)
- Binary Search on Answer
  1. Atomic Item contribution
  2. Sweep Idea
  3. K-th element
- Binary Search on every start (Better way to solve using Two Pointers)
  - → Find # of subarrays
  - → Find shortest/longest subarrays