Binary Search on Array

Rajat Sharma
Vasanth
Subhranil Kundu
Rajendra
Saurabh Ruikar
Garga Chakrabarty
Naval Oli
Madhan Kumar M S
amit khandelwal
Sarthak
Pankaj
Balaji S K
Purusharth A
Vishal Mosa
Sumit Adwani
Murali Mudigonda
Nikhil Pandey
sharath r
Sushant
Murali krishna Talluri
Sneha L
Bhaveshkumar
Venkata Sribhavana Nandiraju
suyash gupta
Gowtham
Sridhar Hissaria
Sanket Giri
Gokul Aditya S
Vimal Kumar
Shani Jaiswal
Gagan Kumar S
Shradha Srivastava
Abhishek Sharma

- Introduction
- Questions.
  - → Search for element k
  - → Search for first and last occurrence
  - → Single Element in an array
  - → Peak element
  - → Find any one local minima.

Current PSP

↓

65 %

Important Announcements

— Mock Interview Awareness ***

— AMA session by Ayush Sharma 25th Feb

- Ways to prioritize time.
- Backlog clearing tactics.
- Persona wise carcer discussion.

Introduction

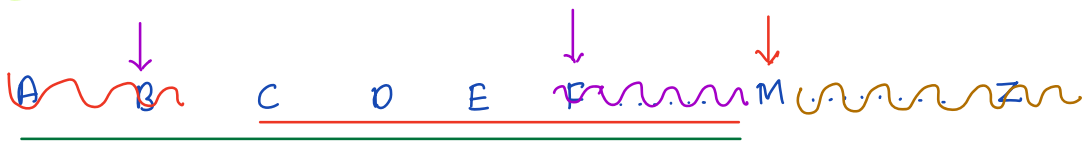
lost keys → MOM → what ? keys
MOM → where to find ? search space

Example

order {sorted}

- Word → { Dict , Book , Newspaper }
- Phone no. → { contacts , phone book }

DOG



Target —— Some item we are looking for.

search space where to look for an item.

Condition —— On which we discard our search space
Eg — see dictionary above.

Binary search —— Repeatedly divide the search space in half based on some condition till you find target or the search space is completed.

Q) Given a **sorted array** with distinct elements, search for index of an element K, if K is not present return -1 K = 12

	0	1	2	3	4	5	6	7	8	9
A[10] =	3	6	9	12	14	19	20	23	25	27

any = 3

Bruteforce Linear search TC : O(N)

$\forall i$ check if $A_i == K$ SC : O(1)

K = 12

	0	1	2	3	4	5	6	7	8	9
A[10] =	3	6	9	12	14	19	20	23	25	27

l = 2, r = 3, m

1) Identify the search space

```
l = 0
r = 9
while ( l <= r ) {
```

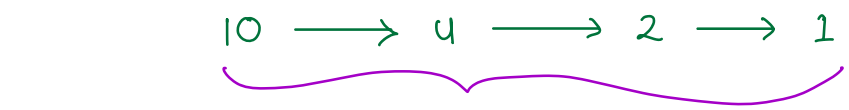
2) check if mid position can be my any

```
    mid = (l+r)/2*      // check below
    if ( A[mid] == k )  return mid
```

3) decide whether to go left or right

```
    if (A[mid] > k) {
        r = mid - 1     // go left
    }
    else { l = mid + 1 }  // go right
}
return -1
```

Search Space



log(N)

→ N → N/2 ..... 1 O(log(N))

How to correctly calculate mid ?

Our datatype can only store max int as 100

l = 97

r = 99

$m = \frac{(l + r)}{2}$ will overflow donot use

$mid = l + \left(\frac{r - l}{2}\right)$ // Exactly same as above

always use me above

$l + \frac{r}{2} - \frac{l}{2}$

$\frac{l}{2} + \frac{r}{2}$ $\left(\frac{l+r}{2}\right)$

Q> Given a sorted array of N elements, find Find first occurrence of given element K Return that index.

ans = 7

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
-5	-5	-3	0	0	1	1	5	5	5	5	5	5	5	8	10	10	15	15

Bruteforce Linear search TC : O(N)

$\forall i$ check if $A_i == K$ SC : O(1)

and return the first time.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
-5	-5	-3	0	0	1	1	5	5	5	5	5	5	5	8	10	10	15	15
						r	l											
							m											

K=5

1> Identify the search space

```
l = 0                ans = -1
r = N-1
while ( l <= r ) {
```

2> check if mid position can be my ans

```
    mid = l + (r-l)/2
    if ( A[mid] == K ) {
        ans = mid
        r = mid - 1          // go left
    }
```

3> decide whether to go left or right

```
    else if ( A[mid] < K ) {
```

```
        l = mid + 1          // go right
    }
    else {
        r = mid - 1          // go left
    }
}
return ans
```

TC: O(log(N))
SC: O(1)

HW. Find the last occurence

Given A[N] and an index
Return ∞ if index is out of bounds else return value at A[index].

	0	1	2	3	4
A =	1	5	3	9	4

index = 5 ⟶ ∞
= 3 ⟶ 9
= 0 ⟶ 1
= -1 ⟶ ∞

```
int safeGet(A[], index) {
    if (0 <= index && index < N) {
        return A[index]
    }
    return ∞
}
```

Q> Every element occurs twice except for 1 element
Find the unique element.

NOTE → Duplicate elements are adjacent to each other.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
3	3	1	1	8	8	10	10	19	6	6	2	2	4	4
l							m							r

ans = 19

TC : O(N)
SC : O(1)

Brutforce XOR all elements

1> Identify the search space

```
l = 0          ans = -1
r = N-1
while ( l <= r ) {
```

2> check if mid position can be my ans

```
    mid = l + (r-l)/2
    if ( A[mid] != A[mid-1] && A[mid] != A[mid+1] )
        return A[mid]
    }
```

use safeGet to take care of edge cases

3> Decide whether to go left or right

```
    // Find start of pair
    if ( A[mid] == A[mid-1] ) {
        mid = mid-1   // mid points to start of pair
    }
```

use safeGet

```
        if (mid % 2 == 0) {
            l = mid + 2
        }
        else {
            r = mid - 1
        }
    }
```

A =

0	1	2	3	4	5	6	7	8	9	10
3	3	4	4	1	1	6	5	5	2	2

Before single element pairs begin at even index

after single element pairs begin with odd index.

TC: O(log(N))

SC: O(1)

Break : 10:45

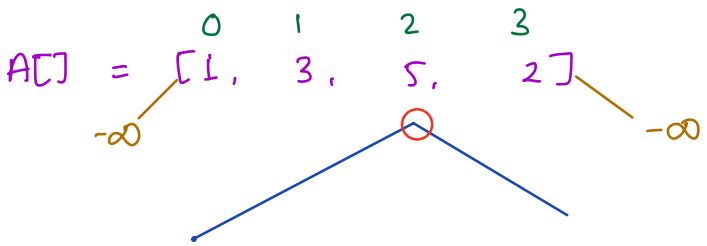
								m & l						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
3	3	1	1	8	8	10	10	19	6	6	2	2	4	4

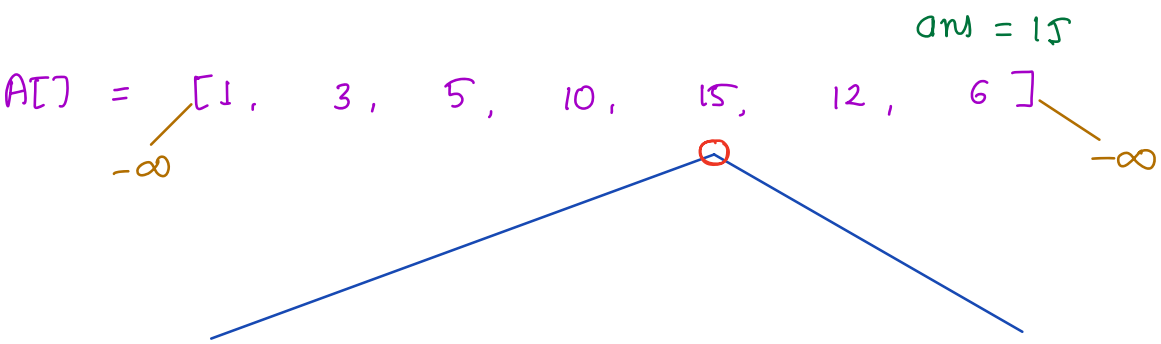
left	right	mid	isUnique	arr[m] == arr[m-1]	m % 2 == 1	m % 2 == 0
0	14	~~7~~ 6	NO	yes	NO	go right
8	14	11	NO	No	go left	NO
8	10	9	NO	NO	go left	NO
8	8	8	yes			

return

Peak Element

Q> Given an increasing decreasing array with distinct elements. Find max element.

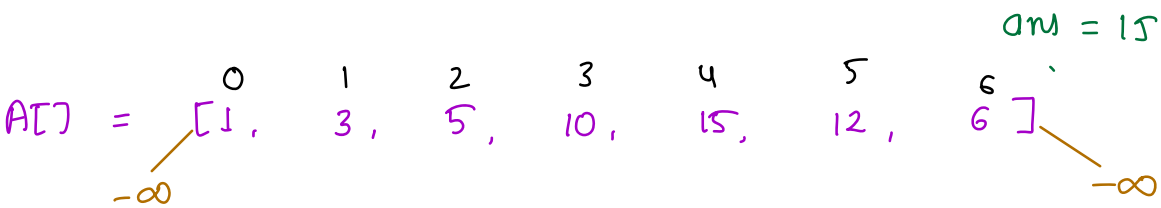




Bruteforce $\forall_i$ maintain max element in a variable.

TC: O(N)

SC: O(1)



1> Search space —— l = 0, r = 6

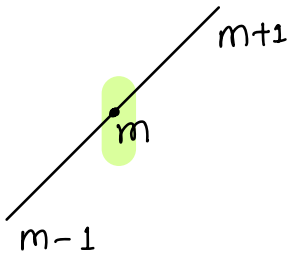
2> How to know mid is my ans ?

if A[mid] is ans ⟶ A[mid] > A[mid-1]

A[mid] > A[mid+1]

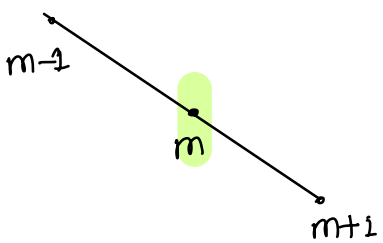
3> Decide whether to go left or right.

Case i



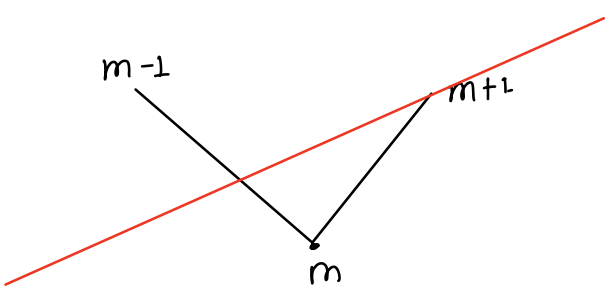
go right

Case ii



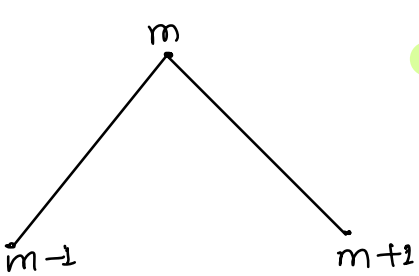
go left

Case iii



not possible

Case iv



return A[m]

```
int safeGet ( A[] , index ) {
    if ( 0 <= index && index < N ) {
        return A[index]
    }
    return -∞
}
```

```
// Step 1 Search space
l = 0    r = N-1
while ( l <= r ) {
    mid = l + (r - l) / 2
    // Check if mid can be my ans
    if ( safeGet ( A , mid ) > safeGet ( A , mid-1) &&
         safeGet ( A , mid ) > safeGet ( A , mid+1) )
        return A[mid]

    if ( safeGet ( A , mid ) > safeGet ( A , mid-1) )
        l = mid +1
    } else {
        r = mid -1
    }
}
return -1
```

TC : O(log(N))

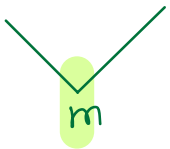
SC : O(1)

A =	0	1	2	3	4	5	6
	10	9	8	7	6	5	4
	l						
	k						

Google

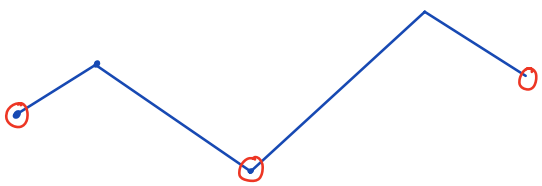
****

Q> Given an array of N distinct elements, find any **local minima** in the array

minima ⟶ 

	0	1	2	3	4	5	6
A[] =	3	6	1	0	9	15	8

ans = any of 3 0 8



A[] = 21 20 19 17 15 9 7 ans = 7

A[] = 5 9 15 16 20 21 ans = 5

A[] = 5 8 12 3 ans = either 5 or 3

Bruteforce Linear search and return min element.

TC: O(N)
SC: O(1)

⟶ anything better than this ⟹ binary search

```
int safeGet(A[], index) {
    if (0 <= index && index < N) {
        return A[index]
    }
    return ∞
}
```

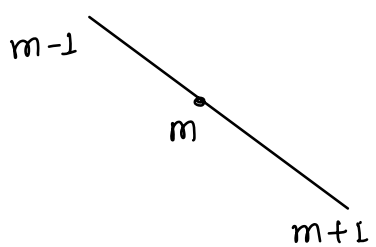
```
// step 1   search space
l = 0    r = N-1
while ( l <= r ) {
    mid = l + (r - l)/2
    // check if mid can be my ans
    if ( safeGet ( A , mid ) < safeGet ( A , mid-1) &&
         safeGet ( A , mid ) < safeGet ( A , mid+1) )
          return A[mid]

    if (safeGet ( A , mid ) < safeGet ( A , mid-1))
          l = mid +1      // go right.
    } else {
          r = mid -1      // go left.
    }
}
```

TC : O(log(N))

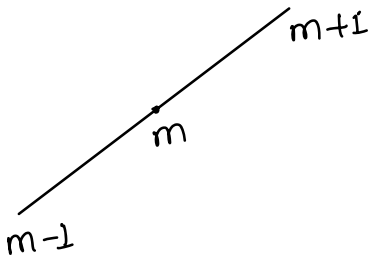
SC : O(1)

Case i



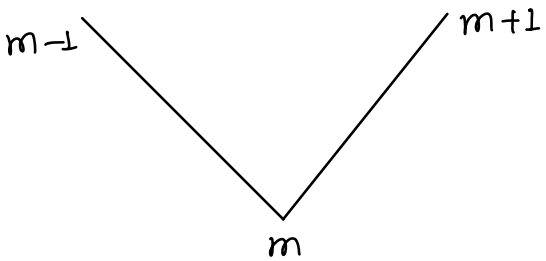
go right

Case ii



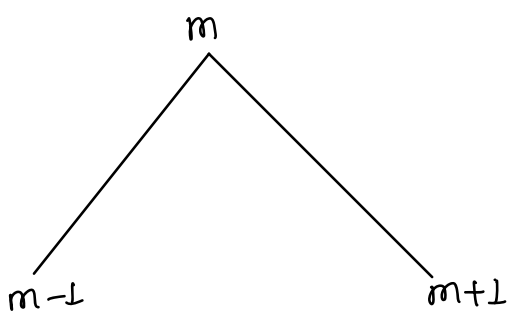
go left

Case iii



return A[mid]

Case iv)



either go left
or right

Doubt Session

	l							r
	0	1	2	3	4	5	6	7
A =	1	7	8	9	5	4	3	2