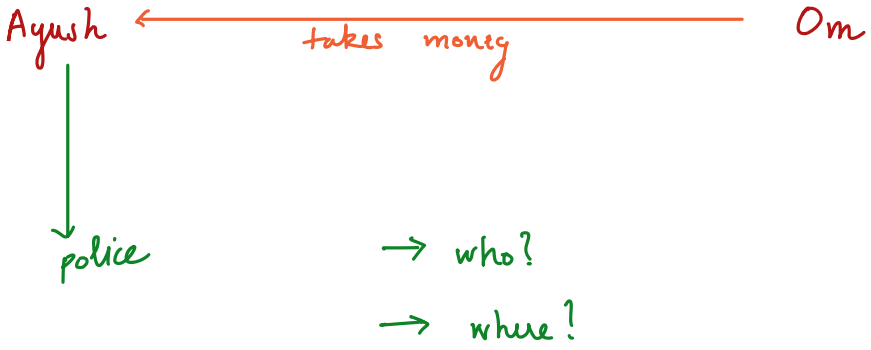
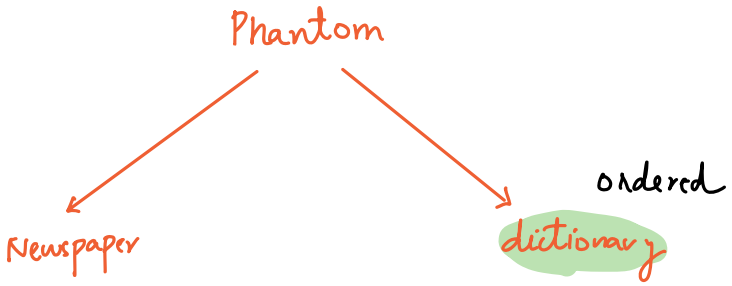


Target (what are we searching)
Search space (where you are looking for)



21

[1 5 7 9 11 | 13 15 21 27]

Mid

$\frac{N}{2}$ elements

[1 5 7 | 9 11 13 15 21 27]

$\frac{2N}{3}$ elements

Binary search

Q1) Given a sorted array of unique elements, search for a target & return its index, if not present return -1.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	3	5	7	9	10	11	13	15	17	19	30	35	40

↑ index

Target : 10 → 5

35 → 12

37 → -1 (Not present)

Target : 15 Search space: [0, N-1]

s	e	Mid ((s+e)/2)
0	13	6
s = mid+1 = 7	13	10
7	9	8

s e

```
s = 0                    e = N-1          # Define search space
while (s <= e) {
    mid = (s+e)/2
    if (A[mid] == target) {                # check if mid is
        return mid                         target or not
    }

    if (A[mid] > target) {                 #search left
        e = mid-1
    }
    else {                                 # Divide search space
        s = mid+1
    }
}
return -1
```

N
↓
N/2
↓
N/4
⋮
1

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

Condition to apply BS

: If you're able divide search space into two parts by rejecting one of the halves

Q2) Find the first occurance of an element in a sorted array. (with duplicates)

	s		m			m				
	0	1	2	3	4	5	6	7	8	9
A = {	2	2	5	5	5	5	8	10	10	13 }

target = 2

Target = 10 → T

5 → 2

8 → 6

Brute force:

iterate the array

A[i] == target && (i == 0 or A[i] > A[i-1])

```
for i = 0 : N
    if (A[i] == target)
        return i
```

BS

O(N)

```
s = 0          e = N-1
while (s <= e) {
    mid = (s+e)/2
    if (A[mid] == target && (mid == 0 or A[mid] > A[mid-1])) {
        return mid
    }

    if (A[mid] >= target) {
        e = mid-1                    # left
    }

    else {
        s = mid+1                    # right
    }
}
```

TC: O(log N)

SC: O(1)

Amazon

Q3) Given a sorted array where every element appears twice except for one element that appears one time, find that unique element.

```
          0  1  2  3  4  5  6
A = { 2  2  5  7  7  10 10 }        output : 5
```

{even, odd} {odd, even}

mid

```
          0  1  2  3  4  5  6  7  8  9  10
A = { 1  1  5  5  7  7  10 13 13 18 18 }        output: 10
```

Target !

Ap1) XOR of all elements O(N)

Ap2) linear search and check

```
(i == 0 or A[i] != A[i-1]) && (i == n-1 or A[i] != A[i+1])
```

5 7 7 8 8 7 7 8 8 5

```
                          m
                  m       s       e       m
          0  1  2  3  4  5  6  7  8  9  10
A = { 1  1  5  5  7  7  10 13 13 18 18 }   Ans : 10
```

index

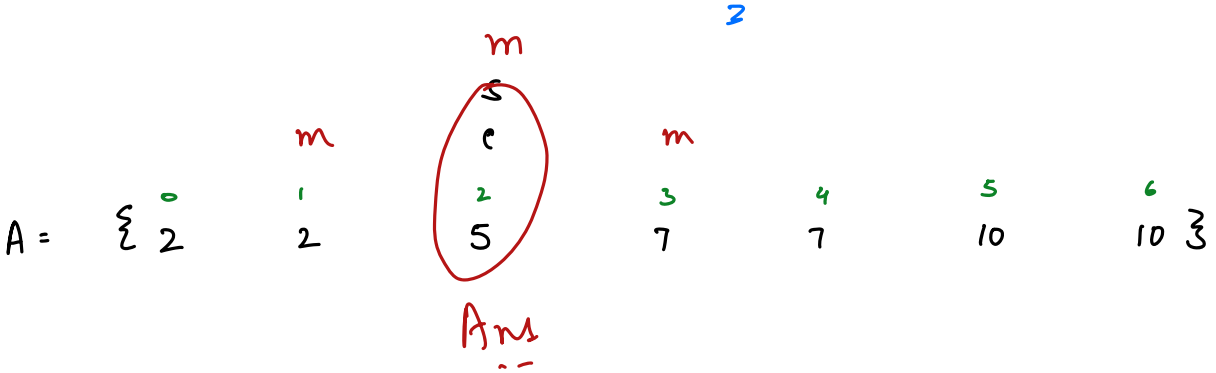
Before unique element: {even odd}

index

After unique element: {odd even}

t_i → first twin index

```
if (A[m] == A[m-1]) {
    ti = m - 1
    if (ti % 2 == 0) {
        s = m+1
    }
    else {
        e = m-1
    }
}
else {
    ti = m
    if (ti % 2 == 0) {
        s = m+1
    }
    else {
        e = m-1
    }
}
```



Ans

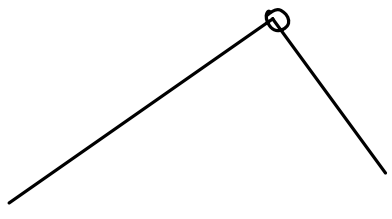
```
s=0                                  e = N-1
while (s<=e){
    mid = (s+e)/2
    # check for mid
if (i==0 or A[i] != A[i-1]) && (i==n-1 or A[i] != A[i+1]){
|       return mid
}

    if (A[m] == A[m-1]){
    |       ti = m-1                    # find first twin
    }                                       index
    else {
    |       ti = m
    }

    if (ti%2==0){                       # left or right
    |       s = m+1
    }.
    else {
            e = m-1
    }

}
```

Break (10:35 - 10:45)



Q Find peak element in an increasing - decreasing array.

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

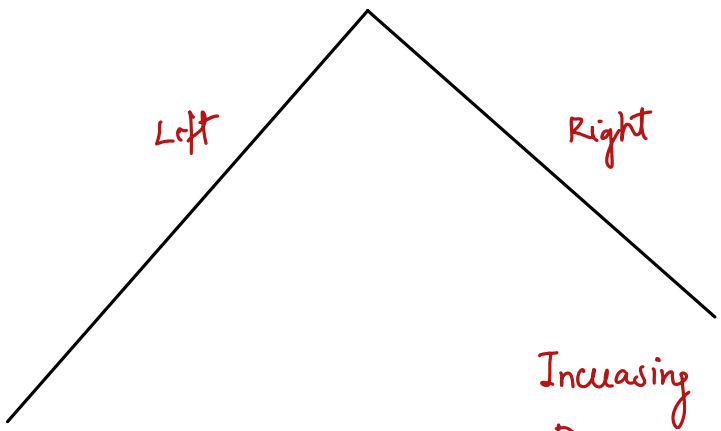
Ans : 10

0	1	2	3	4	5	6
1	5	9	11	7	6	2

Ans : 11

Approach 1) Iterate the array and find max

TC : O(N)



Increasing → Right

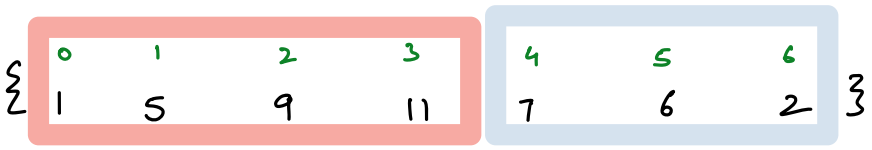
Decreasing → left

```
s = 0                                   e = N-1

while (s <= e) {
    m = (s+e)/2
    # A[m] > A[m+1]   and   A[m] > A[m-1]

    if ((m==0 or A[m] > A[m-1]) &&
        (m==n-1 or A[m] > A[m+1])) {
        return m
    }
    if (m==0 or A[m-1] < A[m]) {        # Right
        s = m+1
    }
    else {
        e = m-1
    }
}
```

Search for a value in increasing decreasing array

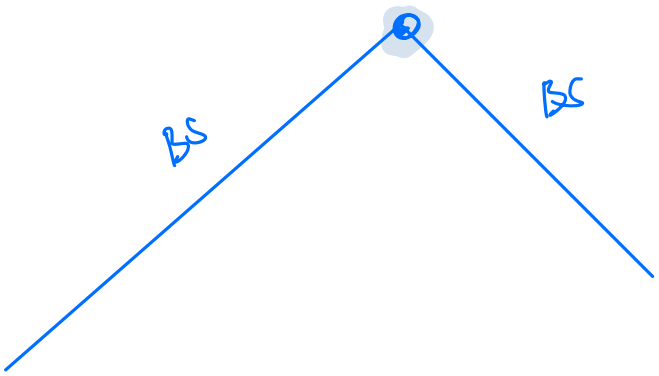


target = 6

peak element $\log N$

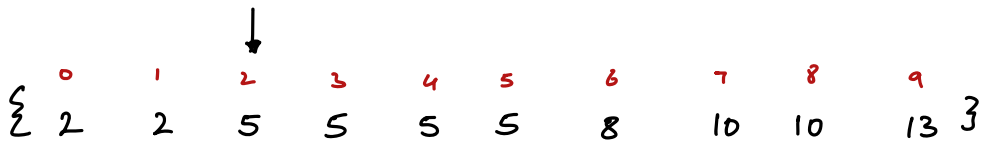
BS: $\log N$

TC: $O(\log N)$



a sorted

Q Given array with repeatitions, count total occurances of a num



count (5) = 4

A[mid] == target && (m == n-1 or A[mid] != A[mid+1])

first occurance — l

Last occurance — r

[l r] ⟶ r - l + 1

Done!