"रास्ते खुद बन जाते हैं,
हौसलों के कदम जब उठते हैं।"

Binary Search

What is Binary Search

- Works on sorted array
- ✓ DnC
- Search Algo.

Searching algo. which search the data in

Sorted Search Space

$mid = \frac{(s+e)}{2}$

N = 5

x = 12

s

e

s

s = 0

e = 4

m = 2

s = 3
e = 4
m = 3

s = 4
e = 4
m = 4

s = 5
e = 4

a(m)	x
a = b	→ return mid
a < b	→ Right
a > b	→ left

- define search space
- find mid
- check (mid element == target) $a = b$
- if (a(m) < target) $a < b$

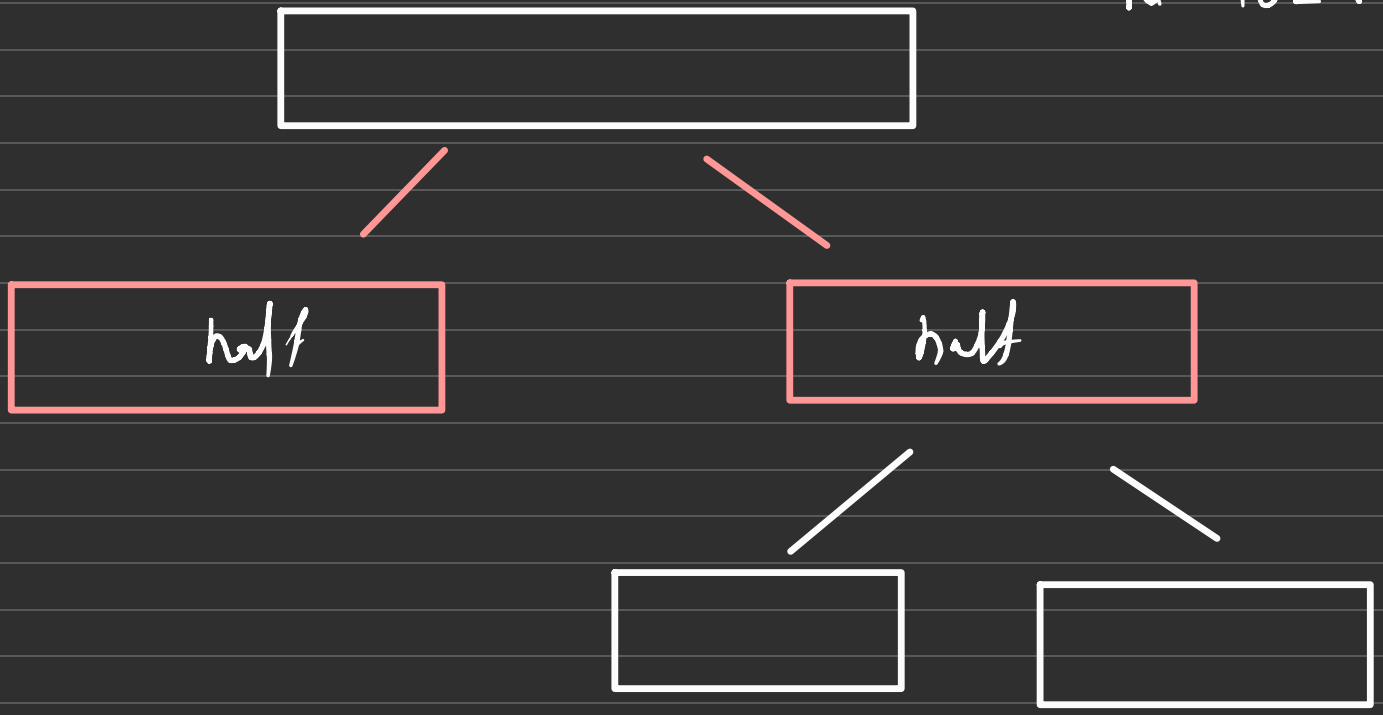
  s = m + 1
- if (a(m) > target) $a > b$

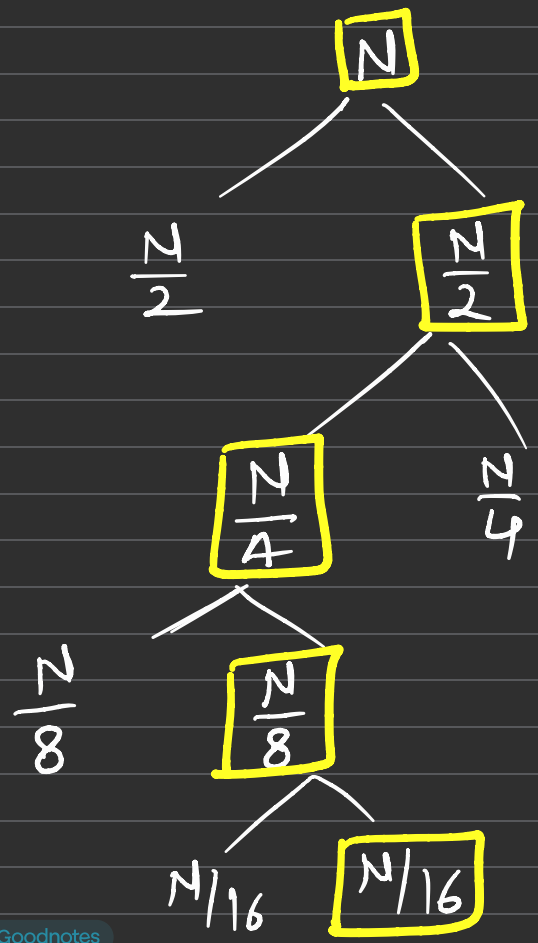
  e = m - 1

```
int binarySearch (arr[], N, target)
{
    int s=0, e= N-1
    while ( s<=e)
    {
        mid = (s+e)/2
        if (a(mid) == target)
            return index
        if (a(mid) < target)
            s = m+1
        else
            e = m-1
    }
    return -1
}
```

```cpp
int search(vector<int>& nums, int target) {

    int n = nums.size();

    // step 1: define the search space
    int start = 0, end = n - 1;

    while(start <= end){

        // step 2: find the mid
        int mid = (start + end) / 2;

        // a == b
        if (nums[mid] == target){
            return mid;
        }

        // a < b : move to right side
        if (nums[mid] < target){
            start = mid + 1;
        }

        // a > b : move to left
        else{
            end = mid - 1;
        }
    }

    return -1;
}
```

N = 1024





$N \rightarrow \frac{N}{2^0}$

$\frac{N}{2} \rightarrow \frac{N}{2^1}$

$\frac{N}{4} \rightarrow \frac{N}{2^2}$

$\frac{N}{8} \rightarrow \frac{N}{2^3}$

$\frac{N}{16} \rightarrow N/2^4$

$$\frac{N}{2^K} = 1$$

$$N = 2^K$$

$$\log N = \log 2^K$$

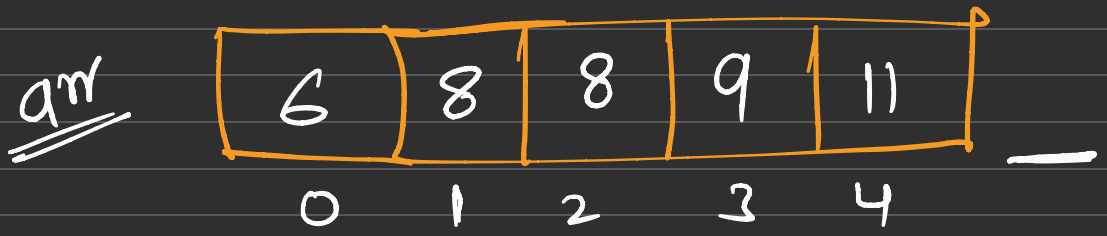
$$\log N = K \log 2$$

$$\frac{\log N}{\log 2} = K$$

$$O(\log_2 N)$$

$$K = \log_2 N$$

Lower Bound



LB(8) = 1 (Return index if elem. present)

LB(7) = 1

LB(12) = N

Lower Bound

N = 4

x = 10

ans =

(c)

(s)

(m)

s = 0

e = 3

m = 1

s = 2

e = 3

M = 2

s = 3

e = 3

m = 3

s = 4

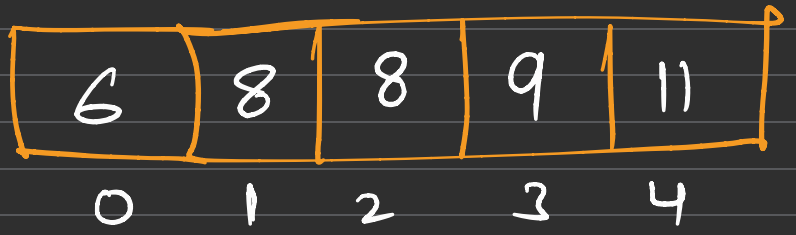
e = 3 ✗

3 ≥ 10

```
int LB (arr, N, x)
{
    ans = N
    s=0, e= N-1
    while (s ≤ e)
    {
        m = (s+e)/2
        if (a(m) ≥ x)
        {
            ans = a(mid)
            e = mid-1
        }
        else
            s = mid+1
    }
}
```

```cpp
int lowerBound(vector<int>& arr, int target) {

    int n = arr.size();

    int ans = n;
    int start = 0, end = n - 1;


    while(start <= end){

        // step 1: find the mid
        int mid = (start + end) / 2;

        if (arr[mid] >= target){
            ans = mid;
            end = mid - 1;
        }else{
            start = mid + 1;
        }
    }

    return ans;
}
```

Upper Bound



N = 5

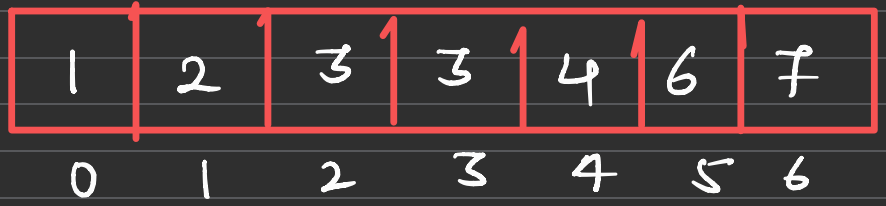
UB(6) = 1

UB(7) = 1

UB(12) = N

Same → LB

Upper Bound



N = 7

$x = 6$

LB

UB $>$ $x = 5$

```cpp
int upperBound(vector<int>& arr, int target) {
    int n = arr.size();

    int ans = n;
    int start = 0, end = n - 1;


    while(start <= end){

        // step 1: find the mid
        int mid = (start + end) / 2;

        if (arr[mid] > target){
            ans = mid;
            end = mid - 1;
        }else{
            start = mid + 1;
        }
    }

    return ans;
}
```

35. Search Insert Position

Solved

Easy | Topics | Companies | Re-do

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with `O(log n)` runtime complexity.

if data is sorted & you want to find something use BS

Example 1:

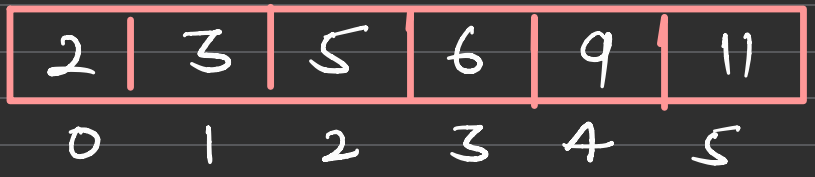
```
Input: nums = [1,3,5,6], target = 5
Output: 2
```

Example 2:

```
Input: nums = [1,3,5,6], target = 2
Output: 1
```

Example 3:

```
Input: nums = [1,3,5,6], target = 7
Output: 4
```

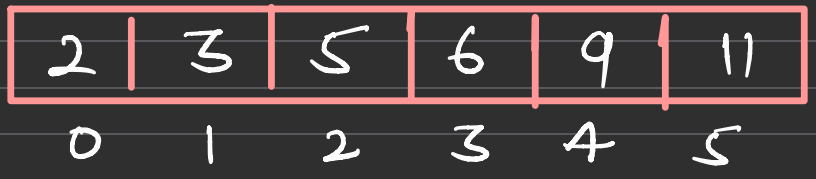


$x = 12$

- target present → index

- target is not present → index of first greater Element

```
int search ( arr, N, x)
{
    for ( i=0 ; i<N ; i++)
    {
        if ( arr(i) == x)
            return i
        else if (arr(i) > x)
            return i
    }
    return N
}
```



$x = 6$

$x = 7$

Find

- $x = 6 \rightarrow 3$
- $LB(6) = 3$
- $LB(7) = 9$
- $x = 7 \rightarrow 4$

```cpp
int lowerBound(vector<int>& arr, int target) {

    int n = arr.size();

    int ans = n;
    int start = 0, end = n - 1;


    while(start <= end){

        // step 1: find the mid
        int mid = (start + end) / 2;

        if (arr[mid] >= target){
            ans = mid;
            end = mid - 1;
        }else{
            start = mid + 1;
        }
    }

    return ans;
}
```

N = 6

s
m
e

x = 3

ans = ~~6~~ ~~2~~ 1

s = 0 e = 5 m = 2

s = 0 e = 1 m = 0

s = 1 e = 1 m = 1

s = 1 e = 0

x = 1

ans = ~~0~~ ~~2~~ 0

s = 0 e = 5 m = 2

s = 0 e = 1 m = 0

s = 0 e = -1

34. Find First and Last Position of Element in Sorted Array

Solved

Medium | Topics | Companies | Re-do

Given an array of integers `nums` sorted in non-decreasing order, find the starting and ending position of a given `target` value.

If `target` is not found in the array, return `[-1, -1]`.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]
```

Example 2:

```
Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]
```

N = 8

x = 3

startIndex = -1

endIndex = -1

```
for (i=0 ; i<N ; i++)
{
    if (a(i) == x)
    {
        if (startIndex == -1)
            startIndex = i

        endIndex = i
    }
}
```

N = 8

x = 3

(s)

(c)

(m)

s = 0

e = 7

m = 3

s = 0

e = 2

m = 1

s = 0

e = 0

m = 0

s = 1

e = 0

s = 0

e = 7

m = 3

s = 4

e = 7

m = 5

s = 4

e = 4

s = 5

e = 4

FO = ~~3~~

1

LO = ~~-1~~

~~3~~

4

```cpp
int findFirstOcc(vector <int>&nums, int n, int target){

    int ans = -1;

    int start = 0, end = n - 1;
    while(start <= end){
        // find the mid
        int mid = (start + end) / 2;

        if (nums[mid] == target){
            ans = mid;
            end = mid - 1;
        }

        else if (nums[mid] > target){
            end = mid - 1;
        }

        else{
            start = mid + 1;
        }
    }
    return ans;
}
```

```cpp
int findLastOcc(vector <int>&nums, int n, int target){
    int ans = -1;

    int start = 0, end = n - 1;
    while(start <= end){
        // find the mid
        int mid = (start + end) / 2;

        if (nums[mid] == target){
            ans = mid;
            start = mid + 1;
        }

        else if (nums[mid] > target){
            end = mid - 1;
        }

        else{
            start = mid + 1;
        }
    }
```

33. Search in Rotated Sorted Array

Solved

Medium | Topics | Companies | Re-do

There is an integer array `nums` sorted in ascending order (with **distinct** values).

Prior to being passed to your function, `nums` is **possibly left rotated** at an unknown index `k` (`1 <= k < nums.length`) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,5,6,7]` might be left rotated by `3` indices and become `[4,5,6,7,0,1,2]`.

Given the array `nums` **after** the possible rotation and an integer `target`, return *the index of* `target` *if it is in* `nums`*, or* `-1` *if it is not in* `nums`.

You must write an algorithm with `O(log n)` runtime complexity.

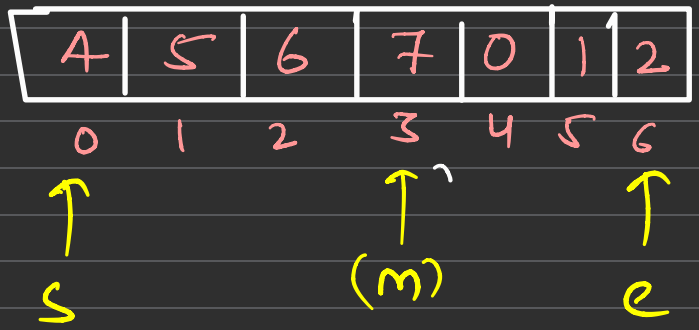
Example 1:

```
Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4
```

Example 2:

```
Input: nums = [4,5,6,7,0,1,2], target = 3
Output: -1
```

N = 7



s = 0

e = 6

m = 3

ans = 0

BS ⇒ SSS

(left) Part 1 = S → M → a(s) ≤ a(m)

(right) Part 2 = M → E → a(m) ≤ a(e)

```
s=0, e=N-1
while (s <= e)
{   m = (s+e)/2
    if (a(m) == x) return m
// left part
    if (a(s) <= a(m))
    {
        if (a(s) <= x && x <= a(m))
        {   end = mid-1
        }
        else
            start = mid+1
    }
```

```
else                    // right  (m) → (e)
{
    if (a[m] <= x && x <= a[e])
        s = m+1;
    else
        e = m-1;
}
return -1;
```

```cpp
int search(vector<int>& nums, int target) {
    int n = nums.size();

    int start = 0, end = n - 1;

    while(start <= end){

        // find the mid
        int mid = (start + end) / 2;

        // step 1: check if mid elem is equal to target
        if (nums[mid] == target){
            return mid;
        }

        // step 2: check which part is sorted i.e
        // left -> [start .... mid]
        // right -> [mid .... end]

        // left part
        if (nums[start] <= nums[mid]){
            // check if target lies in this part or not
            if (nums[start] <= target && target <= nums[mid]){
                end = mid - 1;
            }else{
                start = mid + 1;
            }
        }

        // right part
        else{
            // check if target lies in this part or not
            if (nums[mid] <= target && target <= nums[end]){
                start = mid + 1;
            }else{
                end = mid - 1;
            }
        }

    }

    return -1;
}
```

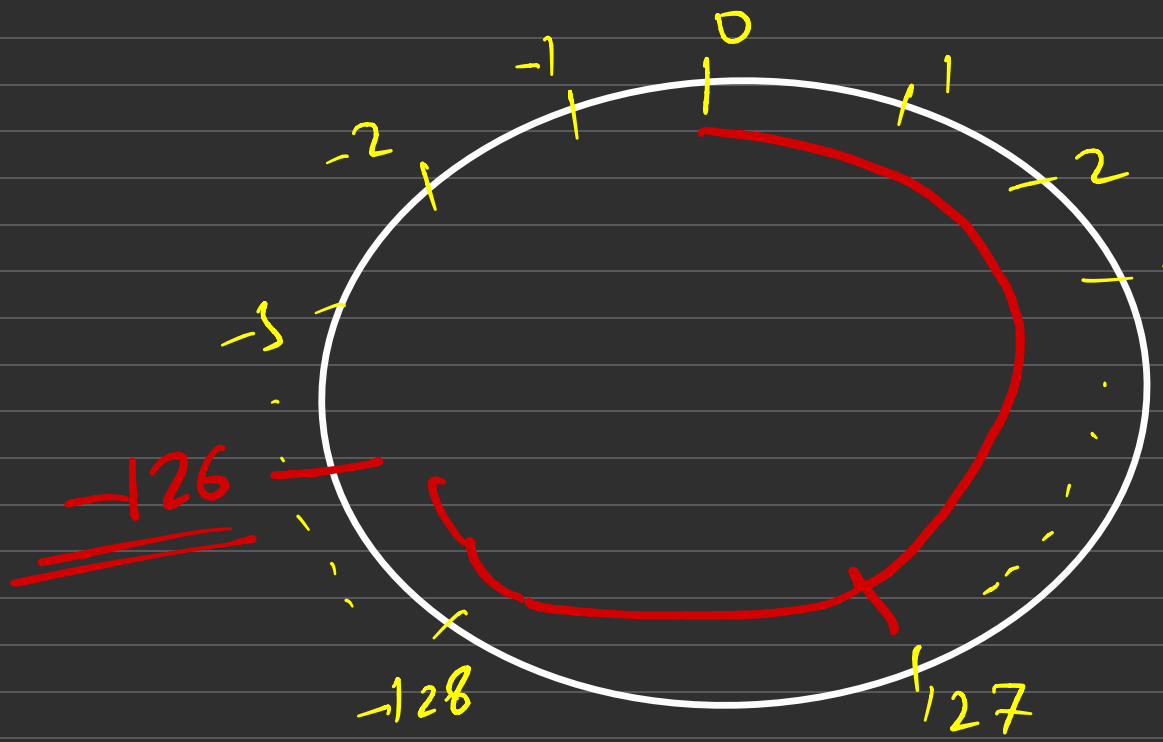
$$mid = (s+e)/2 = \frac{65510}{2}$$

→ $$mid = s + (e-s)/2$$

Range = int 2B = -32768 to 32767

$s = 32760$ $e = 32750$

Char IS = -128 to 127



127+3

Ch=130