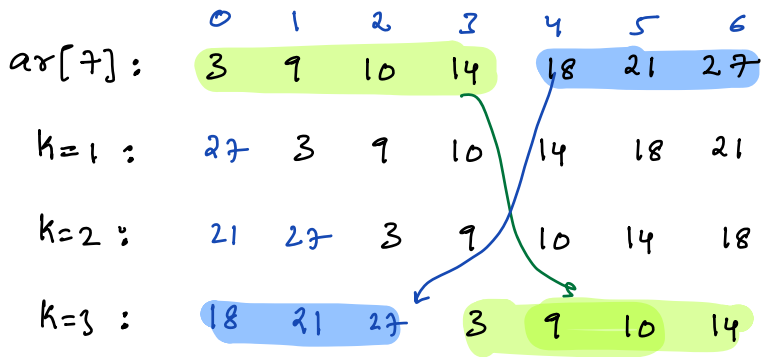
Todays Content:

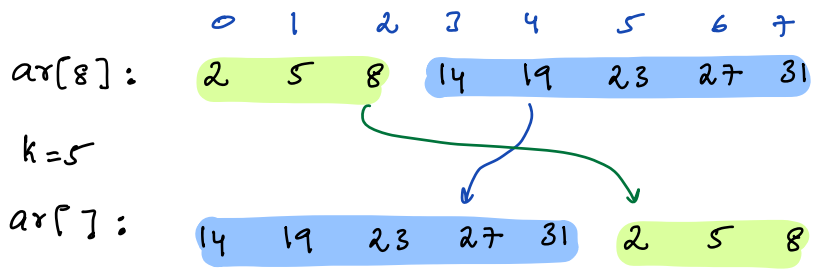
a) Search in a rotated sorted arr[]

b) Search in repeated arr[]

c) Sqrt()

Pre-requisits: Rotate arr[] by k times from end to start →



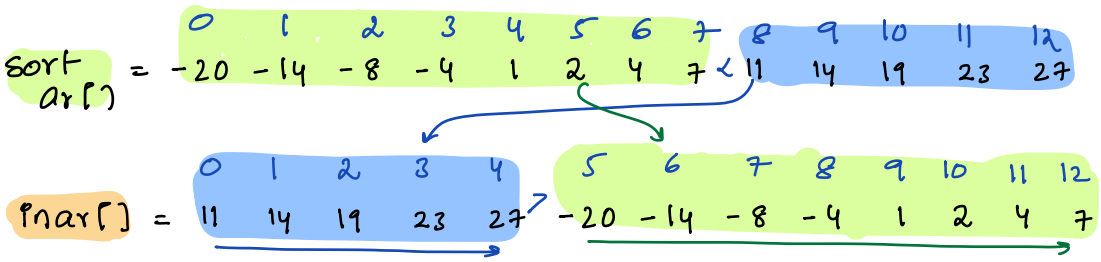


IQ) Given input ar[N] formed by rotating a sorted arr[] k times

In a single rotation, an ele from last is kept at start

Search for target ele in input ar[]

Note: ar[] contains only distinct elements



k=5 [from

// no: of times ar[] sorted

19 : ✓

7 : ✓

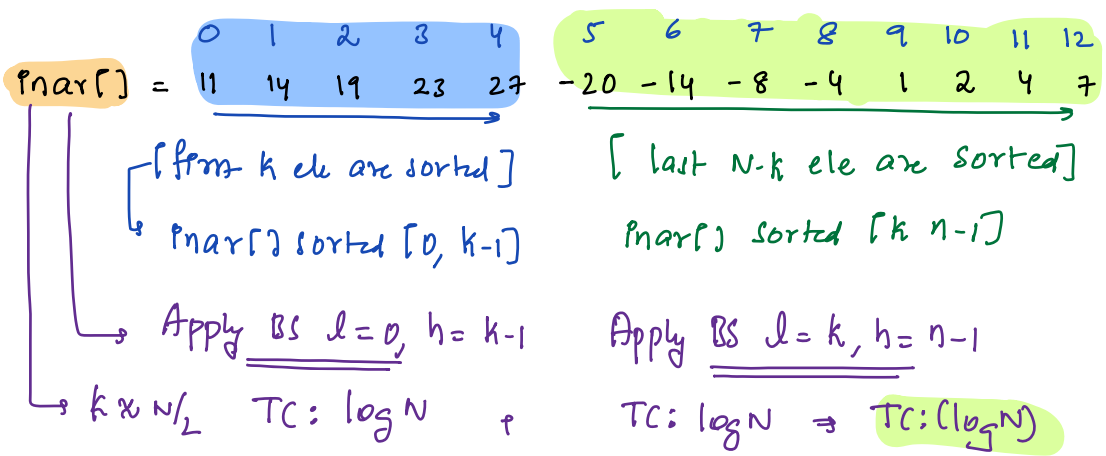
24 : ✗

Idea1: linear search TC: O(N) SC: O(1)

Idea2: Rotate arr[] k times start to end & apply BS

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

Idea2: Apply BS in both sorted arr[]



	0	1	2	3	4	5	6	7	8	9	10	11	12
nar[] =	11	14	19	23	27	-20	-14	-8	-4	1	2	4	7

Obs: Data in blue colour > Data in green colour

↳ ar[0] > Data in green colour

↦ target

↳ Given ele, check if it blue side or green side

```
if( ar[0] > ele ){
    // ele on green side
    apply BS l = k & h = n-1
}
else{
    // ele on blue side
    apply BS l = 0 & h = k-1
}
```

// Applying BS only 1 time

TC: $C \log N$) SC: O(1)

Q) Given ar[] formed by rotating a sorted arr[], search ele (target)

Note: no: of times arr[] is rotated is not given

	0	1	2	3	4	5	6	7	8	9	10	11	12
rnar[] =	11	14	19	23	27	-20	-14	-8	-4	1	2	4	7

Idea1: Linear search

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

Idea2: Find k q using above approach?

: obs: min ele of ar[] is at k^{th} index

↳ // iterate q find min ele index → TC: O(N) SC: O(1)

0	1	2	3	4
11	14	19	23	27

5	6	7	8	9	10	11	12
-20	-14	-8	-4	1	2	4	7

↑ mid

BS Idea:

Case-1: if ar[0] > ar[mid]: mid in greenside

ans = mid

goto left

Case-2: else: mid on red side

goto right

}

```
bool search(int ar[], int target) { TC: O(log N)  SC: O(1)

    l=0  h=n-1, k≠0;   k=N      // to avoid edge cases

    while(l<=h){
        m = (l+h)/2
        if(ar[0] > ar[mid]){
            k = m
            h = m-1  //goto left
        }
        else{ //goto right
            l = m+1
        }
    }

    if(ar[0] > target){
        // element on green side
        BS from [k, n-1]
    }
    else{
        // ele on red side
        BS from [0, k-1]
    }
}
```

finding k

sorar[] = {3 6 8 10 12}

// formed by rotating above array 0 times.

ar[] = {0:3, 1:6, 2:8, 3:10, 4:12}

l	h	m	ar[0] > ar[mid]
0	4	2	3 > 8 goto right
3	4	3	3 > 10 goto right
4	4	4	3 > 12 goto right
5	4	: break, we never updated ans	

ar[] = {0:3, 1:6, 2:8, 3:10, 4:12}

target = 10

if(ar[0] > 10){

    3 > 10 *

}

else{ BS from [0, k-1]

    BS from [0, -1]

}

28) Every element occurs twice except for 1, find unique element

Note: Duplicate are adjacent to each other

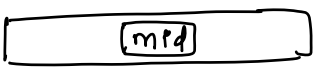
Idea: xor of all elements TC: O(N) SC: O(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
ar[ ]:	3	3	1	1	8	8	10	10	19	6	6	2	2	4	4

if we land on leftside to unique ele
: goto right
: if 1st ocurena is even : on left

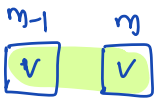
land on right to unique ele
: goto left
1st ourena is odd : on right

Case-I:

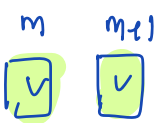



```
if ( ar[mid-1] != ar[mid] && ar[mid] != ar[mid+1] ){
    return ar[mid]
}
```

Can we bring mid to 1st ocurena?

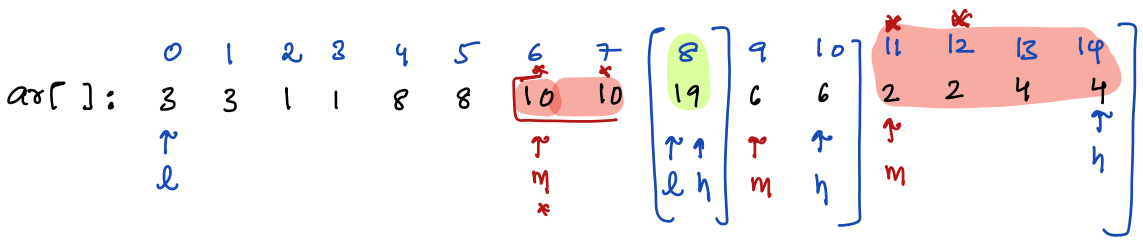



```
if ( ar[m-1] == ar[m] ){
    // 1st occurena = m-1
    m = m-1
}
else // ar[m] == ar[m+1] ){
    // leave it
}
```



// now m is at 1st ourena of ele

```
if ( m % 2 == 0 ){ // 1st ouren even, leftside of unique
    goto right
}
else // m % 2 == 1, 1st occurena odd, right side of unique
    goto left
}
```



l	h	m, m at i^{rr} occurence,	i^{rr} occurence even/odd	
0	14	7	* m = m-1, m = 6	even, goto right	l = m+2
8	14	11	✓ no update	odd, goto left	h = m-1
8	10	9	✓ no update	odd, goto left	h = m-1
8	8	8,	it is ans return ar[8] = 19		

```
int unq(int ar[]){   // n: ar[] = {6} = ans = 6
    if(n==1){return ar[0]}
    if(ar[0] != ar[1]) return ar[0]
    if(ar[n-1] != ar[n-2]) return ar[n-1]
    l = 1   h = n-2
    while(l <= h){          // m=0, ar[-1]==ar[0]
        m = (l+h)/2         //          m=n-1, ar[n-1]==ar[n]
        if(ar[m-1] != ar[m] && ar[m] != ar[m+1]){
            return ar[m+1]
        }
        // bring m to the 1st occurence
        if(ar[m-1]==ar[m]){          // m-1  m
            // 1st occurence = m-1   //  v   v
            m = m-1
        }
        else{ // ar[m]==ar[m+1]){    //  m   m+1
            // leave it              //  v   v
        }
        // using 1st occurence, left or right side
        if(m%2==0){
            l = m+2
        }
        else{ // m%2==1,
            h = m-1
        }
    }
}
```

All of them are edge cases

Q8)

Given +ve N find sqrt(N)

find greatest i such that $i * i <= N$ = sqrt(N)

sqrt(25) = 5

sqrt(16) = 4

sqrt(47) = 6

Eg: N = 40

i	$i*i <= 40$	ans = 6
1	✓	ans = 1
2	✓	ans = 2
3	✓	ans = 3
4	✓	ans = 4
5	✓	ans = 5
6	✓	ans = 6
7	$7*7 <= 40$ ✗	
8	$8*8 <= 40$ ✗	
⋮		

```
int sqrt(int N){
    i=1, ans=1        i<=√N
    while(i*i <= N){
        ans = i
        i = i+1
    }
    return ans
}
```

TC: O(√N) SC: O(1)

Idea2: Search:

a) Target : sqrt(n)

b) Searchspace : lo = 1, hi = N

// ans should 100% in searchspace

c) discard :

Case-I:


```
if(mid*mid == N){
    return mid
}
```

Case-II:


```
if(mid*mid < N){
    ans = mid // goto r
    goto right
}
else if mid*mid > N
    goto left side
}
```

Trace: N = 50

l	h	m	compare m*m & N
1	50	25	25*25 -> 50 : goto leftside h = m-1
			25 26 27 28 ....
			F F F F
1	24	12	12*12 > 50 : goto leftside h = m-1
			12 13 14 15 ...
			F F F F
1	11	6	6*6 < 50 : ans = 6, goto right = l = m+1
			3 4 5 6
			T T T T
7	11	9	9*9 > 50 : goto left h = m-1
7	8	7	7*7 < 50 : ans = 7, goto right l = m+1
8	8	8	8*8 > 50 : goto left h = m-1

8 7 : {break} ans = 7

```
int sqrt(N){   TC: O(log N)  SC: O(1)
    l = 1, h = N, ans = 1
    while(l <= h)
        m = (l+h)/2
        if(m*m == N) {return m}
        if(m*m < N) {ans = m, l = m+1}
        else {h = m-1}
    return ans;
}
```

TC: O(log N) ↳ if we apply BS on N elements it will be log N