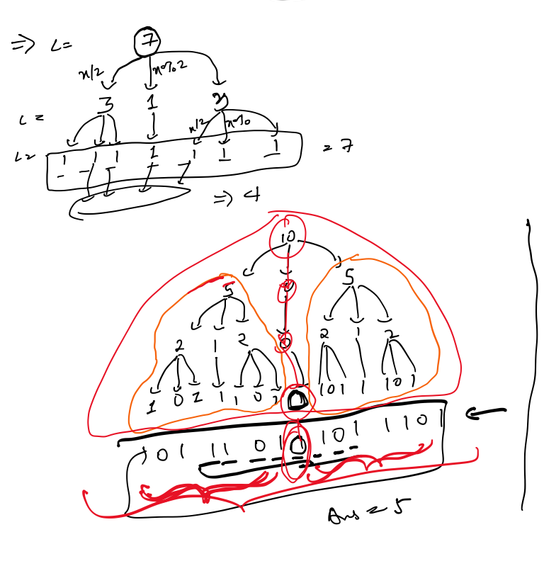
Problem solving 1

Tuesday, 16 February 2021 3:51 PM

N=7 L=2, R=5



⇒ L=
7
N/2
N%2
3
1
3
L=
L=
1 1 1 1 1 1 1
= 7
⇒ 4
10
5
5
2 1 2
2 1 2
1 0 1 1 0 1
101 11 0 1 0 101 110
Ans = 5

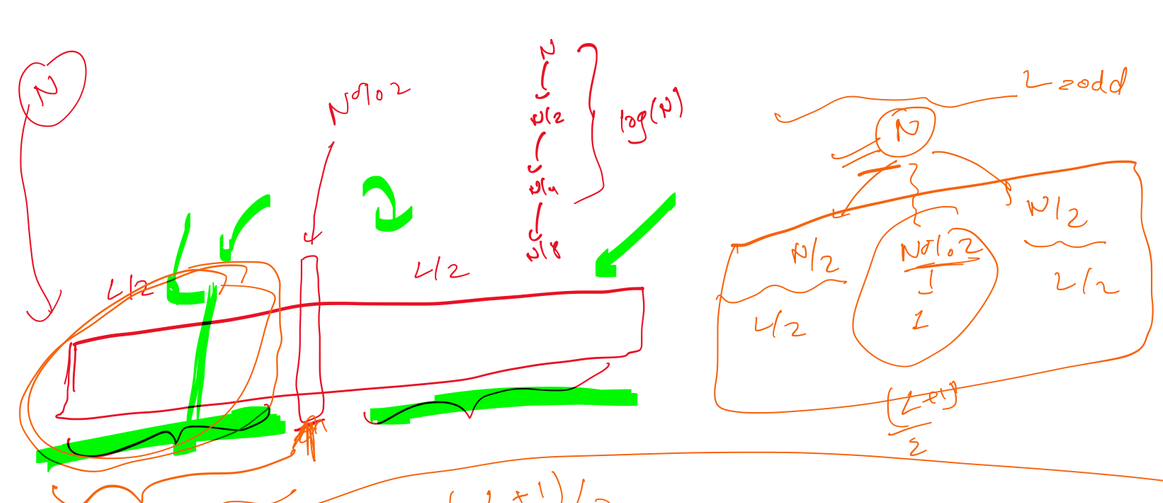
$f(N) \to f(N/2) + 1 + f(N/2)$ Time complexity = (N)

$f(N) \to 2f(N/2) + 1$

if (N==0 or N==1) return 1;

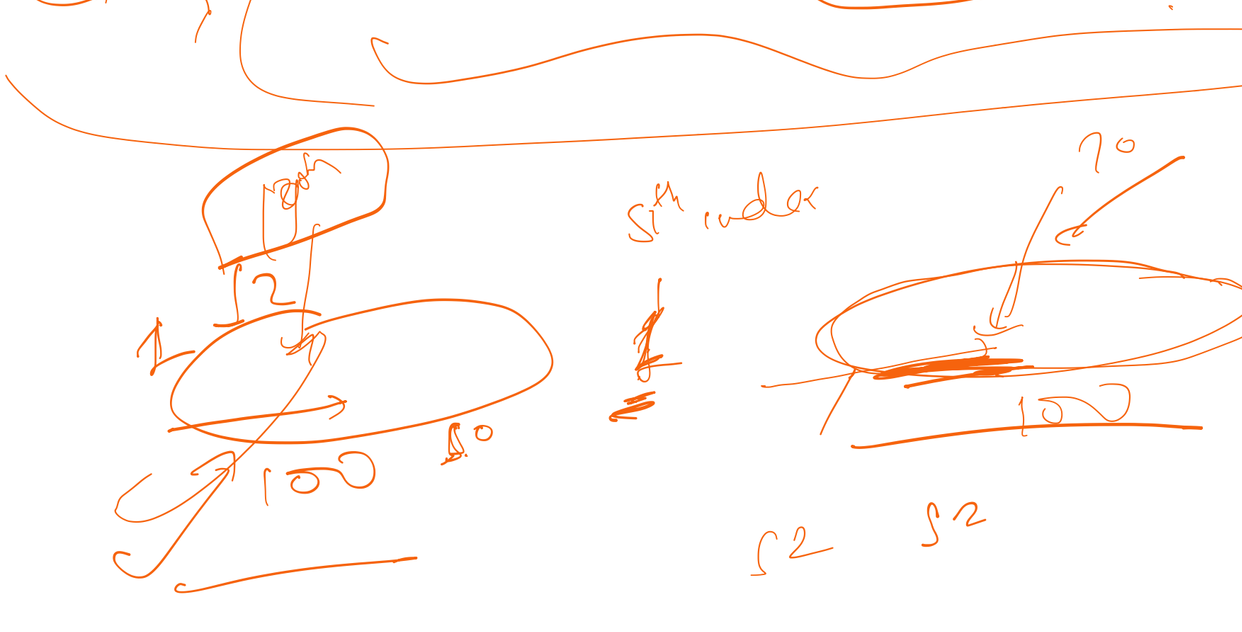
1 - - - - - - - - - - - - - - - - 2en^th

l ← 105 → r

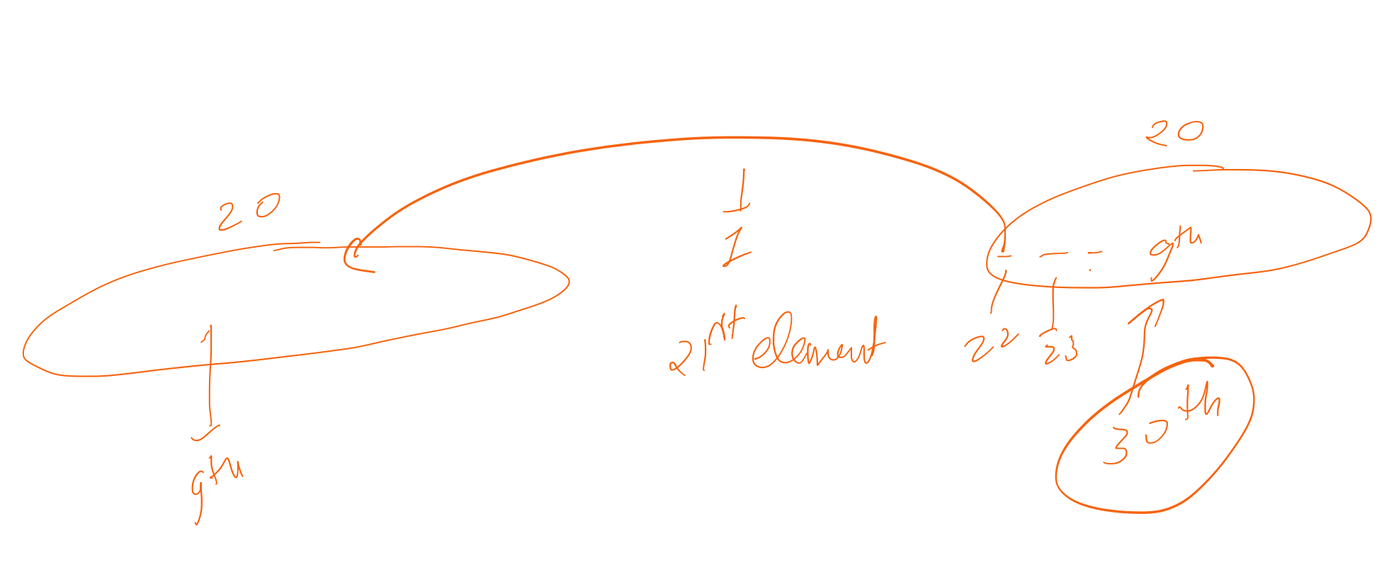


N
N%2
N
N/2
N/4
N/8
log(N)
L/2
L/2
L=odd
N
N/2
N%2
1
1
N/2
L/2
L/2
(L+1)/2

```
(L+1)/2
find(index, N, L){
  if (index ==(L/2 +1) )          => return N%2;
  else if (index < (L/2+1) ){
      return find(index, N/2, L/2);
  else if (index > (L/2+1)){
      return find( index % (L/2+1), N/2, L/2) )
  }
```



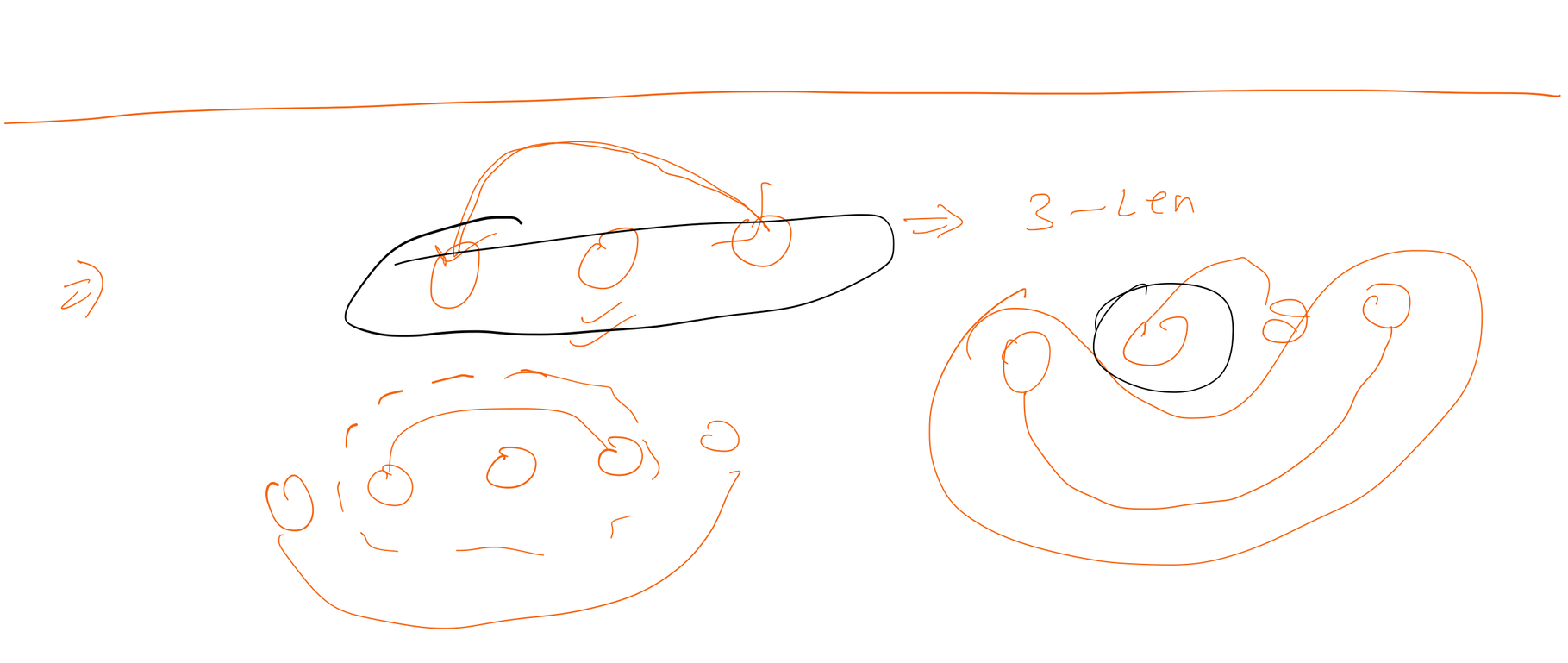
100
2
1
100
50
5th index
70
102
52
52



20
1
21st element
20
22 23
9th
9th
30th

= 22
1
2 ← 23

30%21 (L/2+1)

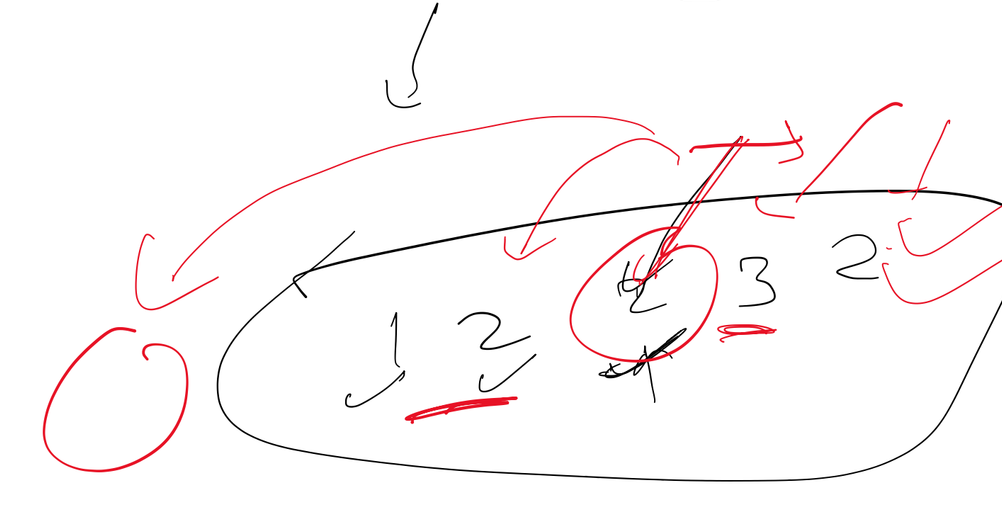


freq [1] = 2
[2] = 2

Iterate

1 2 2 3 2

freq [1] = 1
[2] = ~~2~~ 3
[3] = 1



0 - 1 ⇒ ✓ → prev element should be diff.

1 → 2 ⇒

2 → 3 ⇒ True

__gcd (x,y)

x y

x * y = LCM × GCD

LCM = (x * y) / GCD

x, y = x%y, y

```
gcd (x, y) {
   if(x==0 || y==0)
        return x+y;
} else return (y%x, x);
```

{ map, set, vector, struct, pair, tuple, arrays }

⇒ insertion ~ log(N)

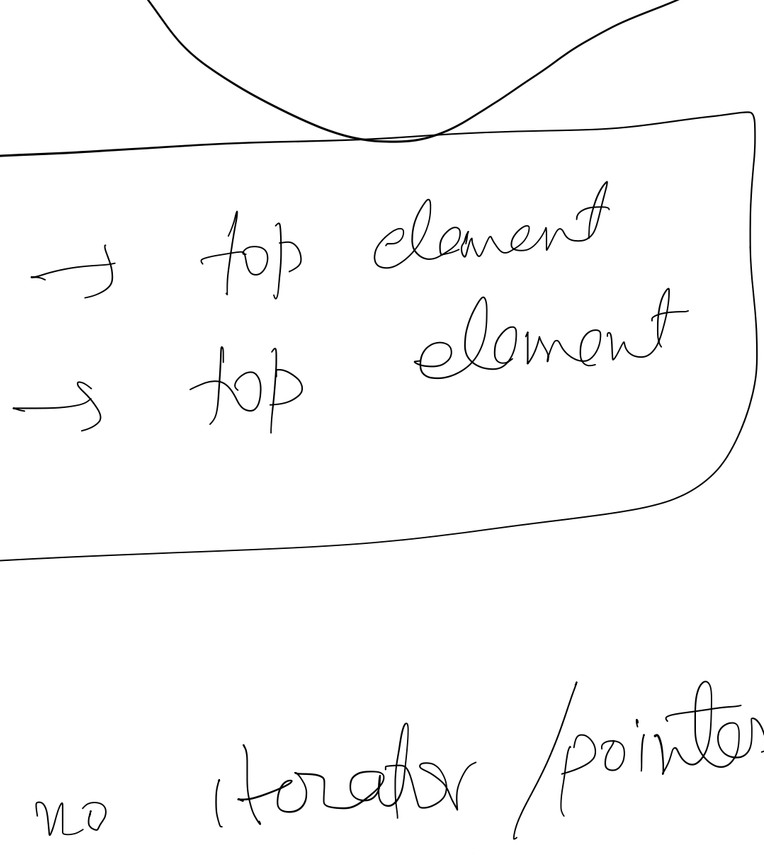
~~unordered~~ unordered_map < string, int > O(1)

---

priority - queue ✓ set

s.begin()

values log(N)



peak value
maximum element
PQ / Heap

* you can only access top element

- insertion → log(N) → top element
- Deletion → log(N) → top element
- access → O(1) → top element

→ low constant factor (

→ Easy to handle (as no iterator / pointers are there in PQ)

MST → PQ

Dijkstra → PQ

2xN ⇒ O(N)

⇒ 5xN ⇒ O(N)

PQ

```
PQ. push (elem)
PQ. top()
PQ. pop()
PQ. size()
PQ. empty()
```

5

300	200	400	500	911
1	2	1	2	3 → a
2	1	3	2	1 → b
0	1	2	3	4

v[a[i]]. push_back(i)

v[b[i]]. push_back(i)

6

2 3 1 2 1 1

type

1 → 0 1 2 4

2 → 0 1 3 3

3 → 2 4

PQ

map < int, bool >

m[price] = true

(n) → (n) —

2 1 7 4 2

K min = 3

⇒ ~~7 4~~ 2 2 1 ~~3~~ 2

7 + 4 + 3

- (A) N log(K)
- (B) N log(N)
- (C) K log(N)
- (D) ~~N~~ log(K)

PQ. top() → O(1)

PQ. pop() + log(N)

N log(N) + (2 log(N)) × K

(N+K) log(N)

K = 10^9

N element ↓ N element

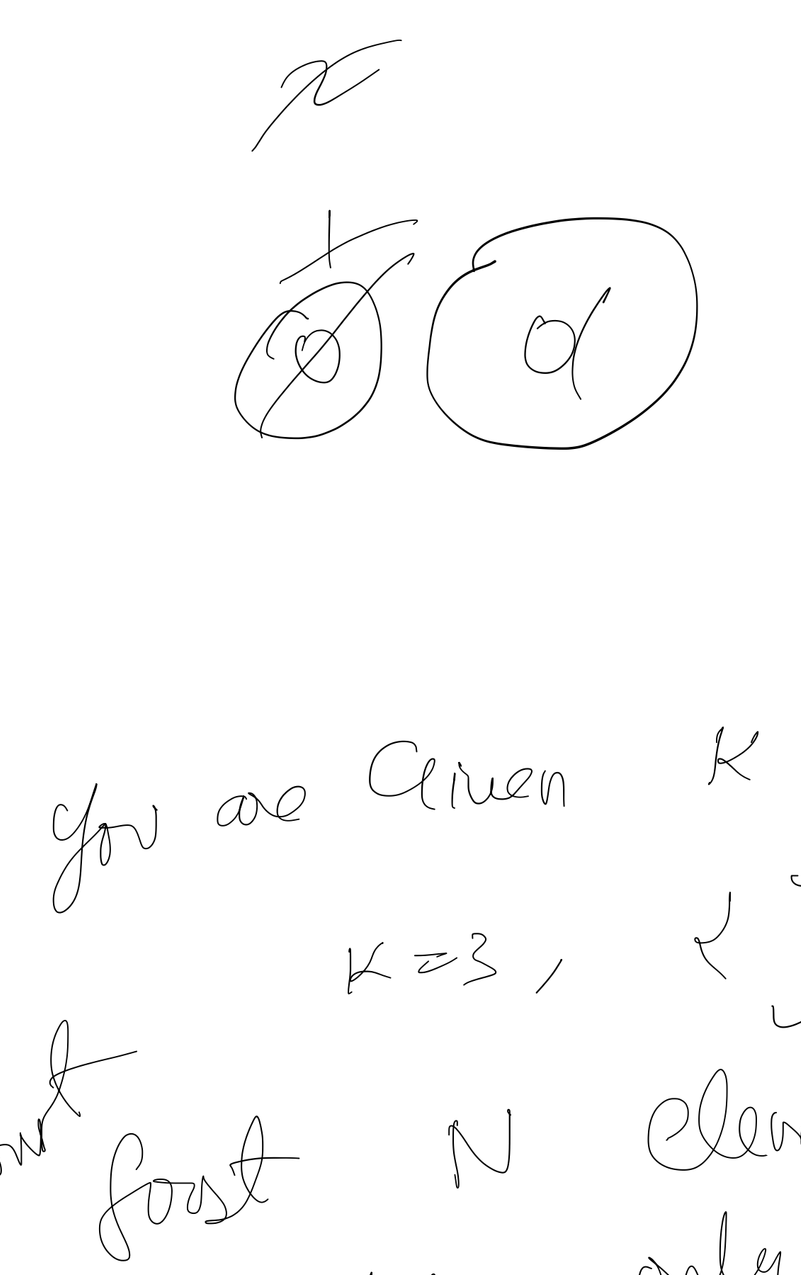
n
10
log(N) + log(N)
n/2
⇒ 2 log(N)
n/4
⇒ 2 log(N)
Total number

(log(N)) × N × log(N)

N ⇓ K × log(N)

K =

K = 1000



20 + 10 + 5 + 1 + 0 + 0

⇒ You are given K primes (1 ≤ K ≤ 5)

K = 3, { 2, 3, 5 }

(N) → print first N elements ~~[illegible]~~ which contain only these primes in their prime factorization

2, 3, 4, 5, 6, 8, 9, 10 - - - -

2 3 2×2 5 2×3 2×2×2