Today's Content

* Searching Basics.
* Why mid at half.
* Problems on binary search.

Searching : To look for something / element / answer.

relative goes missing $\xrightarrow[\text{police}]{\text{go to}}$ Police → Target : what to search
→ Search space : where to search

Search for a word : Book / Magazine / Newspaper / Dictionary.
Search space is ordered.

Eg: Dog

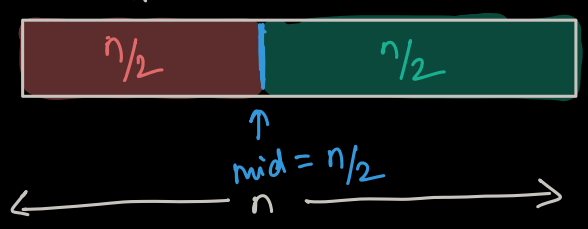
Dict has : A B C D E F G H . . . . X Y Z .

go right

words starting from H
go left

Core idea of Binary search

Land on mid



skip 50% of data

skip 33.33% of data in worst case.

obs: In binary search, search space is reduced by half

Binary Search: Divide search space into 2 parts & neglect one half of the search space by some condition.

Target Search space

Qn: Given a sorted array with distinct elements, search for index of an element K. Return -1, if K not found.

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

idea 1: Iterate array & search for K. → Linear Search.

TC: O(n), SC: O(1)

idea 2: Binary Search. : Target = K

Search Space = entire array.

Case 1:

a[mid] == K // found my element

return mid

Case 2:

a[mid] < K // a[mid] is less than given element

//Search on the right

Case 3:

a[mid] > K // a[mid] is greater than given element

//Search on the left.

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

K = 12

l	r	m = (l+r)/2	decision
0	9	4	a[4] = 14 > 12 go left r = m-1
0	3	1	a[1] = 6 < 12 go right l = m+1
2	3	2	a[2] = 9 < 12 go right l = m+1
3	3	3	a[3] = 12 == 12 found! return 3

Search for 13, same steps as above. Return -1 because l = 4, r = 3 l > r STOP!

// Pseudo code

```
int search (int a[], int N, int K) {
    l = 0, r = n-1
    while ( l <= r) {
        m = (l+r)/2
        if (a[m] == K) { return m }
        else if (a[m] > K) { r = m-1 }
        else { l = m+1 }
    }
    return -1
}
```

TC: O(log n)

SC: O(1)

Qn:- Given a sorted array. Find floor of a given number K.
[Return the idx]

greatest no. <= K

	0	1	2	3	4	5	6	7	8
ar[9] =	-5	2	3	6	9	10	11	14	18

K = 5 : 3 (2)
K = 4 : 3 (2)
K = 10 : 10 (5)
K = 20 : 18 (8)
K = -7 : no answer (-1)

idea 1: Iterate the array & search until elements are smaller or equal.
TC : O(n) , SC : O(1)

idea 2: Binary Search

Target: floor of K (greatest no. <= K)
Search Space: entire array.

Case 1: a[mid] == K return mid

Case 2: a[mid] < K // Go right

ans = mid , l = mid+1

↳ Store potential answer

Case 3: a[mid] > K // Go left

r = mid - 1

Dry - Run

	0	1	2	3	4	5	6	7	8
ar[9] =	-5	2	3	6	9	10	11	14	18
				l r					

K = 5

ans = ~~-1~~ ~~1~~ 2 (index)

l	r	m = (l+r)/2	a[m]	K = 5	
0	8	4	9 > 5	∴ go left	r = m-1
0	3	1	2 < 5	∴ go right	l = m+1
2	3	2	3 < 5	∴ go right	l = m+1
3	3	3	6 > 5	∴ go left	r = m-1
3	2	STOP [l > r]			

	0	1	2	3	4	5	6	7	8
ar[9] =	-5	2	3	6	9	10	11	14	18
	l			r					

K = -10

ans (index) = -1

l	r	m = (l+r)/2	a[m]	K = -10	
0	8	4	9 > -10	∴ go left	r = m-1
0	3	1	2 > -10	∴ go left	r = m-1
0	0	0	-5 > -10	∴ go left	r = m-1
0	-1	STOP [l > r]			

	0	1	2	3	4	5	6	7	8
ar[9] =	-5	2	3	6	9	10	11	14	18

K = 20

ans (index) = ~~-1~~ ~~4~~ ~~6~~ ~~7~~ 8

l	r	m = (l+r)/2	a[m]	K = 20	
0	8	4	9 < 20	∴ go right	l = m+1
5	8	6	11 < 20	∴ go right	l = m+1
7	8	7	14 < 20	∴ go right	l = m+1
8	8	8	18 < 20	∴ go right	l = m+1
9	8	STOP [l > r]			

Pseudocode :

```
int floor(int a[], int k) {
    l = 0, r = n-1, ans = -1
    while(l <= r) {
        m = (l+r)/2
        if (a[m] == k) {
            return m
        }
        else if (a[m] < k) { // Go right
            ans = m
            l = m+1
        }
        else { // Go left
            r = m-1
        }
    }
    return ans
}
```

TC: O(logn)

SC: O(1)

Break: 8:45am

Qn: Given a sorted array $a[N]$. Find the first occurence of given element K. Return its idx. (from left to right)

Eg:

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

$K = 5$: 7

idea: Iterate & find first match TC: $O(n)$, SC: $O(1)$

idea 2: Binary Search Target: first occurance of K.
Search Space: entire array.

Case 1: $a[mid] == K$

Store potential answer & go left because we need to find 1st occurence.

$ans = m$, $r = m-1$

Case 2: $a[mid] < K$ $\therefore$ go right

$l = m+1$

Case 3: $a[mid] > K$ $\therefore$ go left

$r = m-1$

Eg:

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
							l	r										

$K = 5$

ans = ~~-1~~ ~~9~~ 7

l	r	$m = (l+r)/2$	$a[m]$
0	18	9	$5 == 5$ ∴ go left : $r = m-1$
0	8	4	$0 < 5$ ∴ go right : $l = m+1$
5	8	6	$1 < 5$ ∴ go right : $l = m+1$
7	8	7	$5 == 5$ ∴ go left : $r = m-1$
7	6	STOP! $[l > r]$	

Code: To Do TC: $O(\log N)$ SC: $O(1)$

Follow Up Qn: Find last occurance of K

↳ instead of going left during EXACT MATCH, you go right $(l = m+1)$

Qn: Given an array of N distinct elements. Find idx of any one local minima in the array.

a no. smaller than its adjacent neighbours

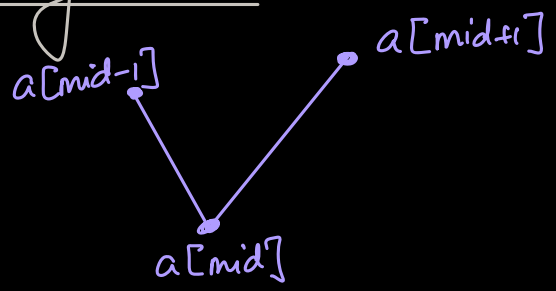
Eg:

	0	1	2	3	4	5	6
a[] =	3	6	1	0	9	15	8
b[] =	21	20	19	17	15	9	7
c[] =	5	9	15	16	20	21	
d[] =	5	8	12	3			

BF idea: for every element, check neighbours. TC: O(n)
SC: O(1)

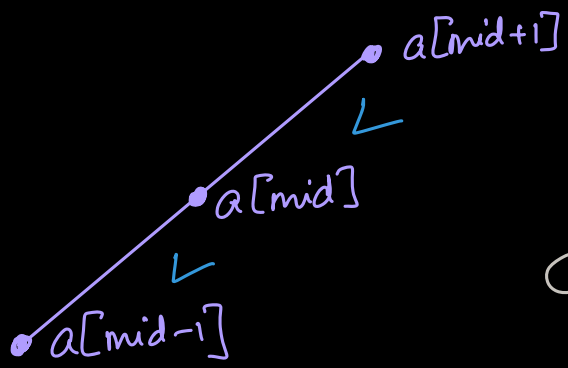
idea 2: Binary Search

Case 1:

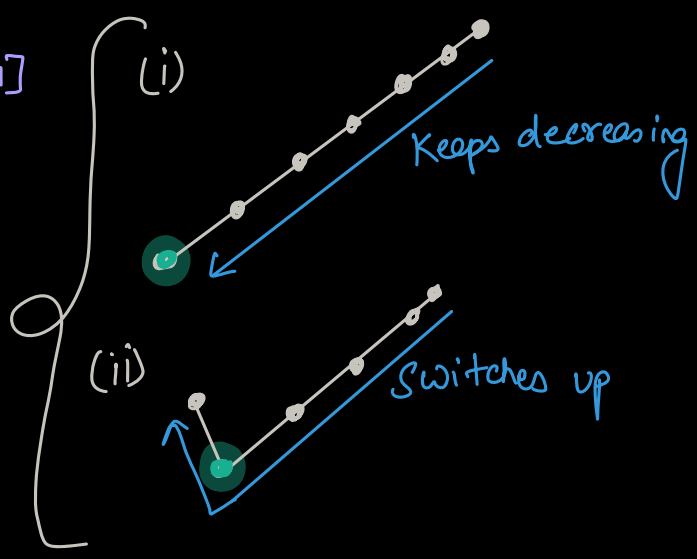


a[mid] < a[mid-1]
a[mid] < a[mid+1]
return mid

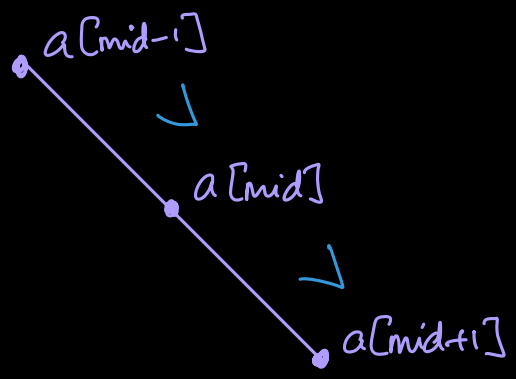
Case 2:

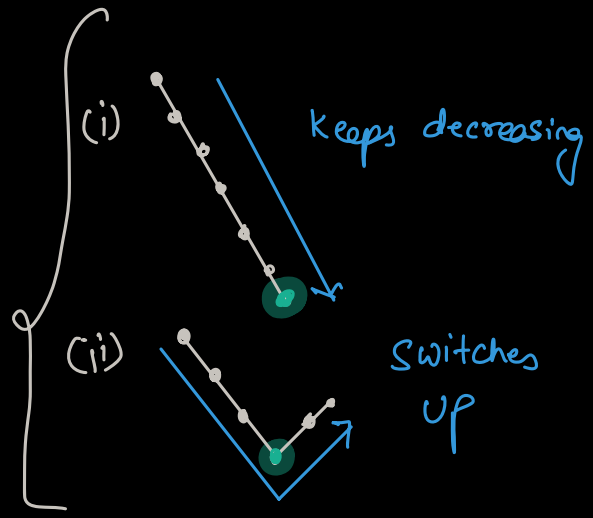


a[mid-1] < a[mid] < a[mid+1]
∴ Go left



Case 3:

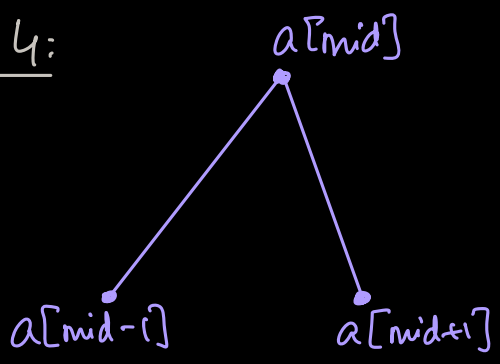




$a[mid-1] > a[mid] > a[mid+1]$

$\therefore$ Go right

Case 4:



// Can go to any side

$a[mid-1] < a[mid] > a[mid+1]$

// Pseudocode:

```
int localMinima(int a[]) {
    if (a.size() == 1) { return 0 }
    if (a[0] < a[1]) { return 0 }
    if (a[n-1] < a[n-2]) { return n-1 }
    l = 1, r = n-2
    while (l <= r) {
        m = (l+r)/2

        if (a[m] < a[m-1] && a[m] < a[m+1]) { // Case 1 V
            return m
        }
        else if (a[m] < a[m+1]) { // Case 2 go left
            r = m-1
        }
        else { // Case 3 + 4, go right
            l = m+1
        }
    }
}
```

TC: $O(\log n)$

SC: $O(1)$

Dry-Run

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

l	r	$m=(l+r)/2$	$a[m-1]$	$a[m]$	$a[m+1]$	Pattern
1	6	3	2	7	6	 right
4	6	5	6	4	1	 right
6	6	6	4	1	5	 found

return 6

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

l	r	$m=(l+r)/2$	$a[m-1]$	$a[m]$	$a[m+1]$	Pattern
1	6	3	2	7	6	 left
1	2	1	9	8	2	 right
2	2	2	8	2	7	 found

return 2