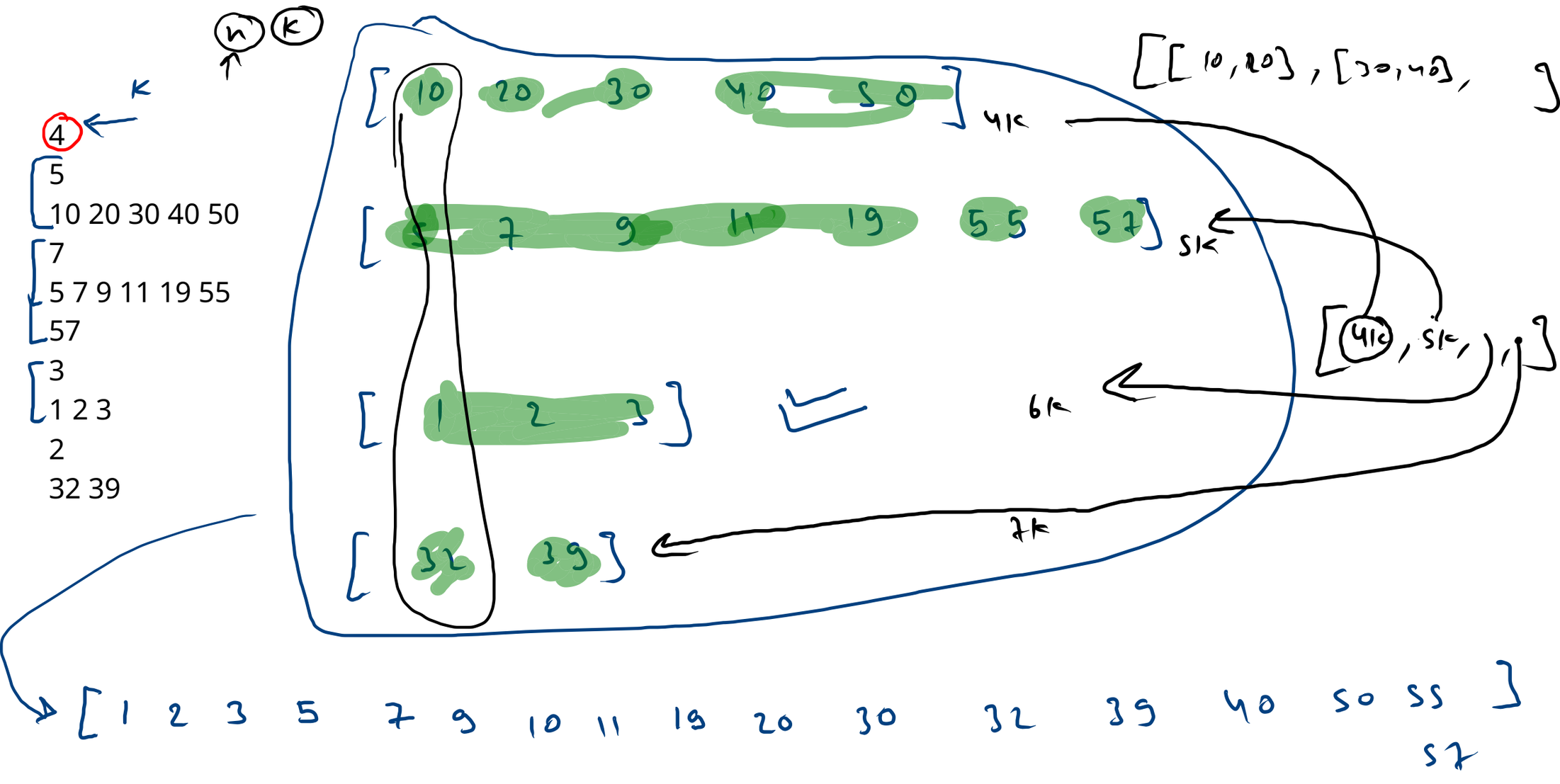
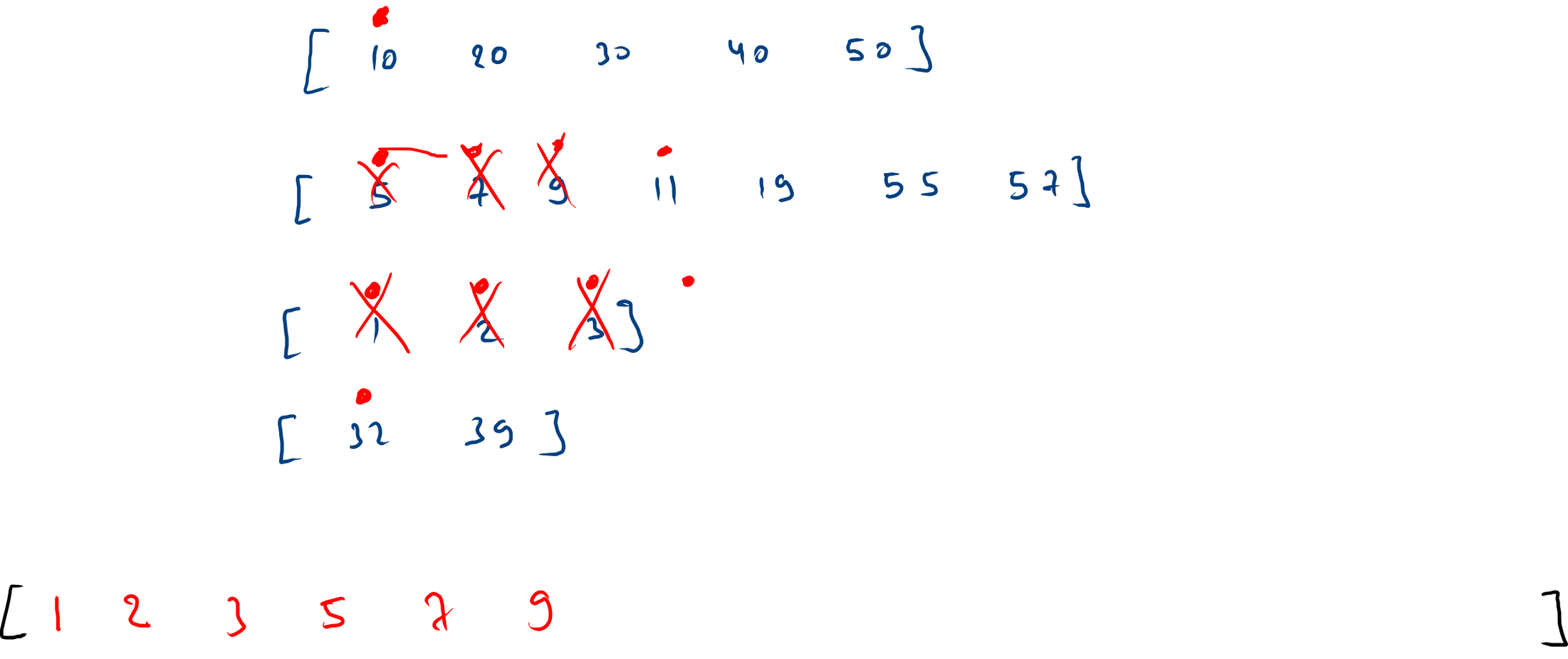

n
k
4
5
10 20 30 40 50
7
5 7 9 11 19 55
57
3
1 2 3
2
32 39
[[10,20],[30,40],
41k
5k
6k
7k
[1 2 3 5 7 9 10 11 19 20 30 32 39 40 50 55 57]


[10 20 30 40 50]
[5 4 9 11 19 55 57]
[1 2 3]
[32 39]
[1 2 3 5 7 9]

k → 4

[10 20 30 40 50]

k

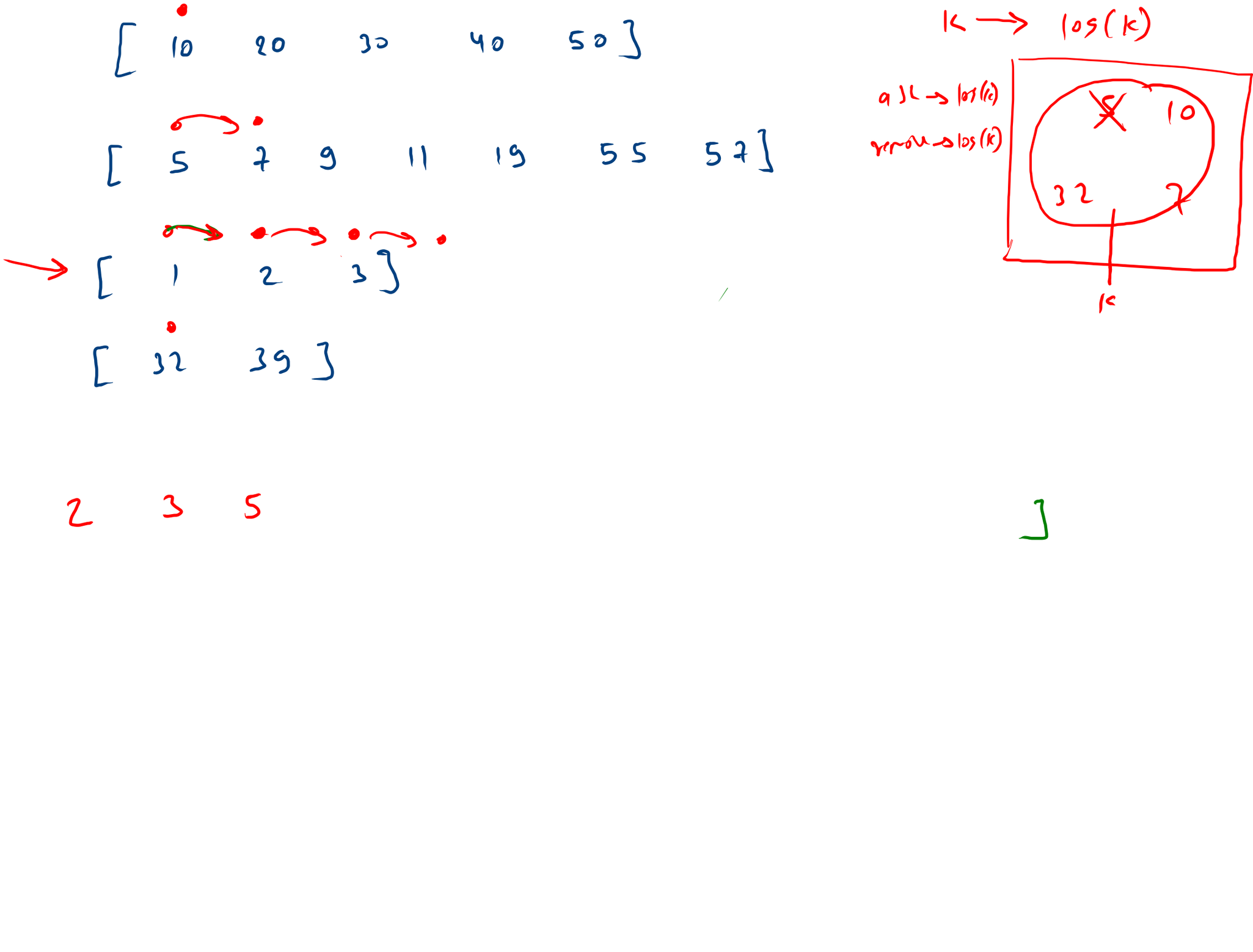
[5 7 9 11 19 55 57]

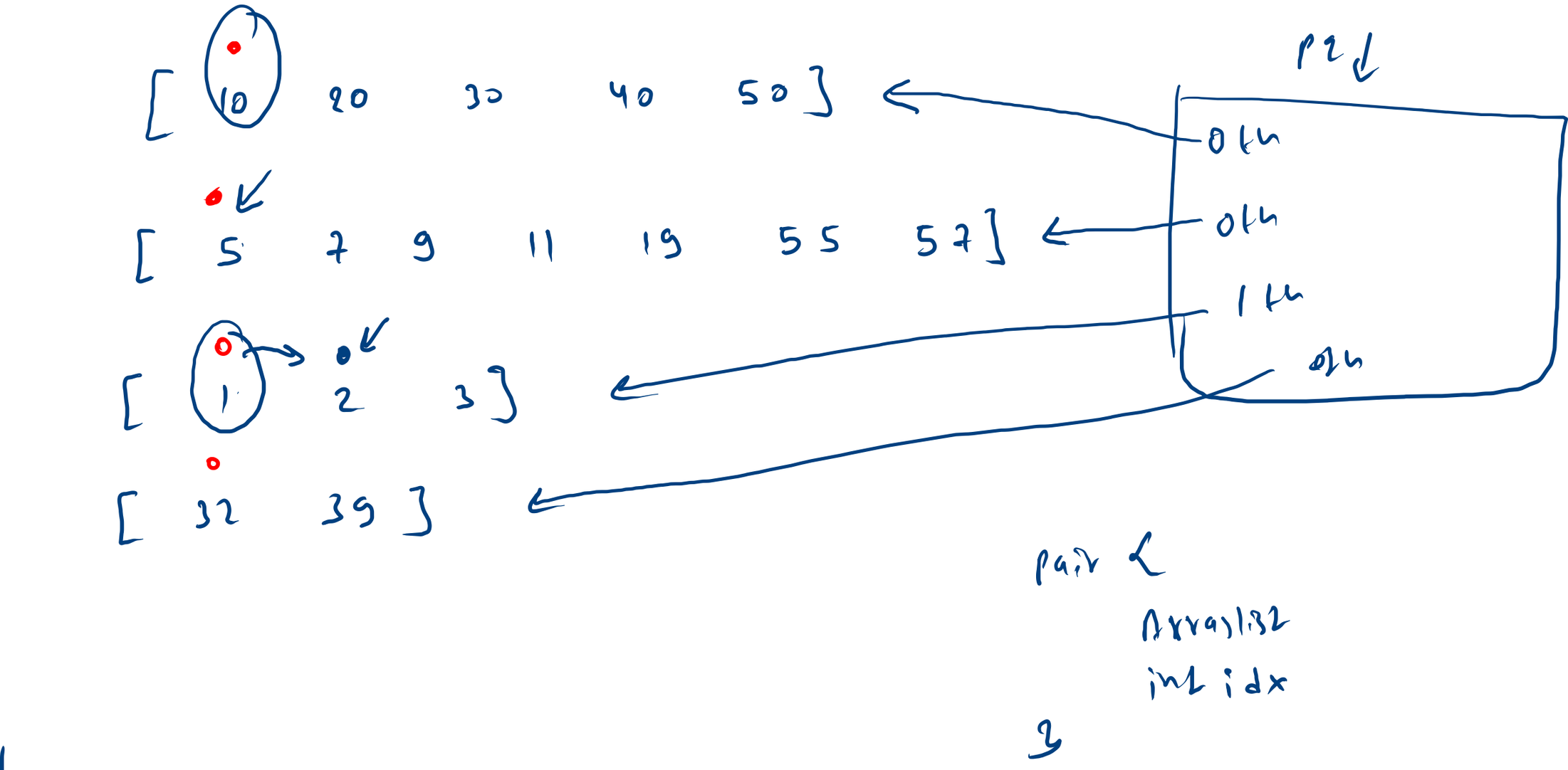
O(n.k)

[1 2 3]

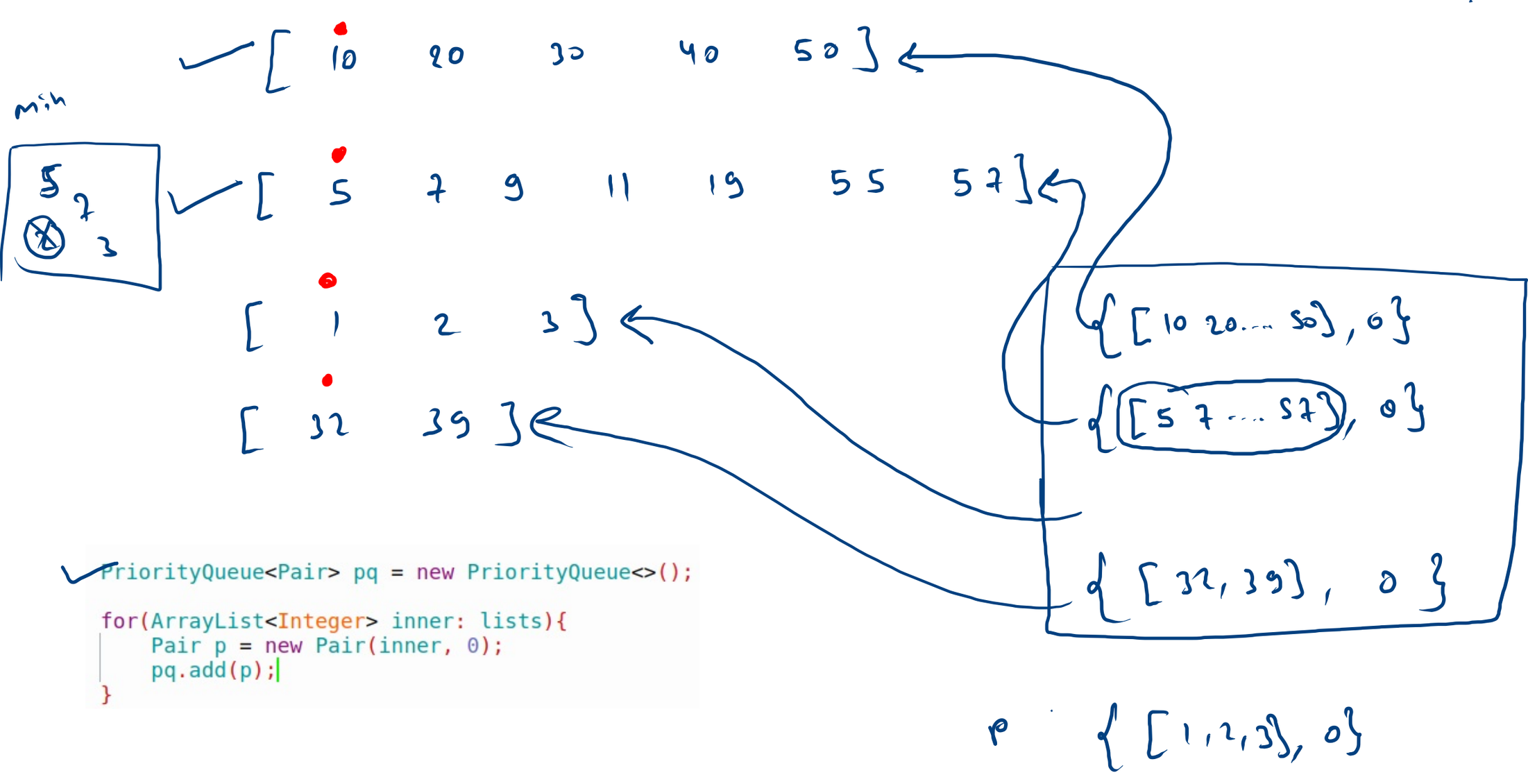
[32 39]

[1 2 3]


10 20 30 40 50
5 7 9 11 19 55 57
1 2 3
32 39
1 2 3 5
k → log(k)
avl → log(k)
normal → log(k)
5
10
32
7
k

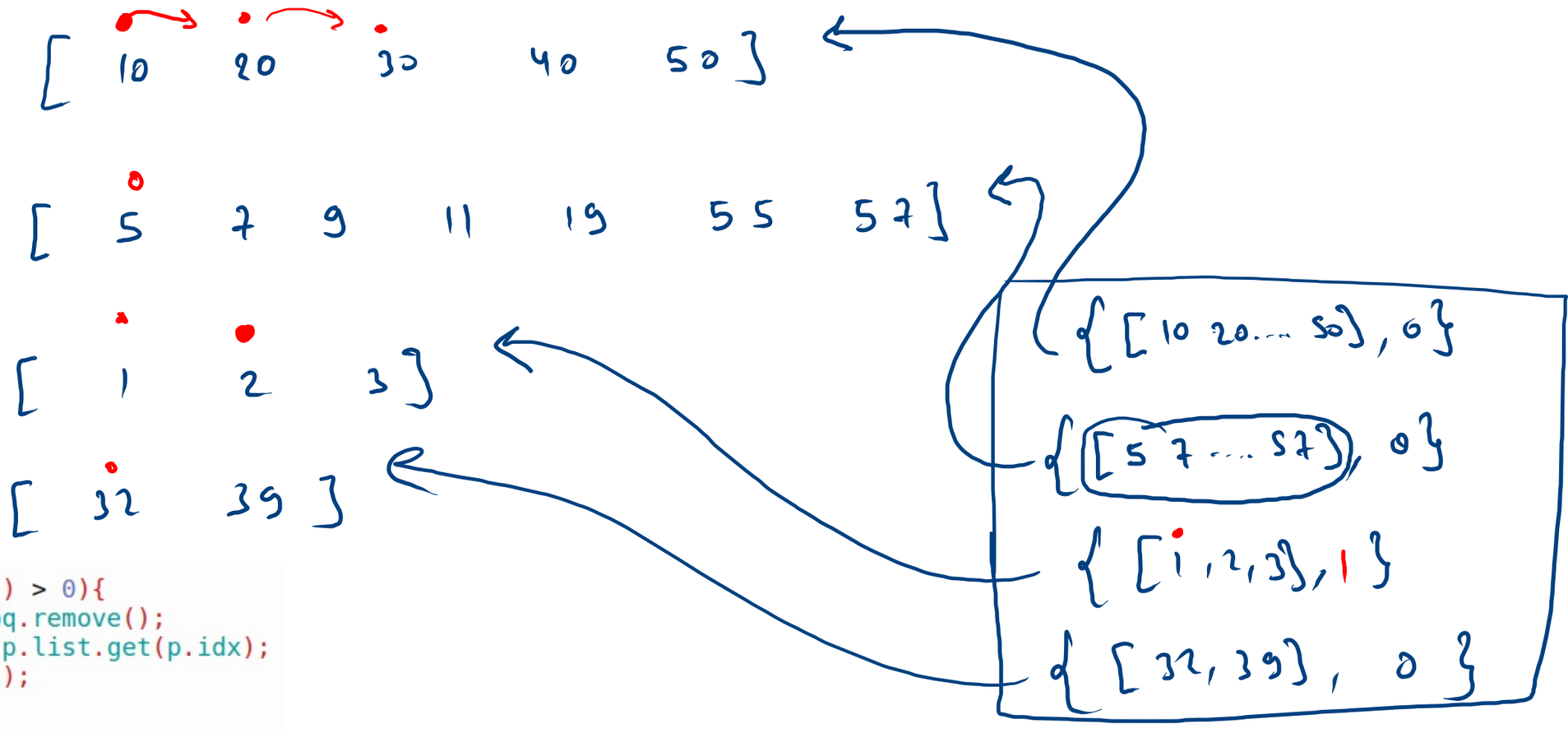


pair {
ArrayList32
int idx
}




```
PriorityQueue<Pair> pq = new PriorityQueue<>();

for(ArrayList<Integer> inner: lists){
    Pair p = new Pair(inner, 0);
    pq.add(p);
}
```




```
while(pq.size() > 0){
    Pair p = pq.remove();
    int val = p.list.get(p.idx);
    rv.add(val);

    p.idx++;
    if(p.idx < p.list.size()){
        pq.add(p);
    }
}
```

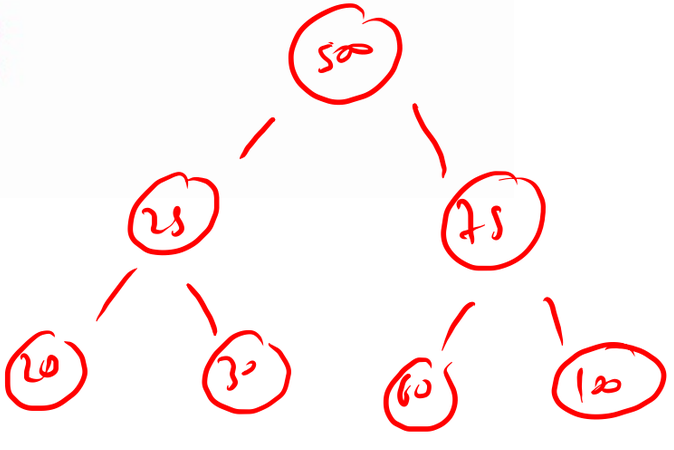
p =

```java
public static class Pair implements Comparable<Pair> {
    ArrayList<Integer> list;
    int idx;
    Pair(ArrayList<Integer> l, int i){
        list = l;
        idx = i;
    }

    public int compareTo(Pair b){
        Pair a = this;

        int x = a.list.get(a.idx);
        int y = b.list.get(b.idx);

        if(x < y)return -1;
        if(x > y)return +1;
        return 0;
    }
}
```




```java
public static ArrayList<Integer> mergeKSortedLists(A
  ArrayList<Integer> rv = new ArrayList<>();

    PriorityQueue<Pair> pq = new PriorityQueue<>()

    for(ArrayList<Integer> inner: lists){
        Pair p = new Pair(inner, 0);
        pq.add(p);
    }

    while(pq.size() > 0){
        Pair p = pq.remove();
        int val = p.list.get(p.idx);
        rv.add(val);

        p.idx++;
        if(p.idx < p.list.size()){
            pq.add(p);
        }
    }

  return rv;
}
```

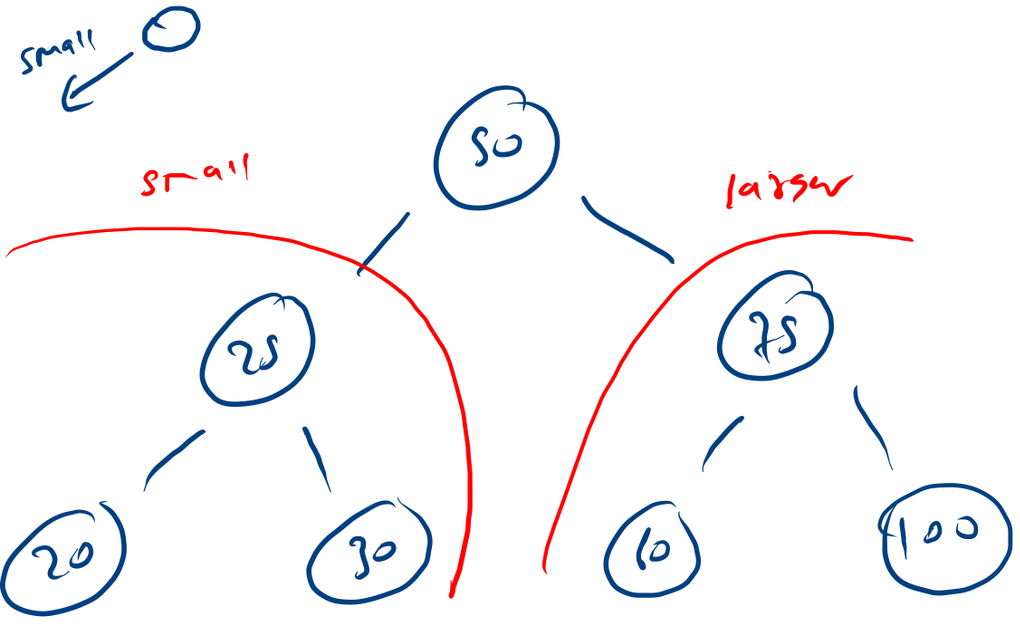
PQ → priority → min ✓
→ max

add	O(1)	log(n)
remove	O(n)	log(n)
Peek	O(n)	log(n)
size		

ar → n

n log(n)
```
for(int val: ar)
    pq.add(val)
```
← O(n)

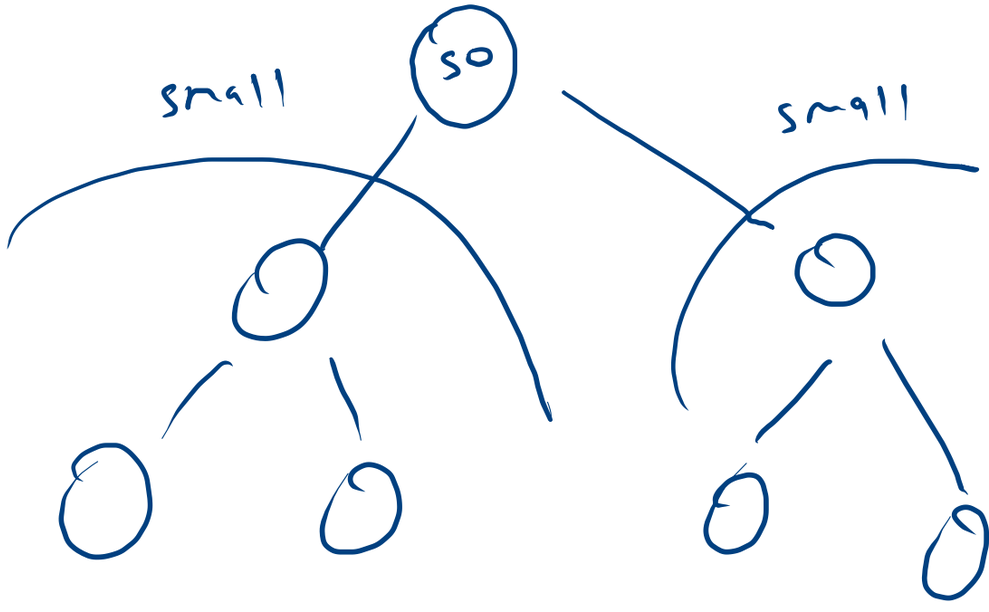
n log(n)
```
while( pq.size() > 0)
    print( pq.Peek())
    print (pq.remove)
```
O(n²)



add → log(n)

remove → log(n)

pick → log(n)



add → ?

remove → ?

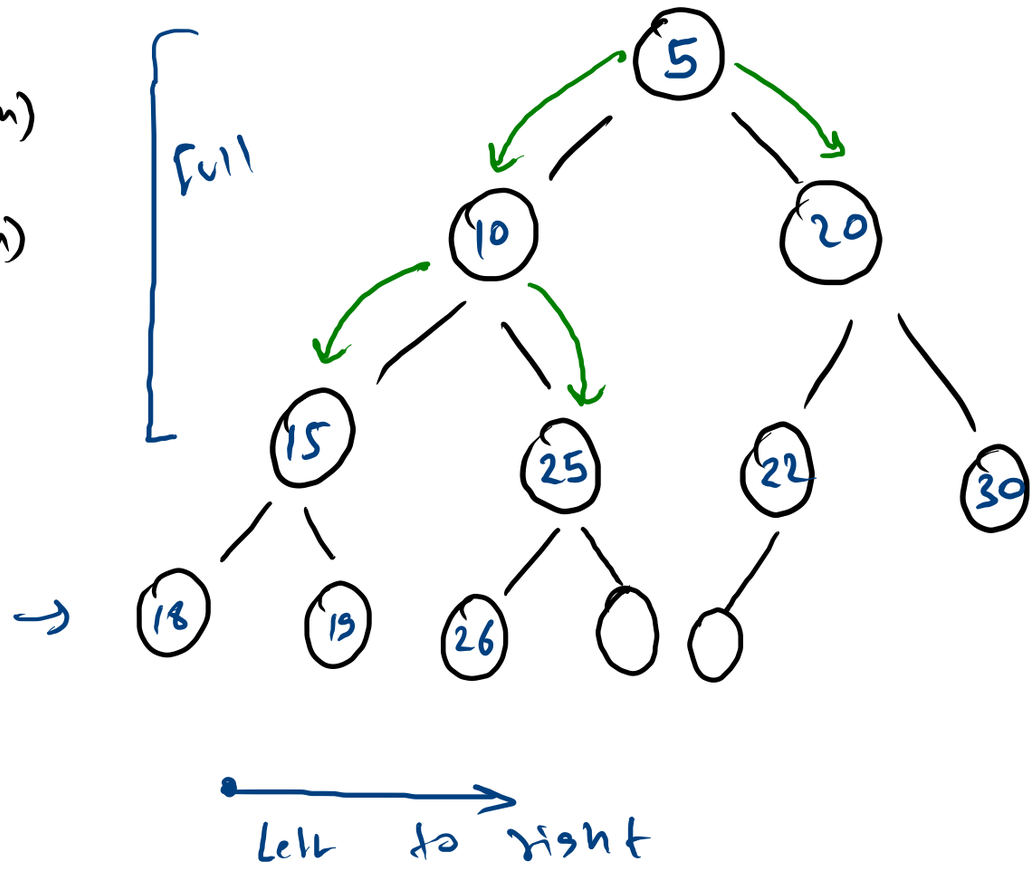
pick → O(1)

Heap → min priority

add → log(n)

remove → log(n)

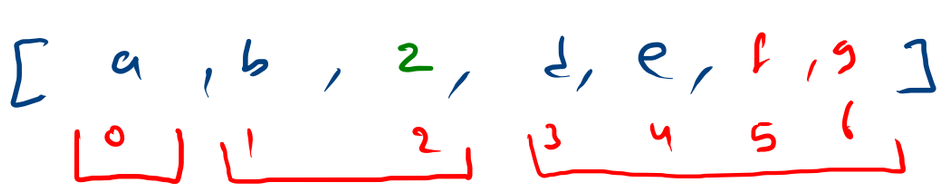
peek → O(1)

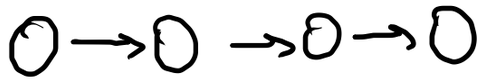


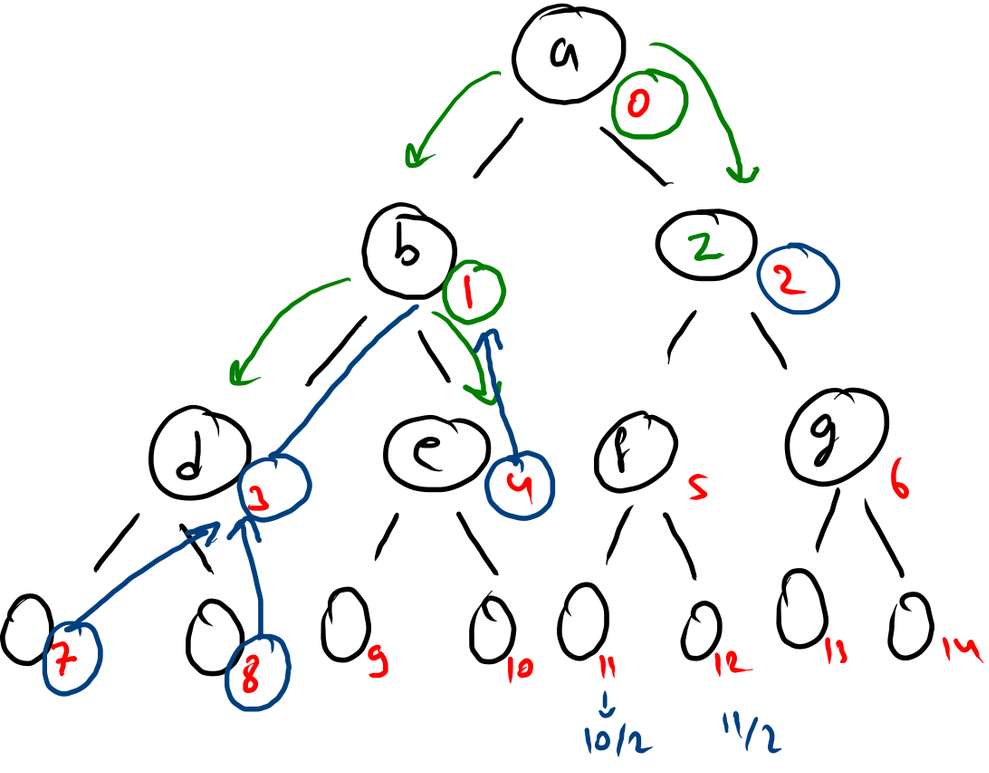
left to right

Property

- HOP → peek
- CBT → add, remove







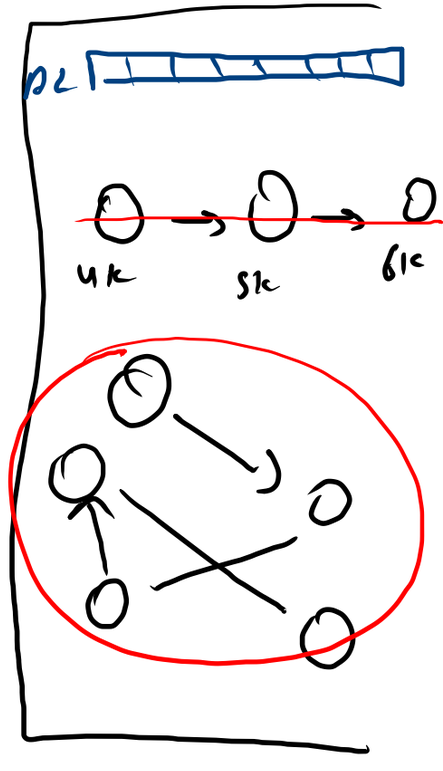
Pi

left → Pi*2 + 1

right → Pi*2 + 2

i

parent → (i-1)/2



$\frac{32}{32}$ $1 \rightarrow 2^0$

$\frac{32}{16}$ $2 \rightarrow 2^1$

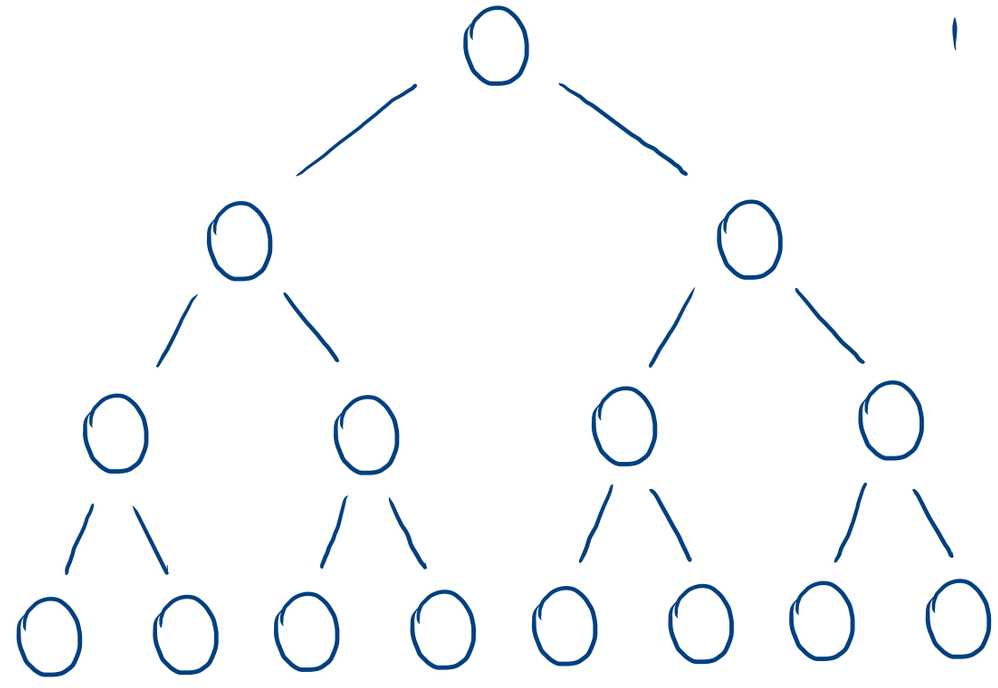
$\frac{32}{8}$ $4 \rightarrow 2^2$

$\frac{32}{4}$ $8 \rightarrow 2^3$

$\frac{32}{2}$ 16

$\frac{32}{1}$ 32

2^m



$\frac{32}{1}$

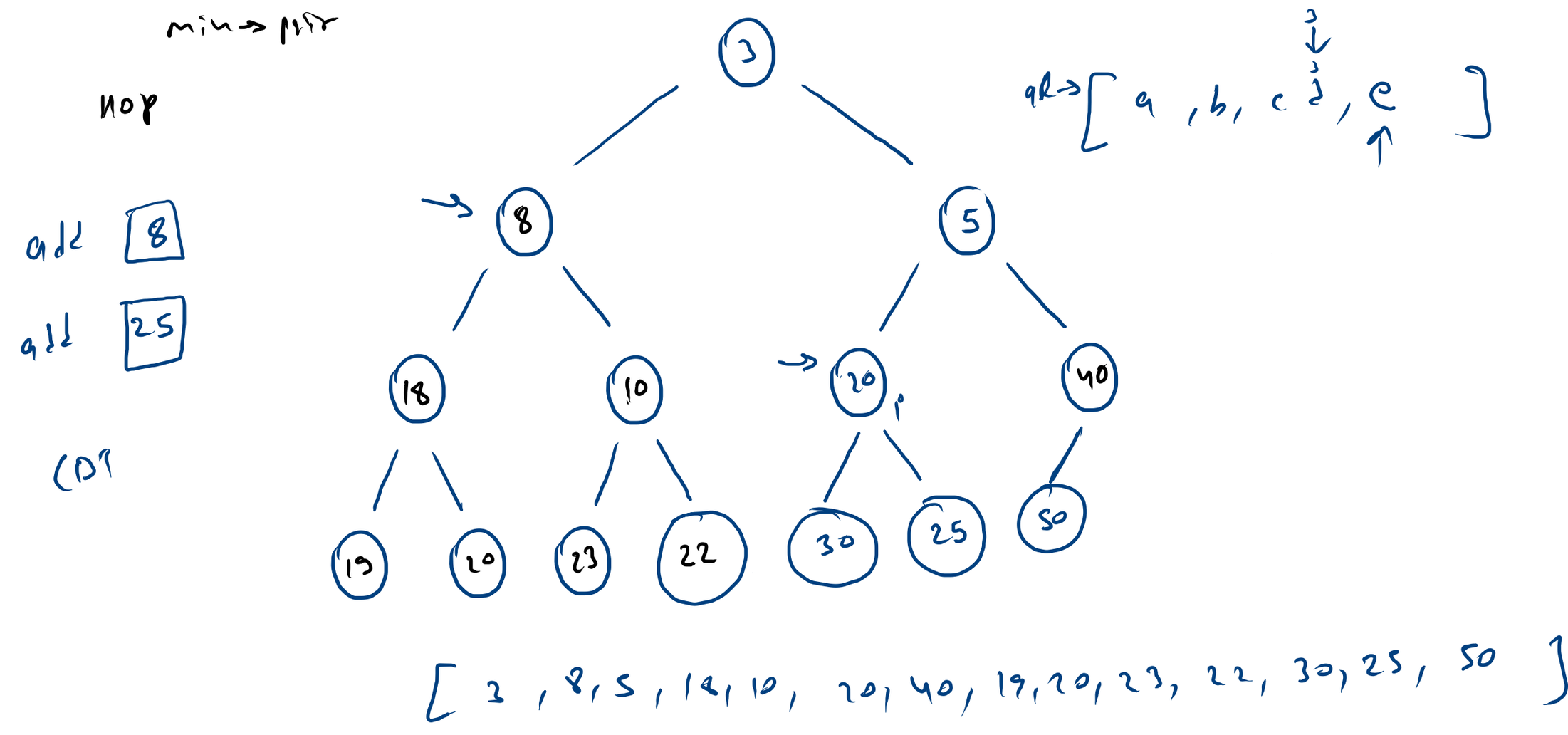
$\frac{32}{2}$	$\frac{32}{4}$	$\frac{32}{8}$	.....	1

32

$h \rightarrow 2^h$

$2^m = h$

$m = \log_2(h)$

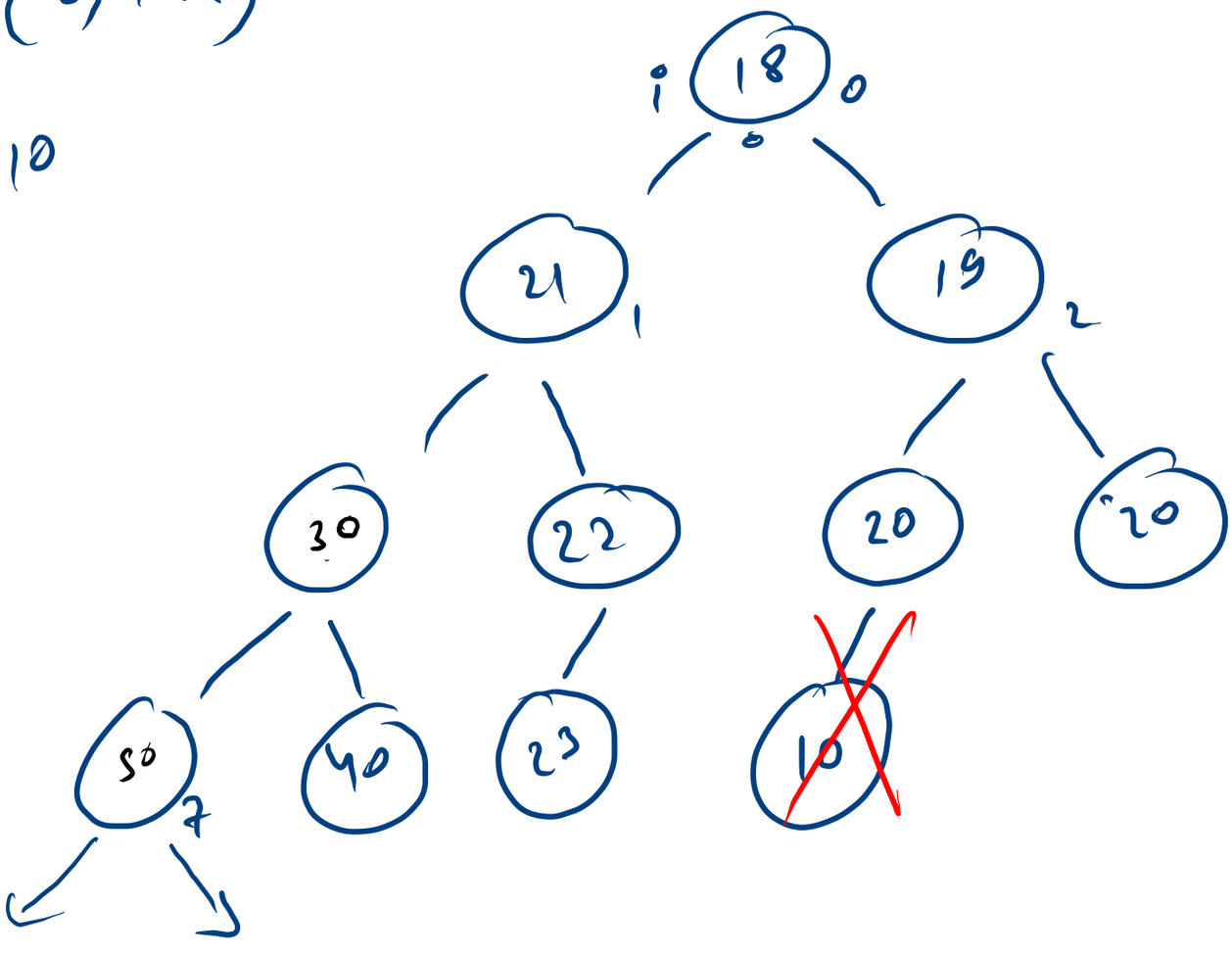

3
8
5
18
10
20
40
19
20
23
22
30
25
50
qR→ [a , b , c , e]
i
j

down heapify

remove swap(0, last)

val = 10

remove()

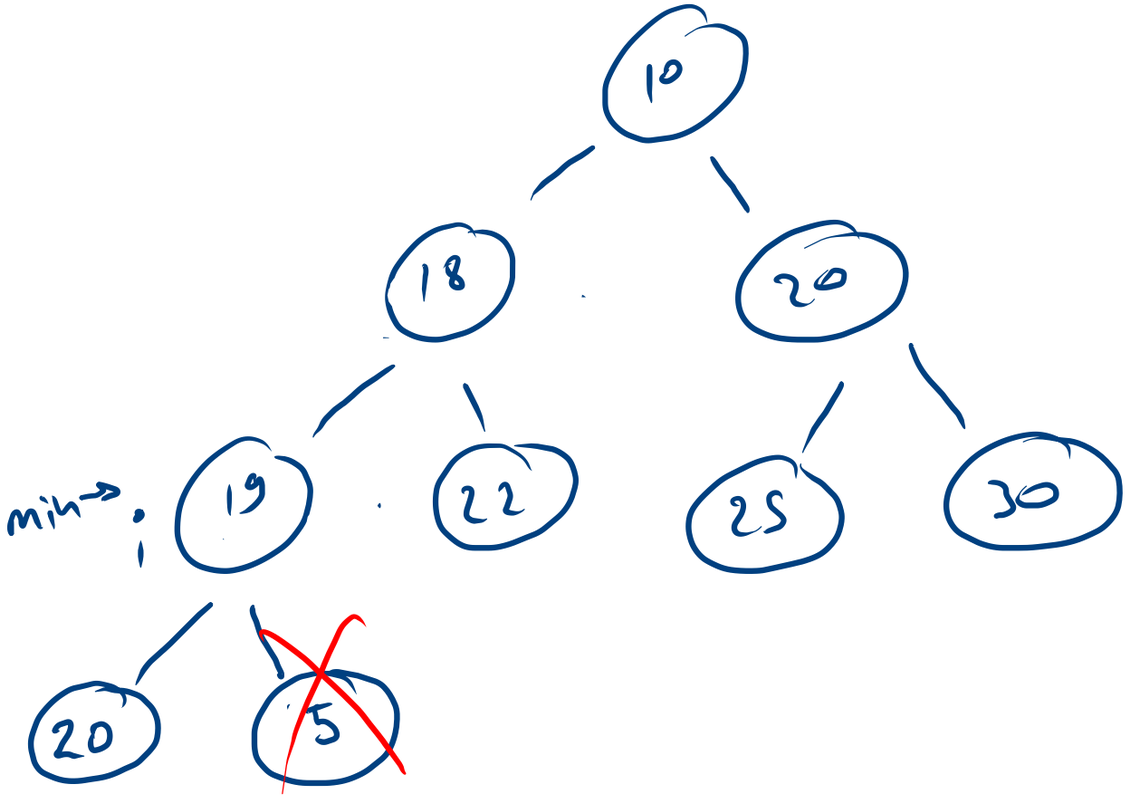


HOP

CBT

downheapify

val = 5



	put	get	containsKey	remove	keySet	size
✗	pv	null	false	null		0
✓	Update	value	True	value		val

TC → O(1) O(1) O(1) O(1) O(n) O(1)

key	value
India	~~100~~ 50
UK	25
US	20
Bangla	30

remove (UUK)

put

get

remove

contain

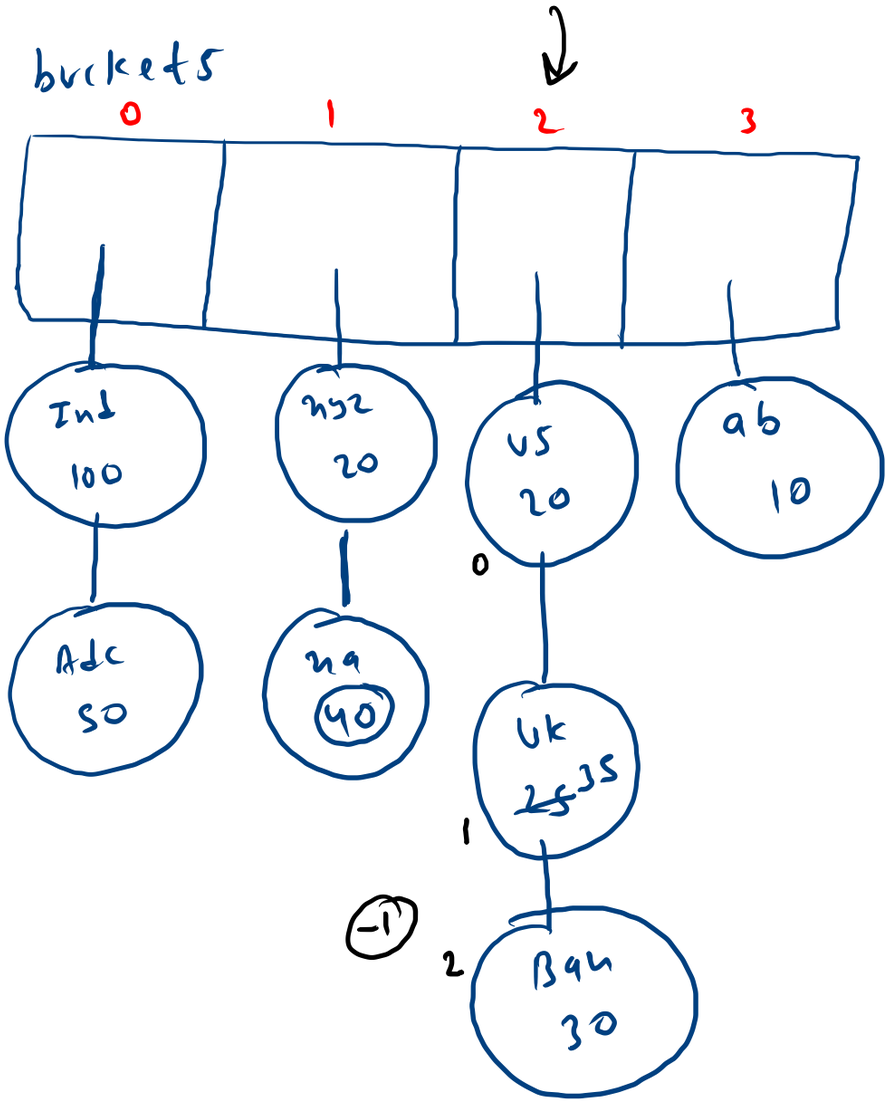
keyset

size

n → 8 size

N → 4

$\lambda \rightarrow \frac{n}{N}$ $\frac{8}{4} = 2$



$\lambda \leq k$ 2

$\frac{n}{N} \leq 2$

O(1)

put(UK, 35) 2 → H.F → int bi

put(na, 40) k

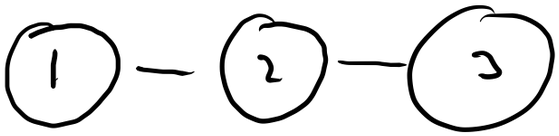
1

1

0

get(na) → 40

get(naa) → null

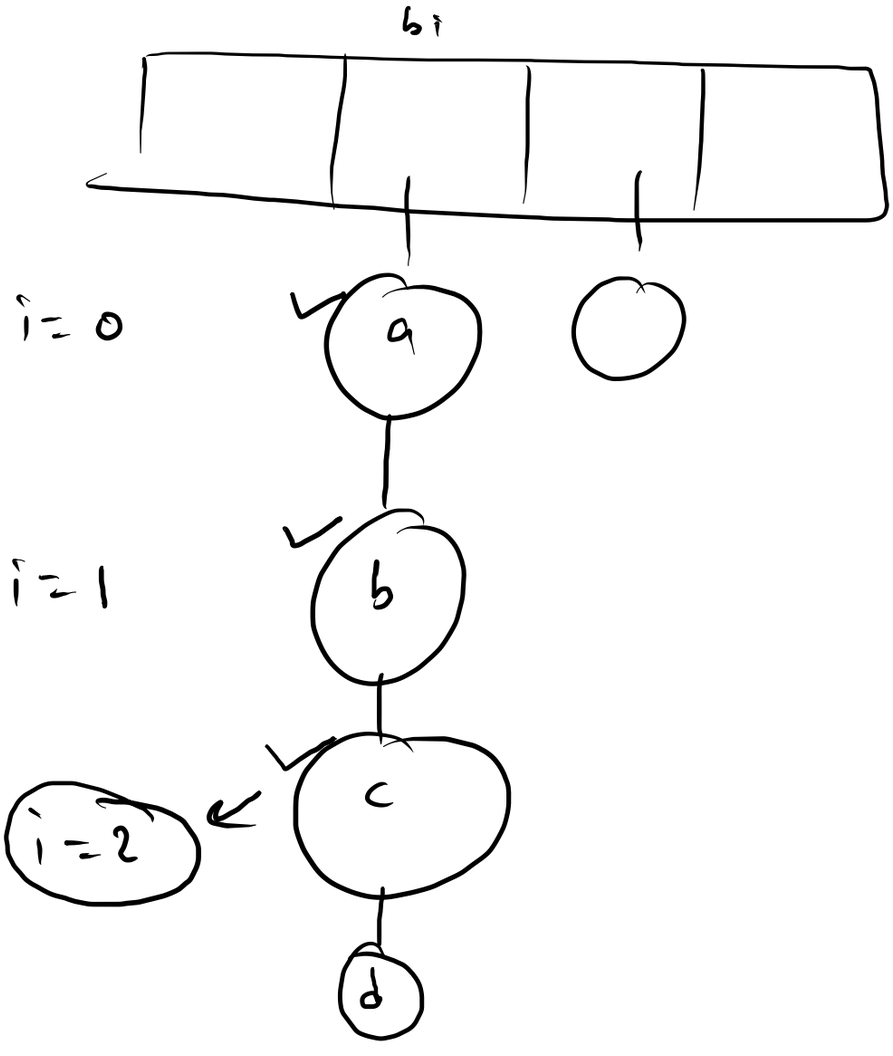


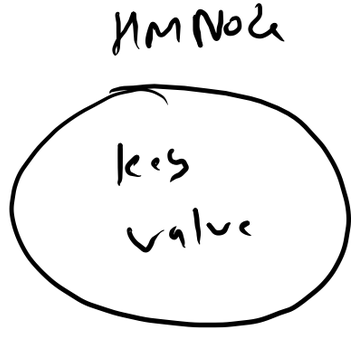

```java
private int getKeyInBucket(int bi, K key){

    int i=0;
    for(HMNode node : buckets[bi]){
        if(key.equals(node.key)){// == problem
            return i;
        }
        i++;
    }
    return -1;
}
```

key = c → 2

key = d → -1

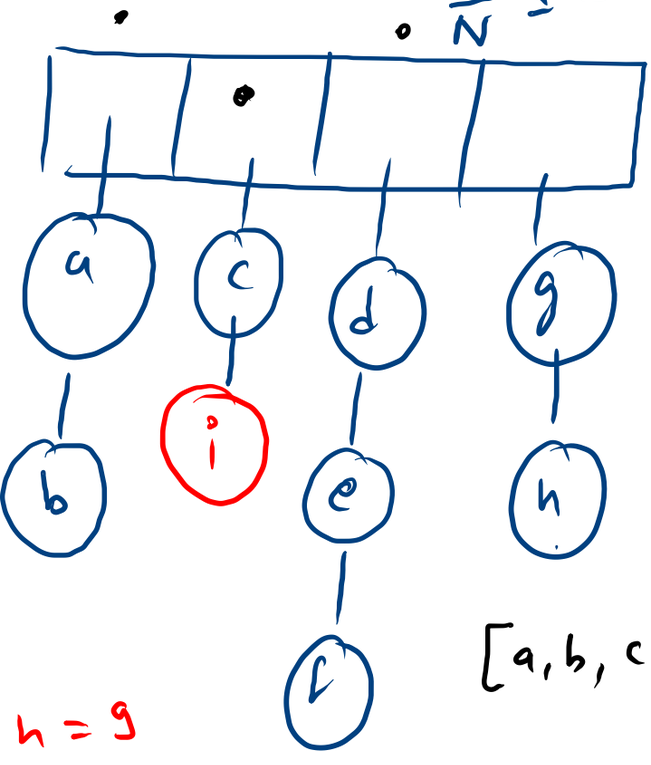




$h = 8$ $k = 2$

$N = 4$ $\lambda \leq k$

$\frac{h}{N} \leq 2$



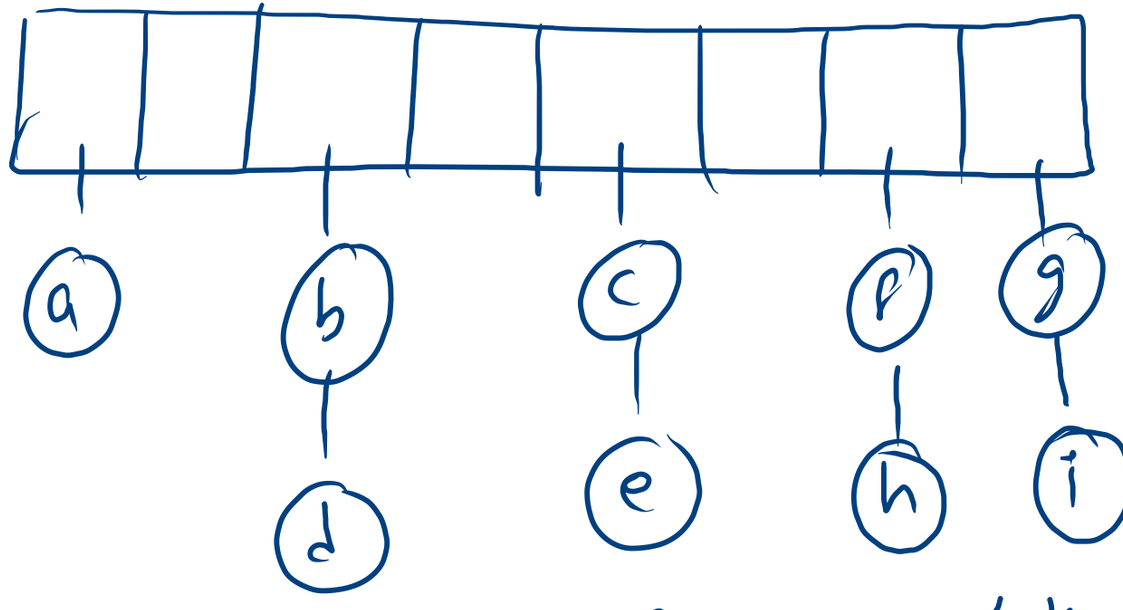
$[a, b, c, i, d, e, f, g, h]$

$h = 9$

$N = 4$

$\frac{h}{N} = \frac{9}{4} = 2.25$

$2.25 \leq 2$ ✗

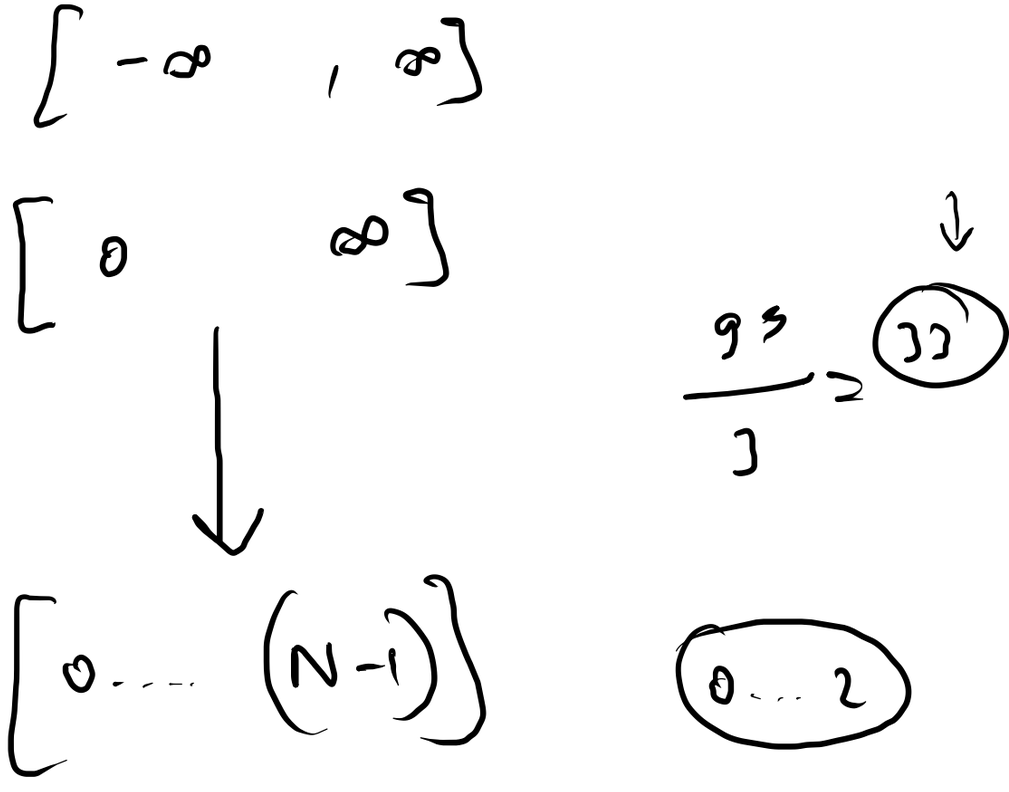


$h = 9$

$N = 8$

$\lambda = \frac{9}{8} = 1.125 \leq \frac{k}{2}$

```
int hs = keys.hashCode();
hs = Math.abs(hs); // -2 -> 2
                   // -3 -> 3
hs = hs % buckets.length;
return hs;
```



$$\frac{n}{N} \leq k$$

$$\frac{n}{N} > k \longrightarrow \text{rehashing}$$

$$n > N.k \longrightarrow \text{size} > \text{buckets.length} \cdot 2$$

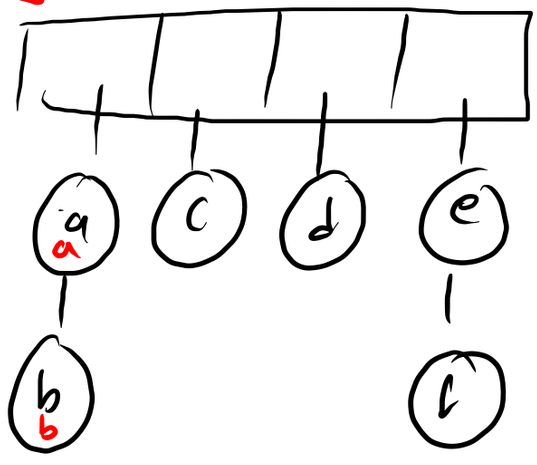
put(a, a')

put (b, b')

old buckets

~~buckets~~ →

~~size=6~~



buckets →

size= ~~6~~ 8

4

2

3

4