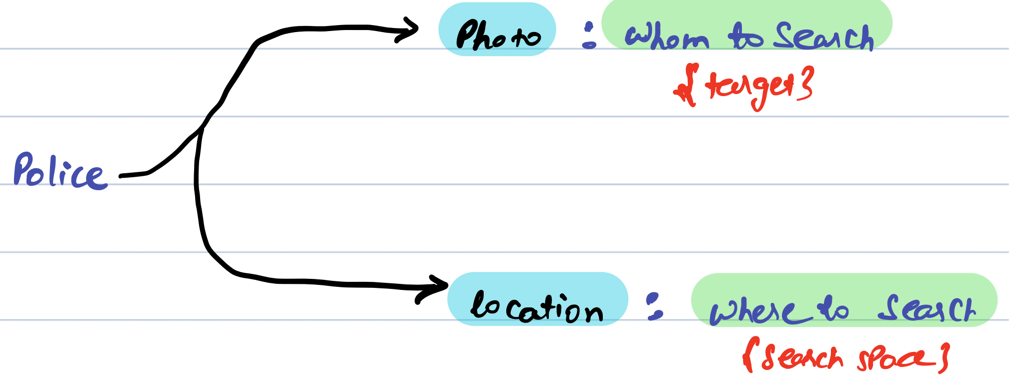
Today's agenda

- Searching basics
- Why mid at half
- Search in sorted array
- floor in a sorted array
- Every element occurs twice except for 1.

Story:



Police → Photo : whom to Search
{target}

Police → location : where to Search
{search space}

Ex:

(whom to search)	(where to Search)
word (AlgoPrep)	{NewsPaper / Book / Dict}
Contact number (HR)	{Telephone directory / Linkedin}

↳ your search space is ordered, Searching becomes easier.

Note: Binary Search Can be Possibly applied even if array is not Sorted.

→ Divide your search space in 2 halfs, if we can discard 1 half, we can apply BS.

Q) Given a sorted arr[N] search if K is Present or not?

	0	1	2	3	4	5	6	7	8	9
arr[10]:	{ 4	7	10	13	15	20	21	24	26	28 }

K=13
↓
true

//idea 1

↳ linear search.

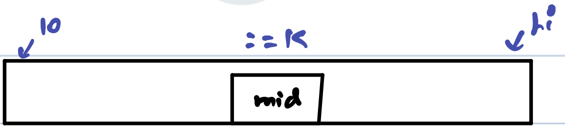
T.C: O(N) S.C: O(1)

//idea2 → binary search

→ search space: array

↳ target : K

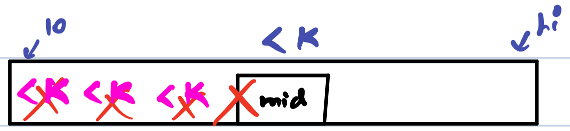
Case 1:



$mid = \frac{(lo+hi)}{2}$

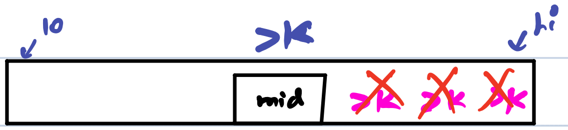
```
if (arr[mid] == K){ return true; }
```

Case 2:



```
if (arr[mid] < K){
    discard left side / search on right.
}
```

Case 3 :



```
if (arr[mid] > K){
    discard right side / Search on left.
}
```

				lo hi							
	0	1	2	3	4	5	6	7	8	9	
arr[10]:	{ 4	7	10	13	15	20	21	24	26	28 }	K=13

lo	hi	m = $\frac{lo+hi}{2}$	
0	9	4	if (arr[m] > K): Search on left ; hi=m-1
0	3	1	if (arr[m] < K): Search on right: lo=m+1
2	3	2	if (arr[m] < K); Search on right: lo=m+1
3	3	3	if (arr[m]==K): return true;

			hi	lo							
	0	1	2	3	4	5	6	7	8	9	
arr[10]:	{ 4	7	10	14	15	20	21	24	26	28 }	K=13

lo		hi	m = $\frac{lo+hi}{2}$	
0	<=	9	4	if (arr[m] > K): Search on left ; hi=m-1
0	<=	3	1	if (arr[m] < K): Search on right: lo=m+1
2	<=	3	2	if (arr[m] < K) ; Search on right : lo=m+1
3	<=	3	3	if (arr[m] > K) ; Search on left : hi=m-1
3		2	→ exit	

//Psuedo Code

```
boolean Search (int arr[N], int k){
        int lo = 0;
        int hi = N-1;

        while (lo <= hi) {
            int m = (lo + hi) /2;

            if (arr[m] == k){
                return true;
            }
            else if (arr[m] < k){
                lo = m+1;
            }
            else {
                hi = m-1;
            }
        }
        return false;
}
```

T.C: O(logN)

S.C: O(1)

$N \rightarrow \frac{N}{2} \rightarrow \frac{N}{4} \rightarrow \frac{N}{8} \cdots 1$

Q) Given a sorted arr[N], find floor of given num k.

↳ just smaller or equal (greatest no. <=k in arr[])

Ex: arr[9]: { -4 3 4 7 10 11 12 15 19 }

	0	1	2	3	4	5	6	7	8
arr[9]:	-4	3	4	7	10	11	12	15	19

k=5 : 4

k=7 : 7

k=11 : 11

//idea1

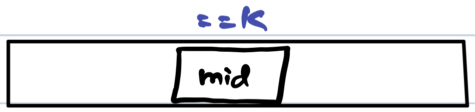
↳ linear search

T.C: O(N)   S.C: O(1)

//idea2

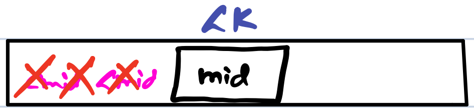
↳ binary search

Case 1:



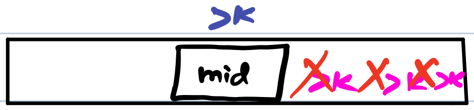
if (arr[mid]==k) {return k; }

Case2:



```
if (arr[mid] < k){
    ans= arr[mid];
    discard left / go to right
}
```

Case 3:



```
if (arr[mid] > k) {
        ↳ discard right | go to left
}
```

//PSuedo Code

T.C: O(logN)

S.C: O(1)

```
int floor (int arr[N], int k ){
    int lo = 0;
    int hi = N-1;
    int ans = -∞;

    while ( lo <= hi ){
        int m = (lo+hi)/2;

        if (arr[m] == k){
            return k;
        }
        else if (arr[m] < k){
            ans = arr[m];   lo = m+1;
        }
        else{
            hi = m-1;
        }
    }
    return ans;
}
```

hi lo

	0	1	2	3	4	5	6	7	8	
Ex: arr[9]: {	-4	3	4	7	10	11	12	15	19 }	K=6
				m						

ans = $-\infty$

lo	hi	m	
0	8	4	: if (arr[m] > K) → go to left : hi=m-1
0	3	1	: if (arr[m]<K) → ans=3 → go to right lo=m+1
2	3	2	: if (arr[m]<K) → ans=4 → go to right lo=m+1
3	3	3	: if (arr[m]>K) → go to left hi=m-1

∴ ans=4

→ Break till 9:44 Pm

Q) Every element occurs twice except for 1, find unique element.

Note: duplicates are adjacent to each other

Ex: arr[15]: 4 4 1 1 9 9 11 11 20 7 7 3 3 5 5

//idea1

↳ Take xor of all elements.

T.C: O(N)   S.C: O(1)

//idea2

↳ Binary search

arr[15]:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	4	1	1	9	9	11	11	20	7	7	3	3	5	5

Pre single occ: Numbers are starting from even index

Post single occ: Numbers are starting from odd index

Case 1:

```
if (arr[mid] != arr[mid-1] && arr[mid] != arr[mid+1]){
        return arr[mid];
}
```

Case 2: my mid is at first occurrence

```
if (mid % 2 == 0){
     discard left | go to right
}
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	4	1	1	9	9	11	11	20	7	7	3	3	5	5

Case 3:

my mid is at first occurrence

```
if (mid % 2 == 1){
     reject right | go to left
}
```

Tracing

mid (at index 3); lo, hi (at index 8)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
arr[15]:	4	4	1	1	9	9	11	11	20	7	7	3	3	5	5

lo	hi	mid	A	
0	14	7	m=6	m%2==0, go to right, lo=mid+2
8	14	11	m=11	m%2==1, go to left, hi= mid-1
8	10	9	m=9	m%2==1, go to left, hi=mid-1
8	8	8	→ return arr[mid];	

* How to make sure that mid lands at 1st occ.

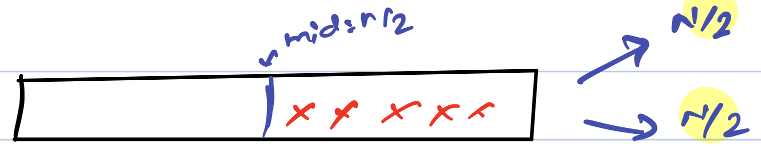
```
if (arr[mid] == arr[mid-1]){
        mid--;
}
else {
✗        //No Change
✗
}
```

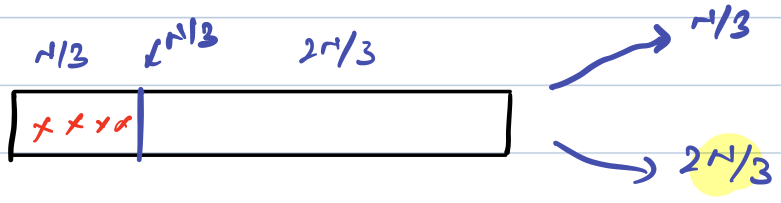
//PSuedo Code

T.C: O(logN)

S.C: O(1)

```
int    unique (int arr[N]){
        //0th inden      if(arr[0]!=arr[1]){return arr[0];}

        // last inden     if (arr[n-1]!=arr[n-2]){return arr[n-1];}

        int lo = 2;
        int hi = N-3;

        while (lo<=hi){
            int mid = (lo+hi)/2;

            if(arr[mid]!=arr[mid-1] && arr[mid]!=arr[mid+1]){
                    return arr[mid];
            }

                if (arr[mid] == arr[mid-1]){mid--;}

                if (mid%2 ==0){lo=mid+2;}
                            else {hi=mid-1;}

        }

            return -1;
}
```


mid = n/2
n/2
n/2


n/3
n/3
2n/3
n/3
2n/3