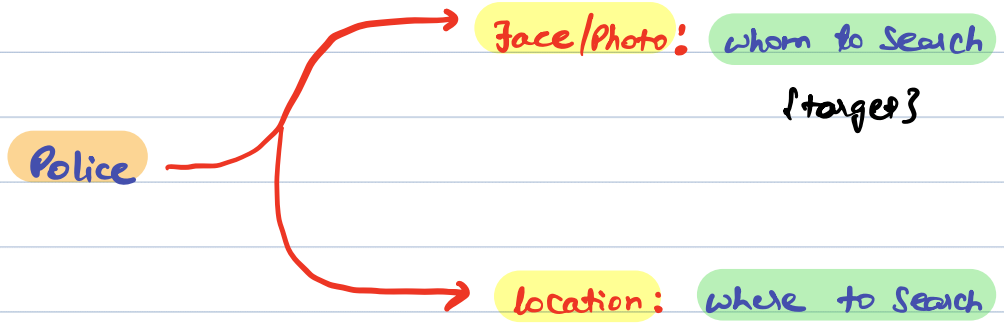
Today's agenda

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

Story:



Ex:

{whom to search}
word (AlgoPrep)
Contact Numbers (HR)

{where to search}
{Newspaper / Book / Dict}
{ LinkedIn / Telephone direct}

↳ Search space is ordered, searching becomes easier.

→ linear search: Searching one by one.

→ binary search: Divide your search space in 2 parts, if we can discard one half, we can apply BS.

↳ Array should not necessarily be sorted.

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

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

K=13 → true

//idea 1

↳ linear search

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

//idea 2

↳ binary search → search space: given 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.

}
```

arr[10]: { 4 7 10 13 15 20 21 24 26 28 } k = 13

lo	hi	m = (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

//Pseudo 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;
}
```

n → n/2 → n/4 → ... → 1

T.C: O(log n)

S.C: O(1)

```
int lo = 0;
int hi = N-1;
```

arr[10]: {4, X, 10, 14, X, 20, 21, 24, 26, 28}   (indices 0–9; hi ↓ at index 2, lo ↘ at index 3)

K = 13

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

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

lo	hi	m = (lo + hi)/2	go to
0	9	4	if (arr[m] > K): left
0	3	1	if (arr[m] < K): right
2	3	2	if (arr[m] < K): right
3	3	3	if (arr[m] > K): left
3	2	→ exit	

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

K=5 : 4

K=7 : 7

K=13 : 12

//idea1

↳ linear search

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

//idea2

↳ binary search

Case 1:

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

Case 2:


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

Case 3:


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

//Pseudo Code

```
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){
            if (arr[mid] > ans){ ans = arr[mid]; }
            lo = mid;
        }
        else{
            hi = m-1;
        }
    }

    return ans;
}
```

T.C: O(logN)

S.C: O(1)

ans: ~~0~~ 3 4

```
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;
}
```

arr[9] : { ~~-4~~ ~~-3~~ ~~-1~~ ~~1~~ ~~3~~ ~~10~~ ~~11~~ ~~12~~ ~~15~~ ~~19~~ } k = 6

(0 1 2 3 4 5 6 7 8; hi → 3, m → 3)

lo	hi	m	
0	8	4	if (arr[m] > k): go to left
0	3	1	if (arr[m] < k): go to right
2	3	2	if (arr[m] < k): go to right
3	3	3	if (arr[m] > k): go to left
3	2	→ exit	

↳ ans = 4

Break till 9:47 PM

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

Note: duplicates are adjacent to each other

Ex: 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

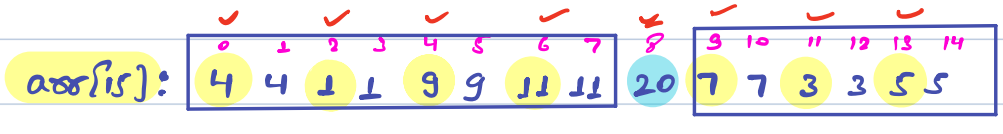
//idea1

↳ Take xor of all elements.

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

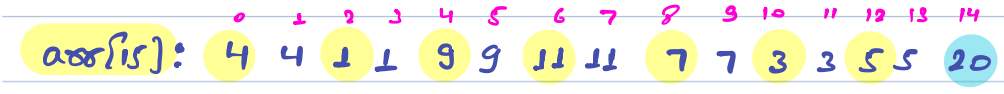
//idea2

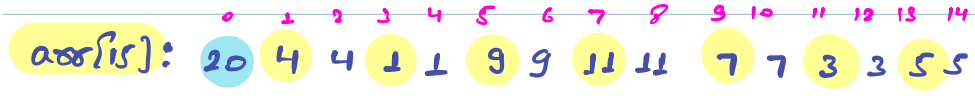
↳ binary search



For single no. occ: Numbers are starting at even idx

Post single no. occ: Numbers are starting at odd idx





Case 1:

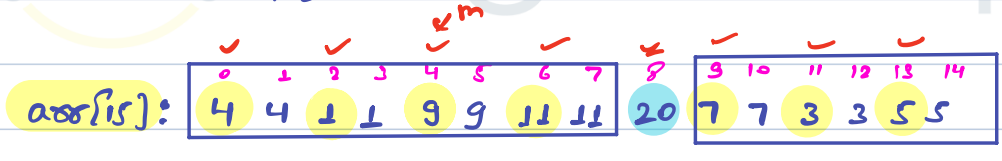

```
if (arr[mid] != arr[mid-1] && arr[mid] != arr[mid+1]){
    return arr[mid];
}
```

→ my mid is at 1st occ. of any number.

↳ if (arr[m] == arr[m+1])

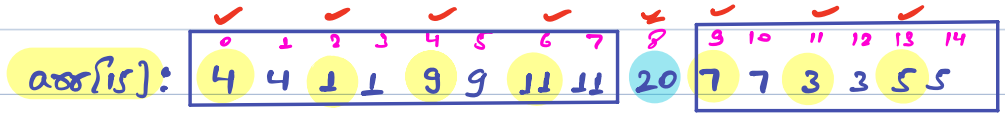
Case 2:

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



Case 3: → my mid is at 1st occ. of any number.

```
if (mid % 2 == 1){
    discard right / go to left.
}
```



→ my mid is at 2nd occ. of any number.

↳ if (arr[m] == arr[m-1])

do mid-- and then solve above cases.

//Pseudo code

```
int uniqueElement(int arr[N]){
    //0th index
    if(arr[0] != arr[1]){return arr[0];}
    //N-1th index
    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;}
    }
}
```

T.C: O(logN)

S.C: O(1)

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

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

hi = 10, m = 8

```
//0th index   if (arr[0] != arr[1]){return arr[0];}
//N-1th index 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; }
}
```

lo	hi	mid	*	
2	12	7	m:6	m%2==0, go to right
8	12	10	m:9	m%2==1, go to left
8	8	8		

↳ arr[mid]