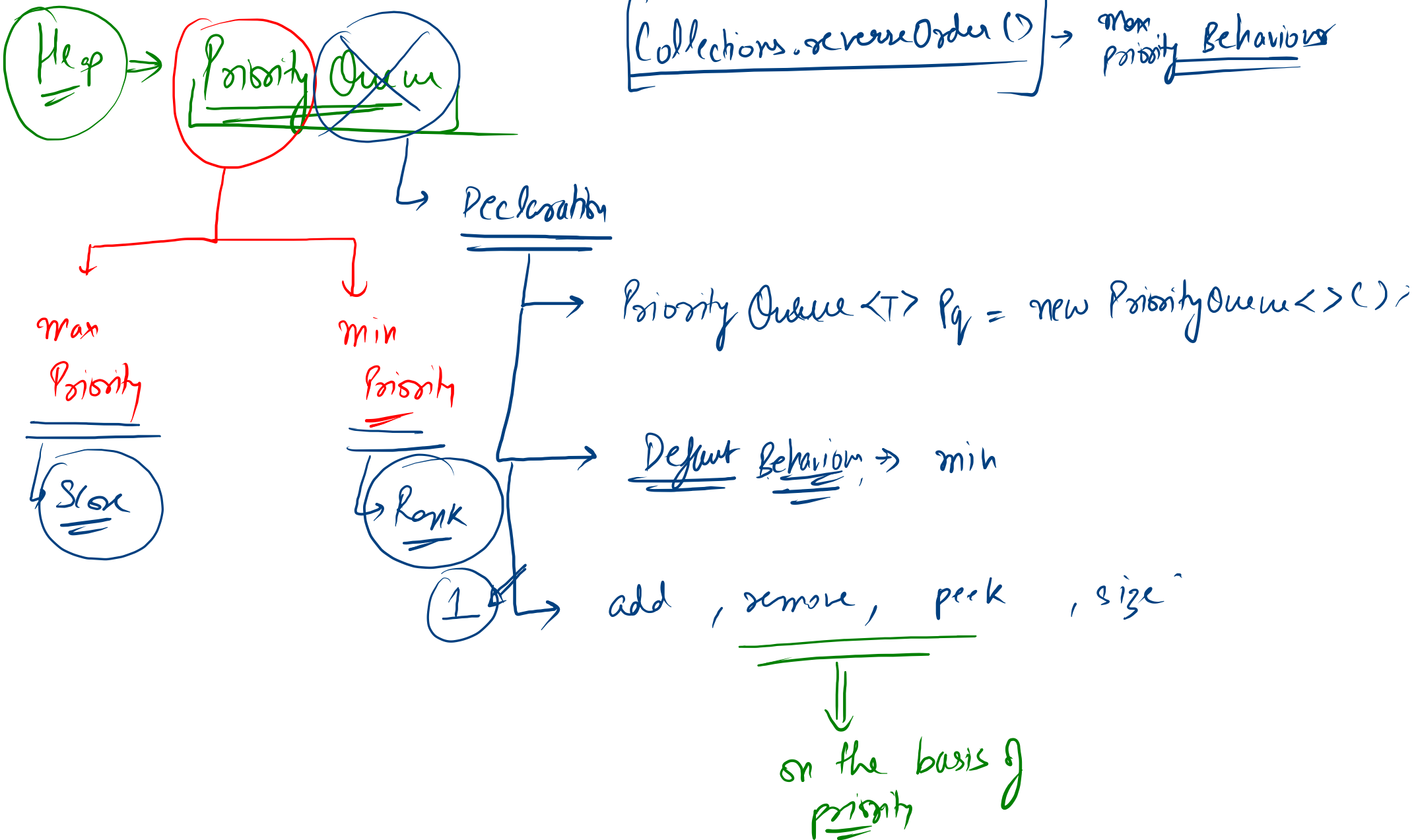
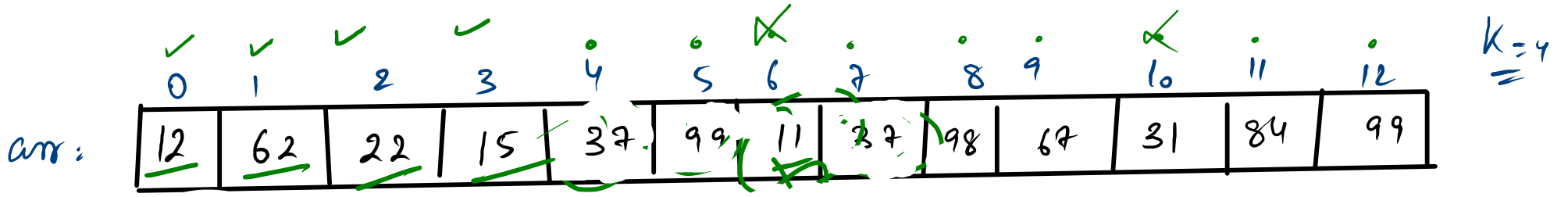

Heap
Priority Queue
Declaration
Collections.reverseOrder() → max priority Behaviour
Priority Queue<T> Pq = new PriorityQueue<>();
Default Behaviour → min
max Priority
Stack
min Priority
Rank
1
add , remove , peek , size
on the basis of priority

add(val)
remove()
⇒ O(log n) *, n = no. of elements in PQ.

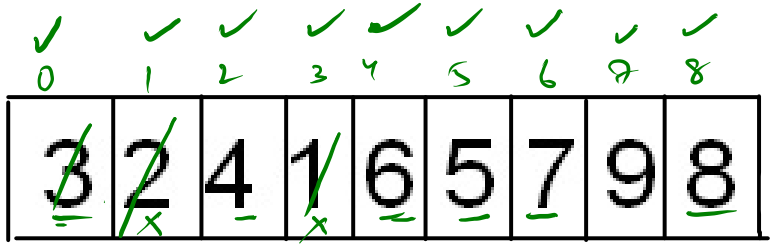
peek()
size()
⇒ O(1) Constant time *

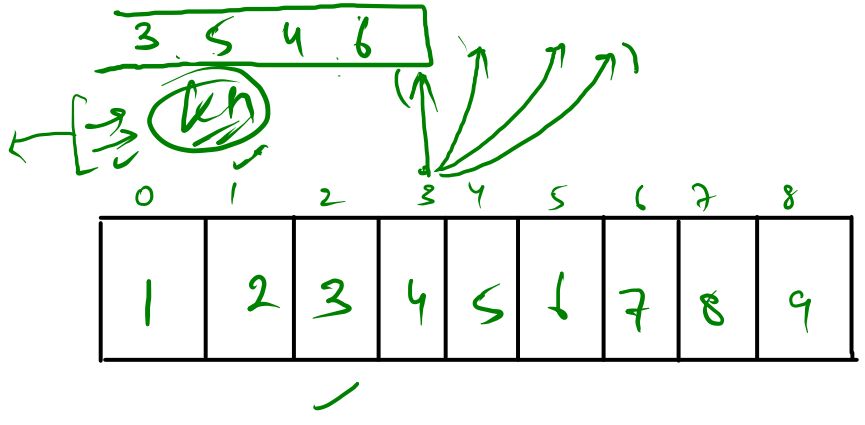


Implement ⇒

given → ① K sorted array
② k=3

arr =





Output → Sorted

k=3

$n \log(k+1)$ < $n \log n$

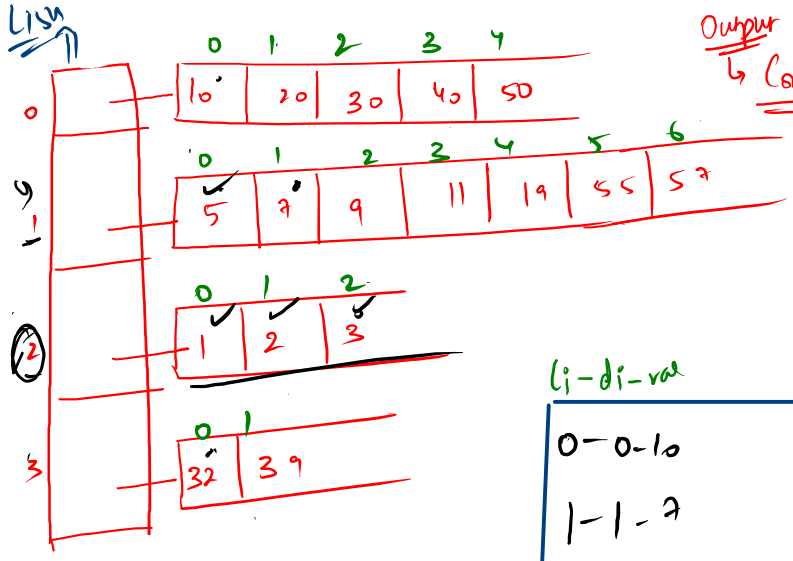
```
public static int[]  sort(int arr[],int k){
    PriorityQueue<Integer> pq = new PriorityQueue<>();

    int tmp = 0;
    int res[] = new int[arr.length];
    for(int idx = 0 ; idx < arr.length ; idx++){
        if(pq.size() < k+1){
            pq.add(arr[idx]);
        }else{
            res[tmp] = pq.remove();
            pq.add(arr[idx]);
            tmp++;
        }
    }

    while(pq.size() > 0){
        res[tmp] = pq.remove();
        tmp++;
    }

    return res;
}
```

Queue size

Queue ops

n → add => log n → n log(n)

k+1 → add => log k+1 → n log(k+1)



Output
↳ Complete Sorted List

Pq → min

nlogk

li-di-val

0-0-10

1-1-7

~~2-2-3~~

3-0-32

1-1-7


```
ArrayList<Integer> rv = new ArrayList<>();

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

for(int idx = 0 ; idx < lists.size() ; idx++){
    pq.add(new Pair(idx,0,lists.get(idx).get(0)));
}

while(pq.size() > 0){
    Pair minPair = pq.remove();

    rv.add(minPair.val);

    if(minPair.di+1 < lists.get(minPair.li).size()){
        minPair.di = minPair.di+1;
        minPair.val = lists.get(minPair.li).get(minPair.di);
        pq.add(minPair);
    }
}
return rv;
```

min?

Priority Queue

2

Parent must always contain higher priority element

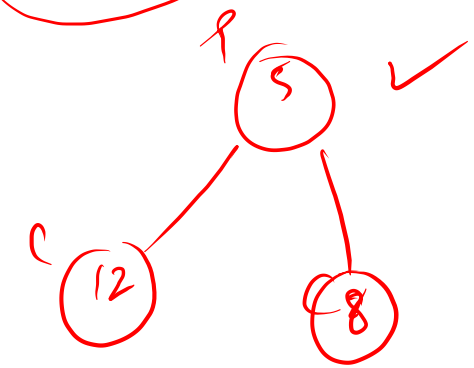
H.O. Property

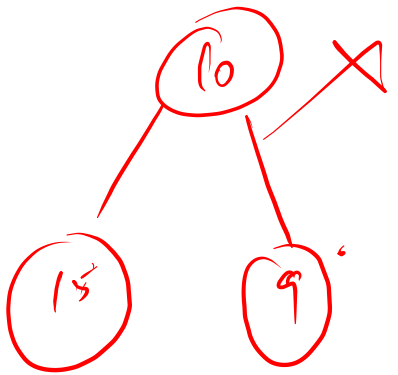
Parent

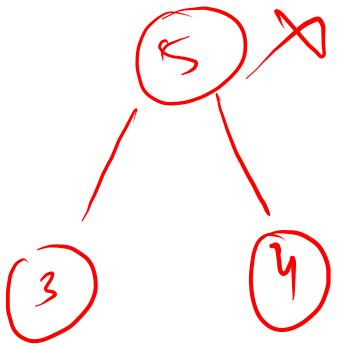
Child 1

Child 2

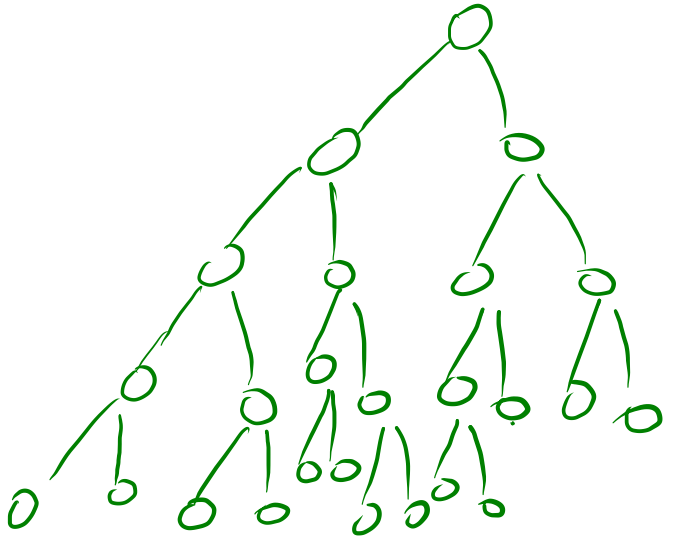
min - Priority





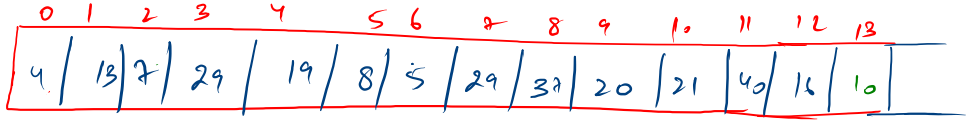


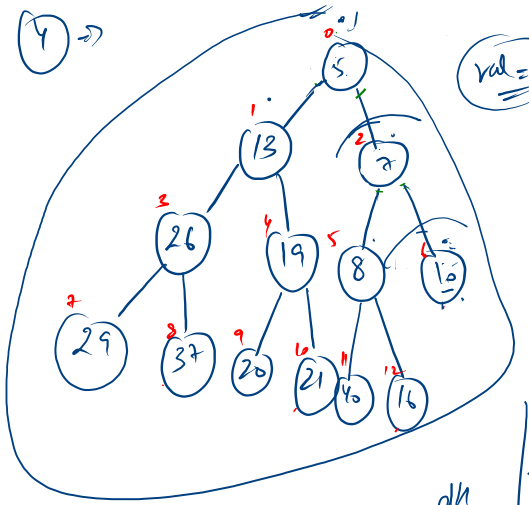
CBT



Complete Binary Tree

- only last level can be incomplete
- addition, L to R






```
public int remove() {
    if(data.size() == 0){
        System.out.println("Underflow");
        return -1;
    }
    int val = data.get(0);
    data.set(0,data.get(data.size()-1));
    data.remove(data.size()-1);
    downHeapify(0);
    return val;
}
```

val = 4

minIdx = 6

```
public void downHeapify(int idx){
    int lidx = (2*idx)+1;
    int ridx = (2*idx)+2;

    int minIdx = idx;

    if(lidx < data.size() && data.get(lidx) < data.get(minIdx)){
        minIdx = lidx;
    }

    if(ridx < data.size() && data.get(ridx) < data.get(minIdx)){
        minIdx = ridx;
    }

    if(minIdx != idx){
        int val = data.get(idx);
        int minVal = data.get(minIdx);

        data.set(idx,minVal);
        data.set(minIdx,val);

        downHeapify(minIdx);
    }
}
```

dh(6)
dh(2)
dh(0)

cidx → pidx
2 → 0
9 → 4
10 → 4

idx
↳ Lidx = (2 * idx) + 1
↳ Ridx = (2 * idx) + 2

cidx
↳ pidx = (cidx - 1) / 2

H.O.P ✓ ✓

C.B.T. ✓ (?)

(data[cidx] < data[pidx])

✓ (1) < 28

✗ (2) > 27