Todays Content:

- → Unique element
- → sqrt()
- → special integer
- → A^{th} magical number

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

Note: Duplicates are adjacent to each other

Idea1: XOR of all elements TC: O(N) SC: O(1)

Eg1:

ar[] =

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

Before unique ele: goto right

1st occurs are in even idx

After unq ele : goto left

1st ocurcna are in odd idx

Case1: mid

if ar[mid] is unq ele

if (ar[mid-1] != ar[mid] && ar[mid] != ar[mid+1]

Making mid land in 1st ocurcn:

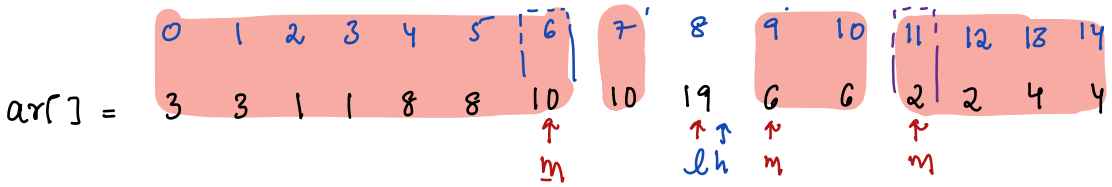
```
if (ar[mid-1] == ar[mid]) {        mid-1 == mid
    mid = mid-1 // Now mid points to 1st occurrence
}
else {  mid == mid+1
    // mid is already at 1st ocurrence
}
```

Case 1:

```
if ( mid % 2 == 0) {
    // We are left side of unique element
    goto right
}
```

Case-2:

```
if ( mid % 2 == 1) {
    // We are right side of unique element
    goto left
}
```



l	h	m	isunq	1st our	
0	14	7	*	m = m-1 = 6	m%2==0 goto right	l = m+2
8	14	11	*	✓ m=11	m%2=1 goto left	h = m-1
8	10	9	*	✓ m=9	m%2=1 goto left	h = m-1
8	8	8	✓	return ar[m]		

```
int uni(int ar[N]){
    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]      // Edge Cases
    l = 1, h = n-2
    while(l <= h){
        m = (l+h)/2
        if(ar[m-1] != ar[m] && ar[m] != ar[m+1])   // if m==0 fails; if m==n-1
            return ar[m]
        if(ar[m-1] == ar[m]){ m = m-1 }            // Bring m to 1st ocurnce
        else{ // ar[m] == ar[m+1]
            do nothing
        if(m%2==0){ // 1st ocurnce even goto right
            l = m+2
        else{ // 1st ocurnce odd goto left
            h = m-1
    }
}
```

Given +ve find SQRT(N)

Find greatest i such that $i * i <= N$

sqrt(25) : 5

sqrt(30) : 5

Idea: N = 30 ... ---> ans = 1, i = 1

i	i*i <= 30	ans
1	Y	1
2	Y	2
3	Y	3
4	Yes	4
5	Y	5
6	No	* break

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

TC: O(√N) SC: O(1)

Idea2: BinarySearch:
- Target: sqrt()
- Searchspace: 1 .... N

Any N, sqrt will always be in range of 1 ... N

Can1:

mid*mid == N return mid

Can2:

mid*mid <= N

ans = mid

goto right

Can3:

mid*mid > N

goto left

Tracing :

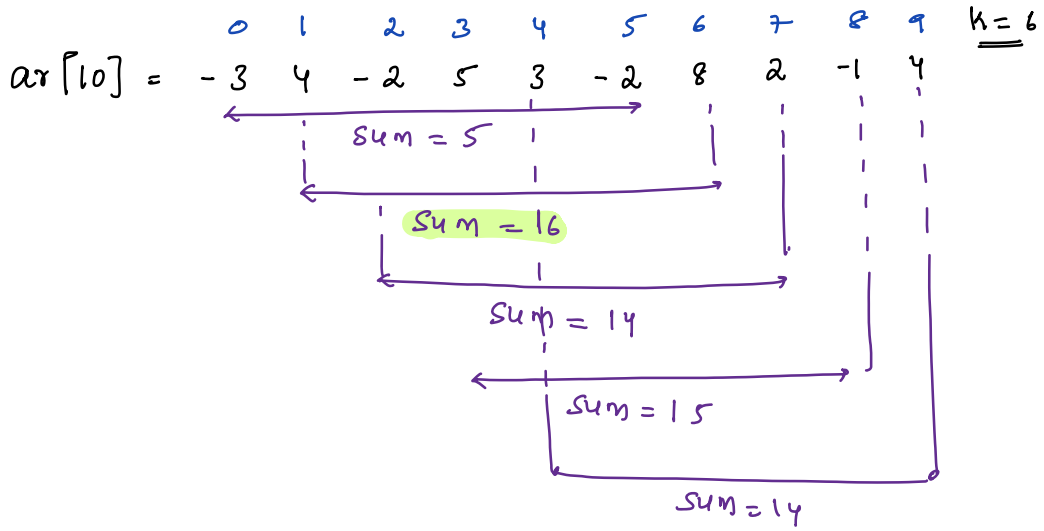
N = 100, ans = 1

l	h	m	update
1	100	50	50*50 > 100, goto left h = m-1
1	49	25	25*25 > 100, goto left h = m-1
1	24	12	12*12 > 100, goto left h = m-1
1	11	6	6*6 <= 100, ans = 6, goto right l = m+1
7	11	9	9*9 <= 100, ans = 9, goto right l = m+1
10	11	10	10*10 <= 100 ans = 10, goto right l = m+1
11	11	11	11*11 > 100 goto left h = m-1
11	10	{ break }	

Code TODO :

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

3Q) Given ar[N] elements, Calculate max subarray sum of len = k




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

4Q) Given ar[N] of +ve Integers > 0 (input)

find max k such that, {max subarray sum of len k} ≤ B

	0	1	2	3	4	5	6	7	
ar[8] =	3	2	5	4	6	3	7	2	B = 20

k	max subarray sum of len = k	
1 :	7 ≤ 20	ans = 1
2 :	10 ≤ 20	ans = 2
3 :	16 ≤ 20	ans = 3
4 :	20 ≤ 20	ans = 4
5 :	25 > 20	*
6 :	[max subarr sum of len(5) > 20 *] (1, ele > 0)	

ar[5] = { 10 2 12 8 7 } B = 25 k =

k	max subarray sum of len = k	ans
		1
1	12 ≤ 25	
2	20 ≤ 25	2
3	27 > 25	*

Idea: For all subarrays of len = k [TC: O(N*N) = O(N²), SC: O(1)]

ans = 0 ? ——→ Edgecase:

ar[3] = {20 18 25}
B = 15

max sub sum of len = 1

k = 1 ; 25 > 15 *,

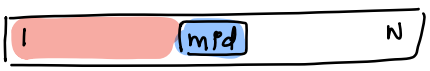
ans = 0

```
k=1; k<=n; k++){
    if( maxSubSum(ar[], k) <= B){
        ans = k
    }else{ break;}
}
return ans
```

Idea2: NlogN → Sorting + BinarySearch

Target: ✓

Searchspace: [1 – N] ✓



Case-I: max subarray sum of len = mid <= B

ans = mid, goto right

Case-II:

max subarray sum of len = mid > B

goto left

	0	1	2	3	4	5	6	7	
ar[8] =	3	2	5	4	6	3	7	2	B = 16

l	h	m	man subarray sum of len m	ans = 0
1	8	4	20 > 16	✗ goto left h = m-1
1	3	2	10 < 16	✓ ans = 2, goto right l = m+1
3	3	3	16 = 16	✓ ans = 3, goto right l = m+1
4	3		[Break]	return ans = 3

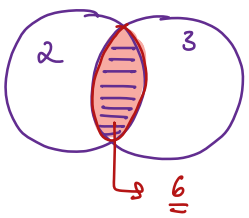
```
int specialInteger(int ar[], int N, int B){
    ans = 0
    l = 1, h = N
    while (l <= h){                    TC: O(logN * N) ≈ O(NlogN)
        m = (l+h)/2
        if ( manSubSum(ar, m) >= B){
            ans = m, l = m+1
        }
        else{
            h = m-1
        }
    }
    return ans;
}
```

48) A^{th} magical number :

Pre-requisits :

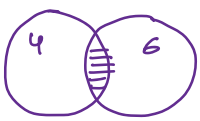
→ number of mul of 3 from 1 to 100 = $100/3 = 33$

→ number of mul of 2 or 3 from 1 to 100 = $50 + 33 - 16 = 67$



→ mul of 4 or 6 from 1 to 100 = $\frac{100}{4} + \frac{100}{6} - \frac{100}{12} = 33$

$12 = lcm(4, 6)$



4 : 4 8 12 16 20 24 28 32 36 . . . .

6 : 6 12 18 24 30 36 42 . . . . .

mul of A or B from 1 – C = $C/A + C/B - C/lcm(a, b)$

4Q) Given A, B, C find C^{th} magical number

magical number is divisibly by A or B or both

A = 2, B = 3, C = 8 : ans = 12 → 1 : 16

1	2	3	4	5	6	7	8	9	10	11	12
*	✓	✓	✓	*	✓	*	✓	✓	✓	*	✓
	1	2	3		4		5	6	7		8

A = 4, B = 6 C = 6 : ans = 18 → 1 : 24

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
*	*	*	✓	*	✓	*	✓	*	*	*	✓	*	*	*	✓	*	✓

→ [1 . . . . C^{*} min(A, B)] ⇒ ans have to present in range

with in this range we should get all C magical numbers?

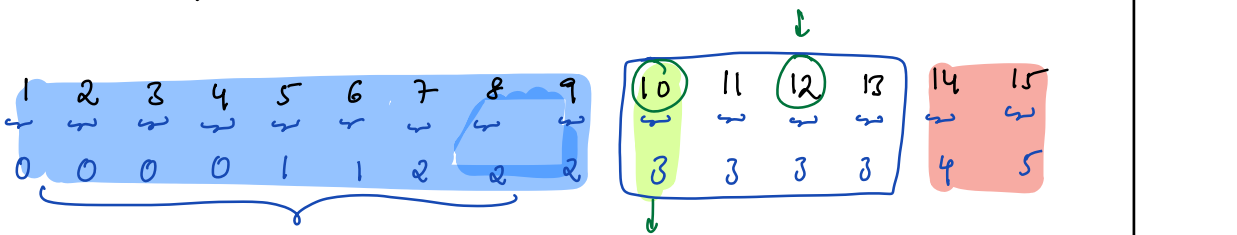
no: of mul of A ∩ B from [1 mid] < C : goto right

no: of mul of A ∩ B from [1 mid] > C : goto left

no: of mul of A ∩ B from [1 mid] == C : ans = mid
goto left

Eg:

$a = 5, b = 7, c = 3$



magical number < c

goto right

mth occurence of number with magical number = 3

ans = m; goto left

magical number > c

goto left

```
int mthMagical(A, B, c) {   TC: log2(max(A,B)*c)   SC: O(1)

    l = 1, h = min(A,B)*c, ans = 0        [1 ... min(A*B)c]

    while(l <= h) {

        m = (l+h)/2

        // number of mul of A n B from [1-m]

        int num = m/A + m/B - m/lcm(A,B)

                                  lcm: a*b / gcd(a,b)

        if(num == c) {
            ans = m;
            h = m-1
        }
        else if(num > c) {
            h = m-1
        }
        else { // num < c
            l = m+1
        }
    }

    return ans;
}
```

N → $\log N$

x → $\log_2 x$