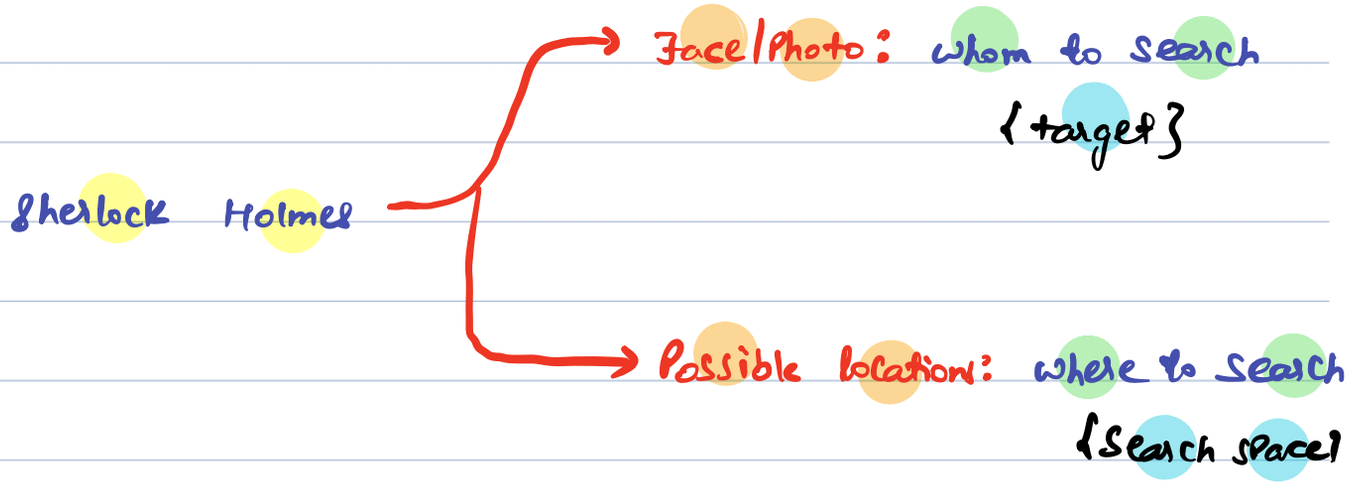
Today's agenda

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

Story:



Ex:

{Target}

Algobot{wood}

Contact no.{HR}

{search space}

{Newspaper/Dict/novel}

{linkedin/naukri.com/HR Tele Phonedirect}

↳ if search space is ordered, searching becomes easier.

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

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

K=13

Idea 1

↳ linear search: Searching one by one.

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

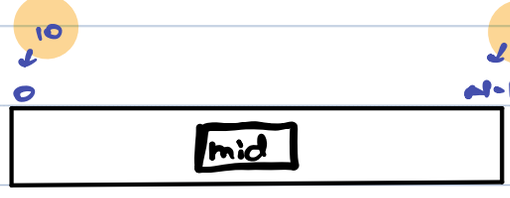
Idea 2

↳ Binary search: Divide your search space in 2 half, if we can discard 1 half, Binary search can be applied.

Search space: Array

Target: K

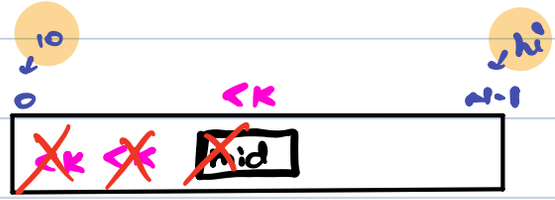
Case 1:



→ mid: $\frac{lo+hi}{2}$

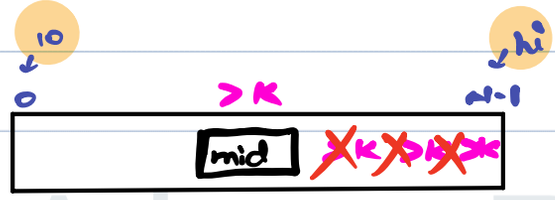
```
if (arr[mid] == K) { return true; }
```

Case 2 :




```
if (arr[mid] < k){
    discard left side | search on right
}
```

Case 3 :




```
if (arr[mid] > k){
    discard right side | search on left.
}
```

arr[10]: { 4 7 10 13 15 20 22 24 26 28 } k = 13

lo = 0, hi = 9, m = 4
if (arr[m] > k): search on left → hi = m-1

lo = 0, hi = 3, m = 1
if (arr[m] < k): search on right → lo = m+1

lo = 2, hi = 3, m = 2
if (arr[m] < k): search on right → lo = m+1

lo = 3, hi = 3, m = 3
if (arr[m] == k): return true

//Pseudo code

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

    while (lo <= hi ){
        int m = (lo + hi)/2 ; [lo + (hi-lo)/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 → N/8 - - - - - - - - → 1

T.C: O(log N)

S.C: O(1)

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

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

Eg: arr[9] = { -4 3 4 7 10 11 12 15 19 }
(0 1 2 3 4 5 6 7 8)

K=5 : 4

K=10 : 10

K=13 : 12

Idea1

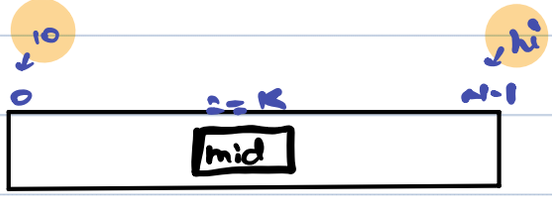
↳ linear search

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

Idea2

↳ binary search

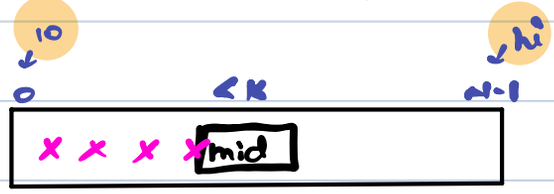
case 1:




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

K = 15

case 2



ans

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

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 = -∞; //Integer.MIN_VALUE

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

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

    return ans;
}
```

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

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

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

k = 6

arr[9] = { 1 3 4 7 10 11 12 15 19 }
(indices 0 1 2 3 4 5 6 7 8; hi = 10 ↓ 2 ↓ 3)

ans = -∞ ⇒ 4

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

return ans;

4 ans = 4

---

① → not correct

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

lo = 1 hi = $2^{31}-1$

$m = \frac{1 + 2^{31}-1}{2} = \frac{2^{31}}{2}$ ← random val

random val

② correct

```
int m = lo + (hi-lo)/2
```

4 to save overflow

lo = 1 hi = $2^{31}-1$

$m = 1 + \frac{2^{31}-1-1}{2}$

$= 1 + \frac{2^{31}/2}{2}$ → ind.

=

Break till 10:50 Pm

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

Note: dulliates are adjacent to each other

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

ans: 20

Idea

↳ Taking xor of all elements

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

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

left side of Single occ no: numbers are starting at even idx.

right side of " " : numbers are starting at odd idx.

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

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

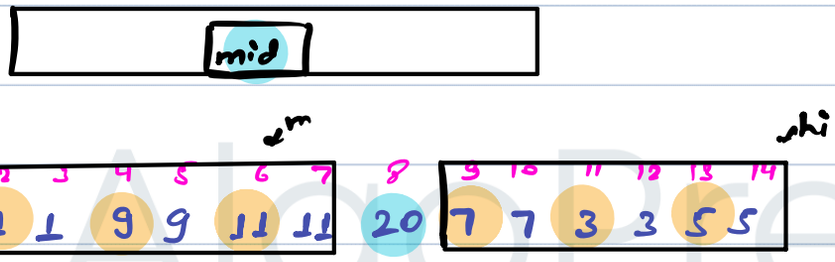
→ mid is at 2nd occ.

Case 1:


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

first occ: arr[m] == arr[m+1]

Case 2:




```
if (m%2==0){
    reject left / go to right
}
```

left side of single occ no: numbers are starting at even idx.

Case 3:


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

right side of single occ no.: numbers are starting at odd idx.

→ my mid is at 2nd occ. of any no.

↳ if (arr[m] == arr[m-1])

↓↓

do m--;

//Pseudo code

```
int uniqueElement (int arr[N]) {
    if (arr[0] != arr[1]) { return arr[0]; }
    if (arr[N-1] != arr[N-2]) { return arr[N-1]; }

    int lo = 2;
    int hi = N-1;

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

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

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

    return -1;
}
```

T.C: O(logN)

S.C: O(1)

```
int uniqueElement(int arr[N]) {
    if (arr[0] != arr[1]) { return arr[0]; }
    if (arr[N-1] != arr[N-2]) { return arr[N-1]; }

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

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

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

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

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

    return -1;
}
```

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

hi = 10

lo	hi	m	a[m]	
2	12	7	6	even, go to right
8	12	10	9	odd, go to left
8	8	8	20	

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