Today's Content:

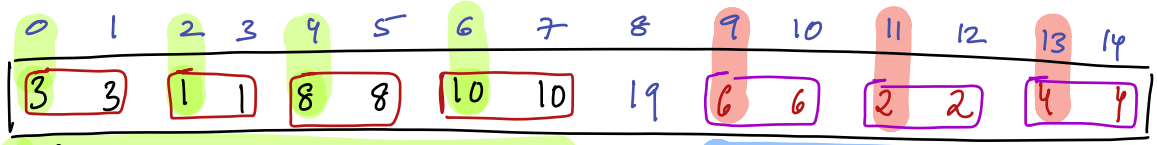
- Search in rotated array [TODO]
- Search in almost sorted
- Sqrt()
- Special Integer

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

Note: Duplicates are adjacent to each other

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

Ex 1:



1st occurs at even index
→ If we land left: go to right

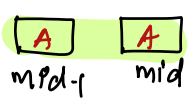
1st occurs at odd index
= If we land in right: go to left

Observation
→ Before unique ele: All 1st occurs are at even index
→ After unique ele: All 1st occurs are at odd index

Idea 2:
- target: Unique Element ?
- search space: Given array

Case 1: ar[mid] is unique element: return ar[mid]

```
if ( ar[mid] == ar[mid-1] {
    mid = mid - 1
}
```



→ mid is 2nd occurs
bring it to first
= mid = mid-1

mid is in first occurs

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

Tracing:

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

ar[m-1] != ar[m] && ar[m] != ar[m+1]

l	h	m	is m unique	if (ar[m] == ar[m-1])	m%2
0	14	7	*	m = m-1; m = 6	0 : left side goto right side { l = m+2 }
8	14	11	*	no change m = 11	1 : right side goto left side { h = m-1 }
8	10	9	*	no change m = 9	1 : right side goto left side { h = m-1 }
8	8	8	✓ { return ar[8] } 19		

// Eg:

0	1	2
3	5	5

l	h	m	un	if (ar[m] == ar[m-1])	m%2
0	2	1	x	no change	1 : right side, goto left h = m-1
0	0	0	✓ { ar[-1] != ar[0] && ar[0] != ar[1] } { RT[-] }		

Pseudo Code :

```
findunique( int ar[], int N){
    l = 0, h = N-1;
    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]}
    while ( l <= h) {
        m = (l+h)/2
        if( ar[m-1] != ar[m]  &&  ar[m] != ar[m+1] ){
            return ar[m]
        }
        if (ar[m] == ar[m-1]){        // m => m 2nd occurrence
            m = m-1;
        }
        if ( m % 2 == 0){ // goto right
            l = m+2
        }
        else{
            h = m-1
        }
    }
}
```

Edge Cases :

If unique at 0 :
ar[-1] != ar[0]

If uniq at n-1;
ar[N-1] != ar[N]

Edge :

If N==1 :
ar[0] == ar[1]
Edge Case

TC: $O(\log \frac{N}{2})$ SC: O(1)

308) Given +Ve N find SQRT(N)

↳ floor(sqrt(N)), (int part)

4.2 → flr : 4

3.09 → flr : 3

sqrt(25) = 5

sqrt(20) = 4

sqrt(10) = 3

Idea:

TC: O(√N) SC: O(1)

```
i=1; ans;
while( i*i <= N){
    ans = i;
    i++;
}
return ans;
```

i*i <= N

i^2 <= N

i <= √N

i: [1 √N]

N = 30

i	i*i <= 30	ans
i=1;	1*1 <= 30	1
i=2,	2*2 <= 30	2
i=3,	3*3 <= 30	3
i=4,	4*4 <= 30	4
i=5,	5*5 <= 30	5
i=6	6*6 <= 30	return ans }

5

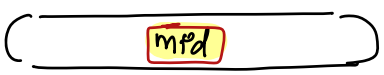
Idea BS

Target : flr(sqrt)

Search Sp : [1 N]

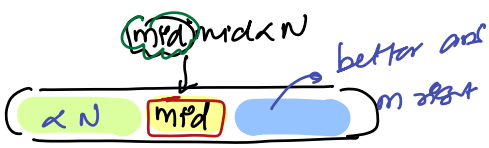
Sea Sp : [1 N/2] : [1 0] ← N=1

fail



Case-I:

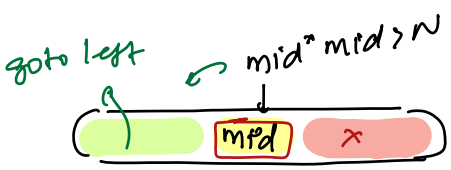
mid*mid == N : return mid



Case-II

mid*mid < N : // goto right

ans = mid l = mid+1



Case-III

mid*mid > N : // goto left

h = mid-1

N = 50

l	h	m	
1	50	25	m*m > 50, goto left, h = m-1
1	24	12	m*m > 50, goto left, h = m-1
1	11	6	6*6 < 50, { ans = m; ans = 6, goto right; l = m+1, }
7	11	9	m*m > 50, goto left, h = m-1
7	8	7	7*7 < 50, { ans = m; ans = 7, goto right; l = m+1, }
8	8	8	m*m > 50, goto left, h = m-1

8 7 : (return ans 7)

10:30

10:40 pm break

10 mins:

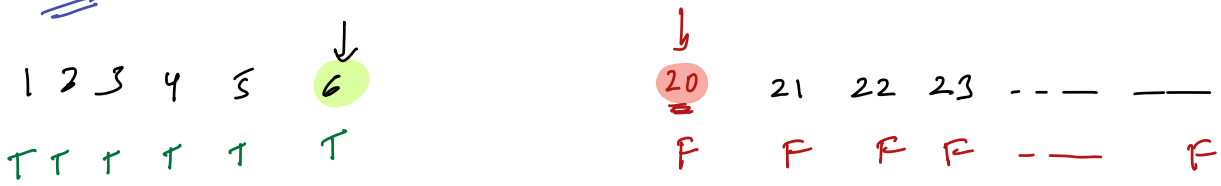
```
int sqrt(N)
    l = 1, h = N
    while(l <= h){
        m = (l+h)/2
        { coding }
    }
    return ans
}
```

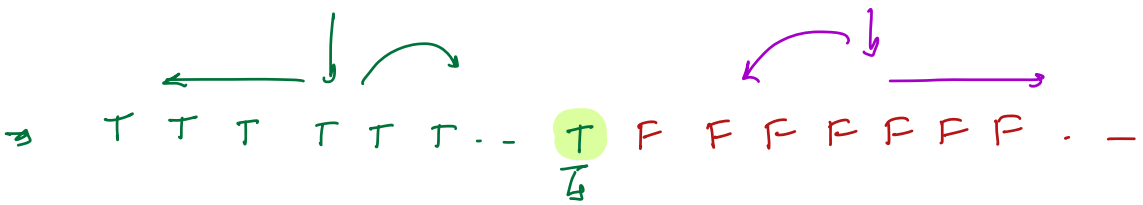
TC: $\log_2^N$

SC: O(1)

//

N = 101

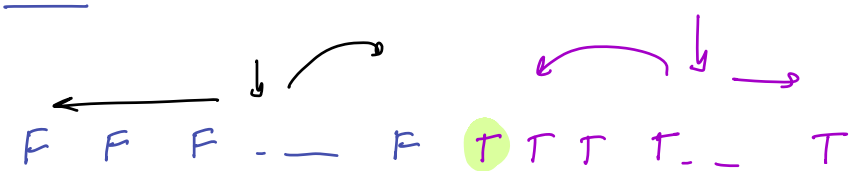




Last True :

obs :

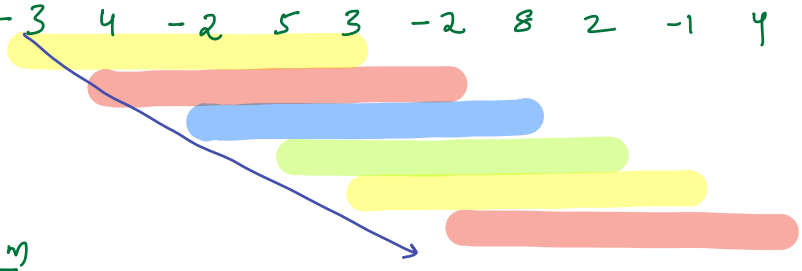
Conditions :



48) Given N array elements

Calculate Max Subarray Sum of len = k

	0	1	2	3	4	5	6	7	8	9
ar[10] =	-3	4	-2	5	3	-2	8	2	-1	4



k = 5 :

s	e	sum
[0	4]	: 7
[1	5]	: 8
[2	6]	: 12
[3	7]	: 16
[4	8]	: 10
[5	9]	: 11

→ max = 16

Sliding window →

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

```
int Subsum(int ar[], int N, int k){
    // should return max subarray sum
    of len = k
    TC: O(N) SC: O(1)
}
```

non-negative Integer: ar[i] >= 0

Q) Given an array of +ve Integers, find max k such that

{ max subarray sum of len k $\leq B$ } ← given as Input

B = 20

	0	1	2	3	4	5	6	7
ar[8] =	3	2	5	4	6	3	7	2

20 20 25 22

Ideas:
→ Sort → * → order gets changed

k	max subarray sum of len = k	ans
1	7 $\leq$ 20	1
2	10 $\leq$ 20	2
3	16 $\leq$ 20	3
4	20 $\leq$ 20	4
5	25 $\not\leq$ 20 *	
6	! *	
	*	

Target: find max (k) who max subarray sum of len k $\leq B$

Search Sp: {0,

ar[20], find max k

obs:

max subar sum of len 5 $\leq B$

max subar sum of len 9 > B

1	2	3	4	5		n	9	10	11	...	
T	T	T	T	T	...	T	F	F	F	F	F

We are looking last True

Ex1:

ar[3] = { 10 14 9 } B = 7

max subarray sum of len = k } ans = 0

k = 1: 14 < 7 *

Ex2:

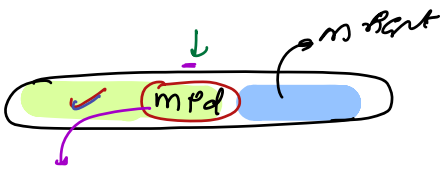
ar[3] : { 3 2 5 } B = 14

max subarray sum of len = k

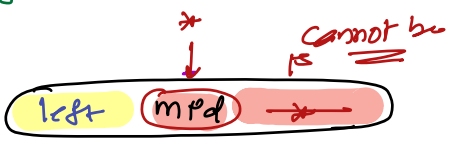
k = 3: 10 < 14 ; ans = 3

1) Given k → (Target)

2) [0, ... N] → (Search Space)




```
if (Subman(ar, N, mid) < B){
    ans = mid;
    l = mid + 1
}
```




```
if ( Subman (ar, N, mid) > B){
    h = mid - 1
}
```

BF Idea:

```
ans = 0
for (i = 1; i <= N; i++) {
    sum = Subman(ar, N, i)
    if (sum < = B){
        ans = i;
    }
    else { break }
}
return ans;
```

TC: N * N → O(N^2)

SC: O(1)

Tracing

	0	1	2	3	4	5	6	7	
ar[8] =	3	2	5	4	6	3	7	2	} B = 10

↳ BS on k → Max subarray len

l	h	m	Max subarr sum of len = m
0	8	4	20 > 10 : { goto left }
0	3	1	7 < 10 : { ans = 1, goto right, l = m+1 }
2	3	2	10 = 10 : { ans = 2, goto right, l = m+1 }
3	3	3	16 < 10 : { goto left }

3 2 {Break} return ans;
↳ 2

Pseudo code :

```
l = 0  h = N
while (l <= h) {
    m = (l+h)/2
    sum = {SubMax(ar, N, m)}      // Max subarray sum of len = m
    if( sum <= B){
        ans = m;  l = m+1
    }
    else{
        h = m-1
    }
}
return ans
```

→ l: 0, h: N } Subarr Max of len = k

→ Sum Binary Search Iteration → $\log_2^N \times N$

→ SC: O(1)