Today's content

- Unique element
- Sqrt()
- Special Integer
- Sorted & Rotated Array (idea)

Q) Every element occurs twice except one, find unique element.

Note → Duplicates are adjacent to each other.

idea-1 → Take XOR of all elements. T.C → O(N), S.C → O(1)

Eg 1: [3 3] [1 1] [8 8] [10 10] 19 [6 6] [2 2] [4 4]

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

1st occurence is at even index (L to R)

1st occurence is at odd index (L to R)

observation: { Before unique element → 1st occurence is present at even idx.
After unique element → 1st occurence is present at odd idx.

idea-2: → target : unique element
→ search space : given array.

Case-1: arr[mid] == unique element : return arr[mid]

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

A A
mid-1 mid

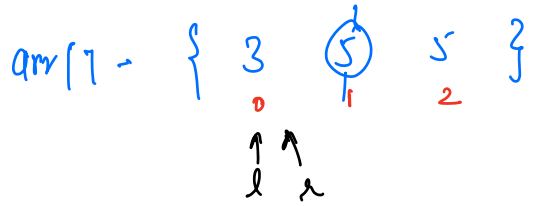
mid is first occurence.

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

Tracing.

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

left	right	m	if m unique	if [arr[m] == arr[m-1]]	
0	14	7	X	m = m-1, m=6	m is even ⇒ go to right. ⇒ left = m+2
8	14	11	X	no change, m=11	m is odd ⇒ go to left. right = m-1
8	10	9	X	no change, m=9	m is odd ⇒ go to left right = m-1
8	8	8	{ return arr[8] } 19		



left	right	m	if m unique	if(arr[m] == arr[m-1])
0	2	1	x	No change, m=1

m is odd
→ go to left.
→ right = m-1;

			edge-case. ★	
0	0	0	if(arr[0] != arr[-1]	&& arr[0] != arr[1])

if(arr[m] != arr[m-1] && arr[m] != arr[m+1])
→ arr[m] is unique element.

Pseudo-code:

```
int findunique ( arr, N ) {
    left = 0 , right = N-1
    Edge cases:
    if (N == 1) { return arr[0] }
    if (arr[0] != arr[1]) { return arr[0] }
    if (arr[N-2] != arr[N-1]) { return arr[N-1] }
    while ( left <= right) {
        int m = (left + right) / 2 ;
        if ( arr[m-1] != arr[m] && arr[m] != arr[m+1]) {
            return arr[m];
        }
        if ( arr[m] == arr[m-1]) {
            m = m-1      //m - first occurence.
        }
        if ( m % 2 == 0) {
            left = m+2
        }
        else {
            right = m - 1 ;
        }
    }
    return -1
```

T.C → O(log₂N)

S.C → O(1)

Q) Given +ve N. Find sqrt(N).

↳ floor(sqrt(N)) → integer part

sqrt(25) = 5

sqrt(20) = 4

sqrt(10) = 3

idea:

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

{T.C → O(√N), S.C → O(1)}

N = 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-2.

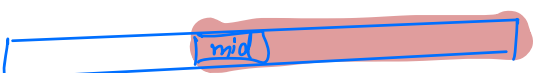
Binary Search
- target - floor(sqrt(N))
- search space - [1 N]
- Search space - [1 N/2] (N=1)

Case1: mid * mid == n
(return mid)

Case2: mid * mid < N

ans = mid

//go to right → left = mid + 1



Case3.: mid * mid > N

//go to left

right = mid - 1

Tracing (N = 50)

left	right	mid	
1	50	25	m*m > 50, go to left, right = mid-1
1	24	12	m*m > 50, go to left, right = mid-1
1	11	6	m*m < 50 ans = 6, go to right. left = mid+1
7	11	9	m*m > 50, go to left, right = mid-1
7	8	7	m*m < 50, ans = 7, go to right left = mid+1
8	8	8	m*m > 50, go to left, right = mid-1
8	7		→ break from loop & return ans.

pseudo-code:

```
int sqrt(N){
    left = 1, right = N, ans =
    while(left <= right){
        mid = (left + right)/2;
        if(mid * mid == n){ return mid }
        else if(mid * mid < n){ ans = mid
                                left = mid+1 }
        else { right = mid - 1 }
    }
    return ans;
}
```

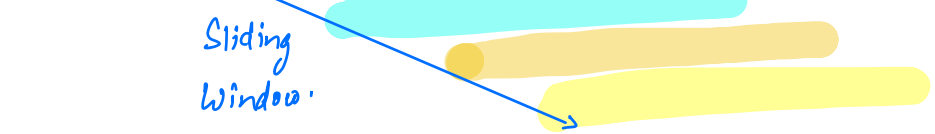
T.C → O(log₂N)

S.C → O(1)

Q: Given N array elements,
calculate max subarray sum of len = K.

Ex:

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



s	e	sum
0	4	8
1	5	17
2	6	20
3	7	19
4	8	15
5	9	13

ans = 20

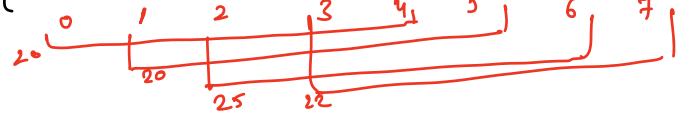
```
int maxSubarraySum(arr, N, K) {
// it should return max subarray
   sum of len = K.
}
```

T.C → O(N)
S.C → O(1)

Q. Given an array of N +ve integers. Find max K such, that { max subarray sum of len K <= B }.
(input)

arr[8]: [3 2 5 4 6 3 7 2]

B = 20



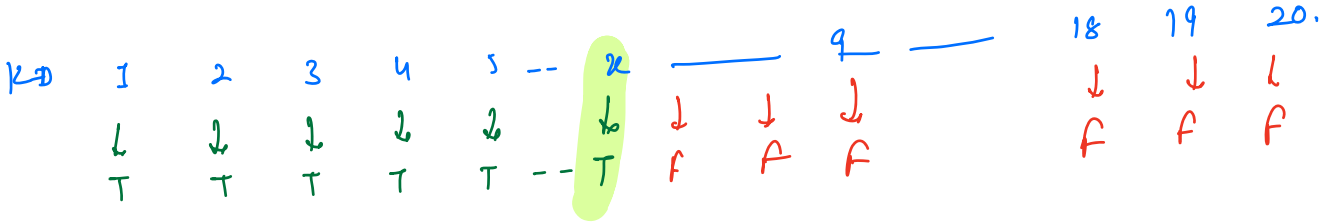
K.	max Subarray sum of len = K	ans.
1.	7 <= 20	1
2	10 <= 20	2
3	16 <= 20	3
4	20 <= 20	4
5.	25 <= 20 x	
6	x	
7.	x	

idea-1. → for every value of K,
find maximum subarray sum of len = K & check if sum <= B or not.

T.C → O(N^2) , S.C → O(1)

Observation:

arr[20], find max value of K satisfying that condition.



{max subarray sum of len x $\leq B$}

if this condition is true for any value of x

⇒ It will be 100% true for $K \to [1, x]$.

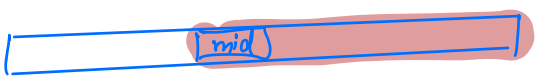
→ We are looking for last true condition.

Idea for B.S.:

forget → max value of K s.t., {max subarray sum of len K $\leq B$}

search space → [1 — — N]


```
if ( maxSubarraySum(arr, N, mid) <= B ){
    ans = mid
    left = mid + 1
}
```




```
if ( maxSubarraySum(arr, N, mid) > B ){
    // go to left.
    right = mid - 1
}
```

Tracing: [idea → B.S on value of K]

arr[8]: [3 2 5 4 6 3 7 2] , B = 15
0 1 2 3 4 5 6 7

left	right	mid	max Subarray Sum of len K
1	8	4	20 ≤ 15 , {go to left} ⇒ right = mid - 1
1	3	2	10 ≤ 15 , go to right ⇒ left = mid + 1 ans = 2.
3	3	3	16 ≤ 15 , {go to left} ⇒ right = mid - 1
3	2		∵ (left > right) , break.

pseudo-code:

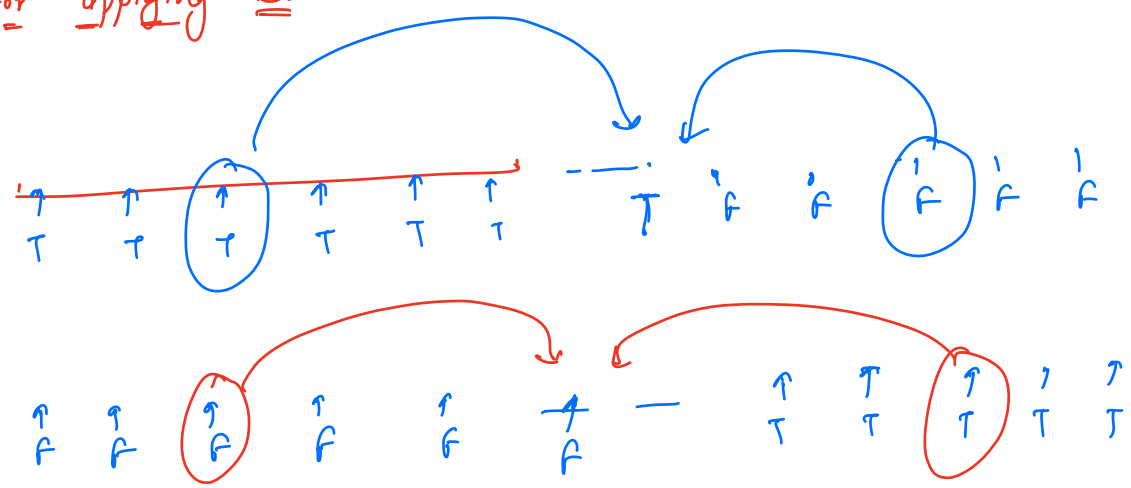
```
left = 1 , right = N
while ( left <= right) {
        mid = (left + right) / 2;
        sum = maxSubarraySum( arr, N, mid); → O(N)
        if (sum <= B) {
            ans = mid
            left = mid+1
        }
        else {
            right = mid - 1
        }
}
return ans;
```

T.C → O(NlogN)

S.C → O(1)

---

idea for applying B.S.



Sorted & Rotated Array.

→ K times clockwise rotation.

arr → [2 3 5 9 11 20 30].

⇓ K = 4.

arr → [9 11 20 30 2 3 5]

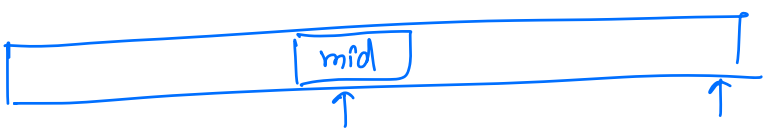
0 1 2 3 4 5 6

left ↑ , right ↑

Apply B.S. ? target → K ⇒ { smallest / largest }

Search space → Entire array.

(smallest element)



arr[mid] < arr[right]

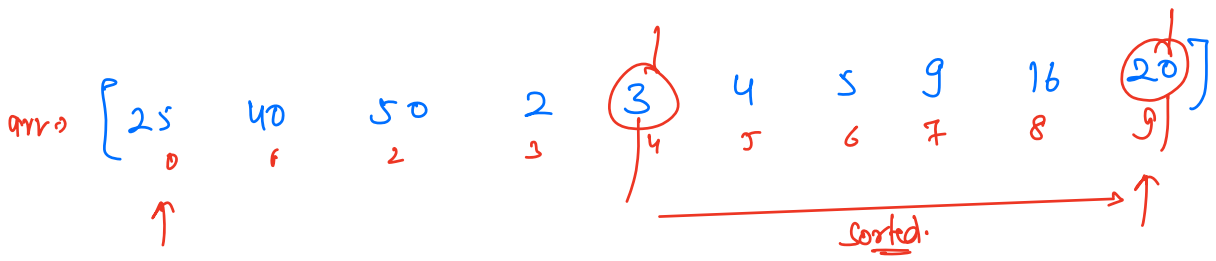
move on left.

{ right = mid }

arr[mid] > arr[right].

→ go to right.

→ left = mid+1.


arr =
25
40
50
2
3
4
5
9
16
20
0
1
2
3
4
5
6
7
8
9
sorted.