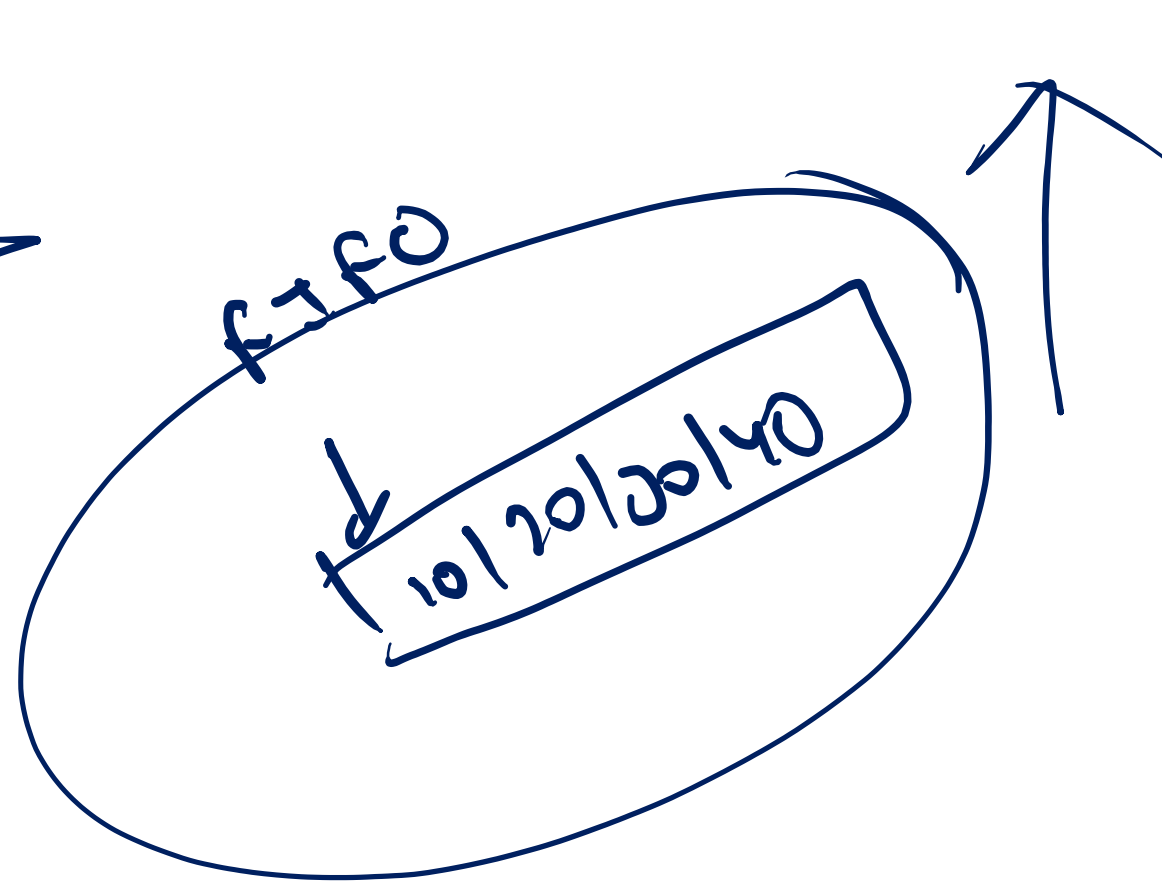
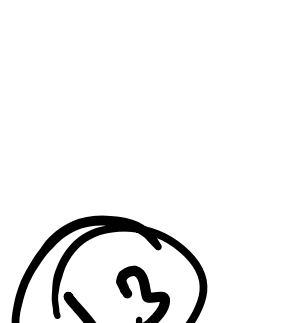
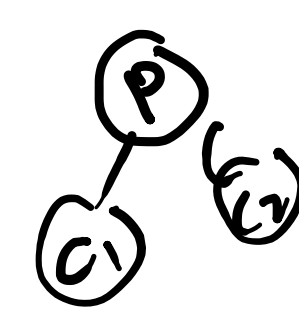
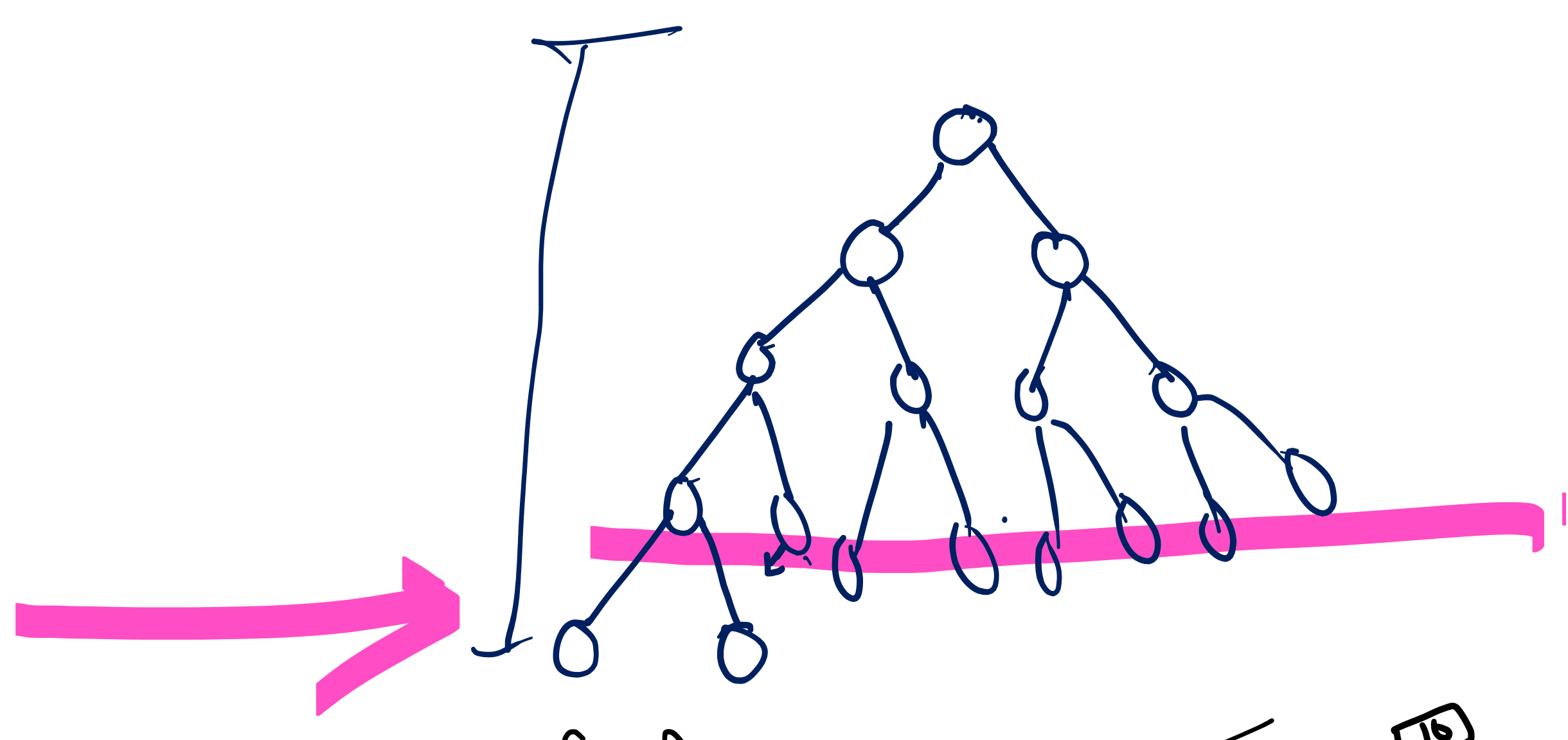


S-4

Heap / priority queue



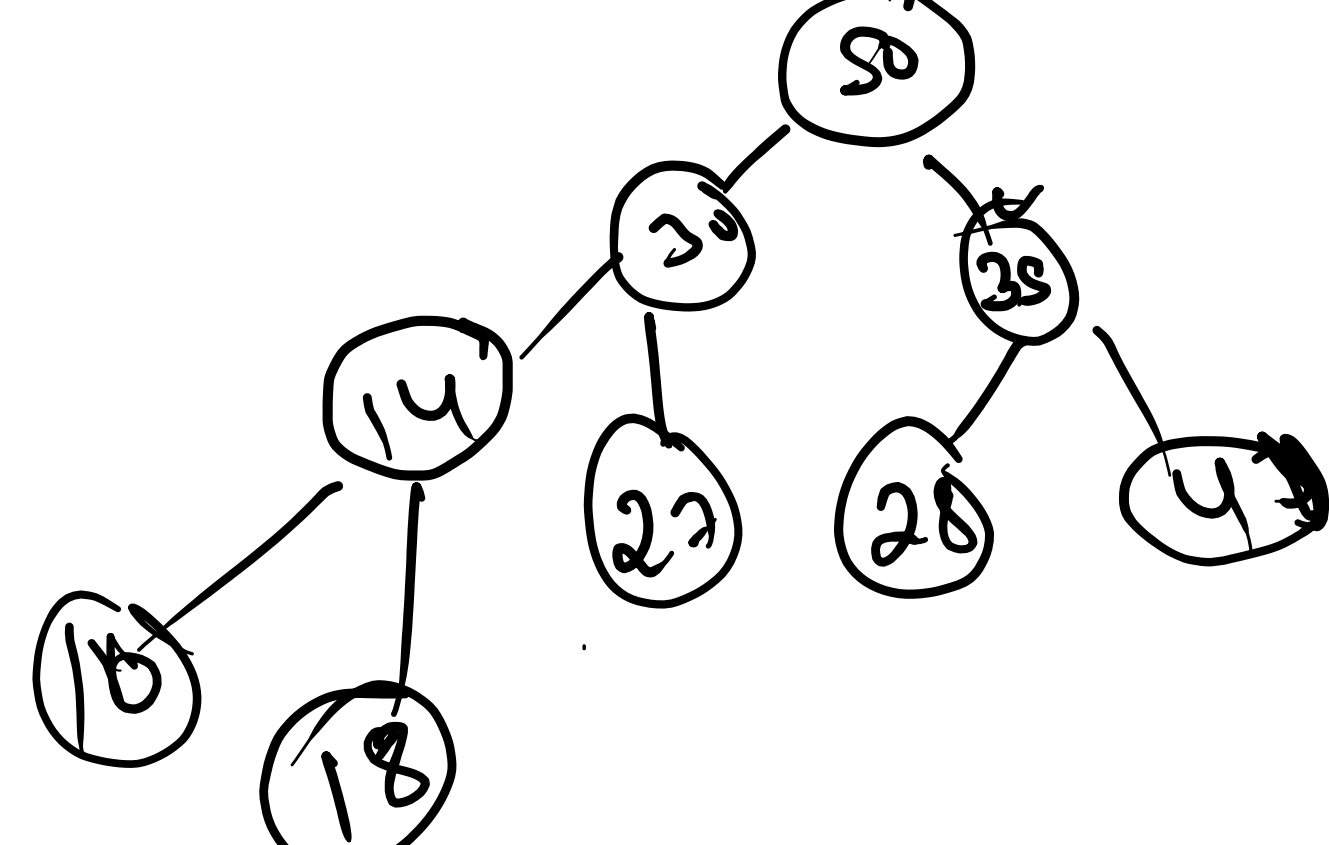
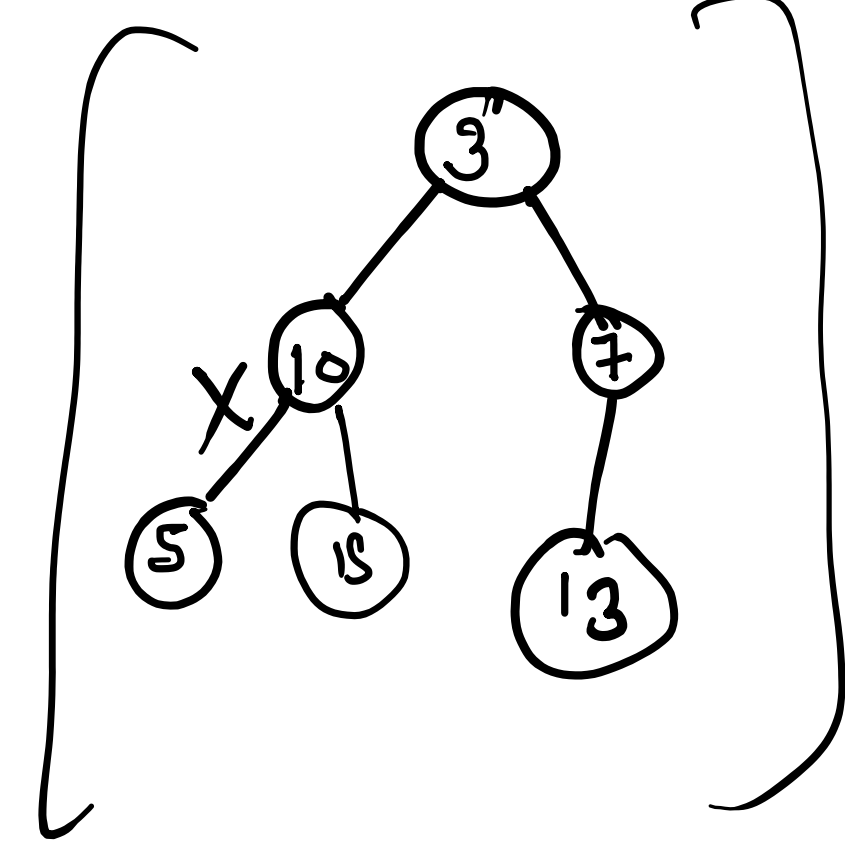
Heap
S + 10 CBT & has some



PT (c1, c2)

min heap

min heap
max heap



upheapify

0/2 = 0

[10, 20, 30, 1, 5, 7, 3, 2, -4]

min heap

11.add(item)

6/2 = 3

3/2 = 1

1/2 = 0

8/2 = 4

Private void upheapify(int ci) {
// TODOAuto-generated method stub
int pi = (ci - 1) / 2;
if (ll.get(pi) > ll.get(ci)) {
swap(pi, ci);
upheapify(pi);
}
}

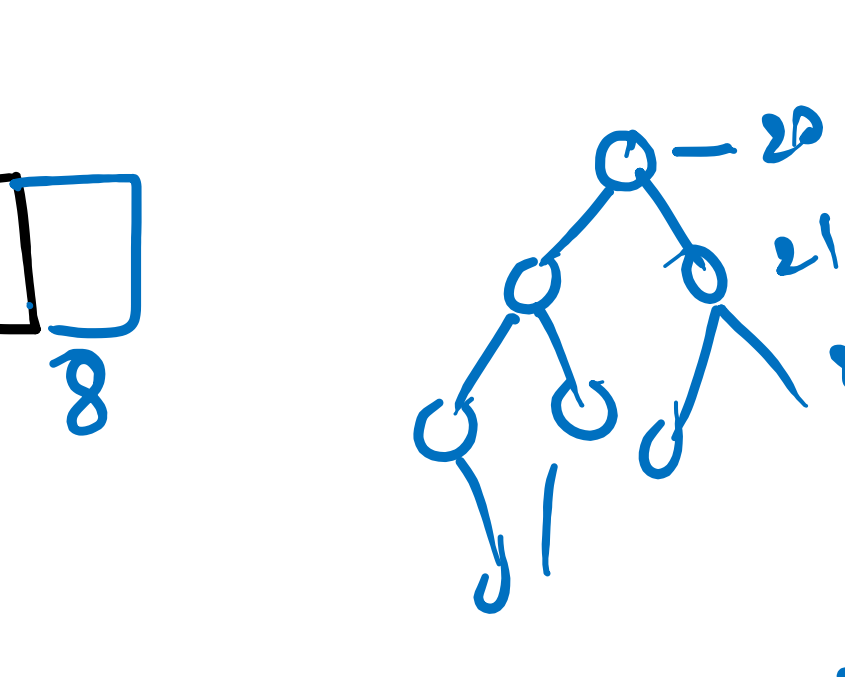
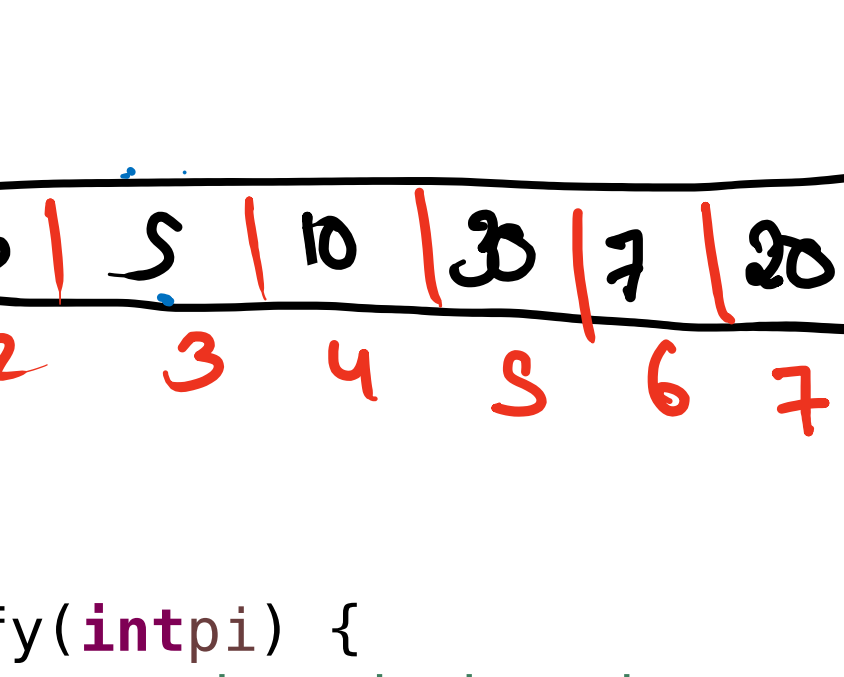
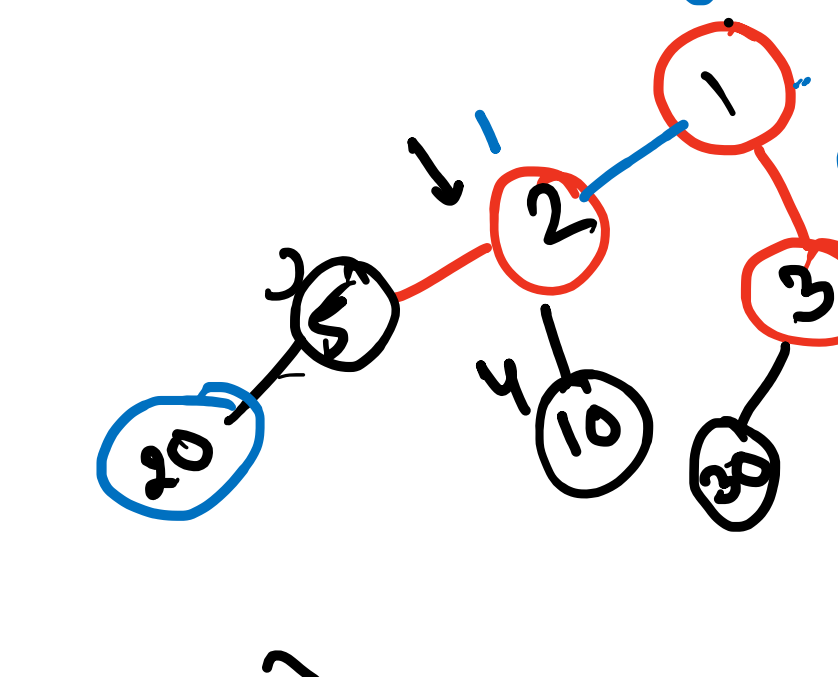
downheap

log(m)

0(m)

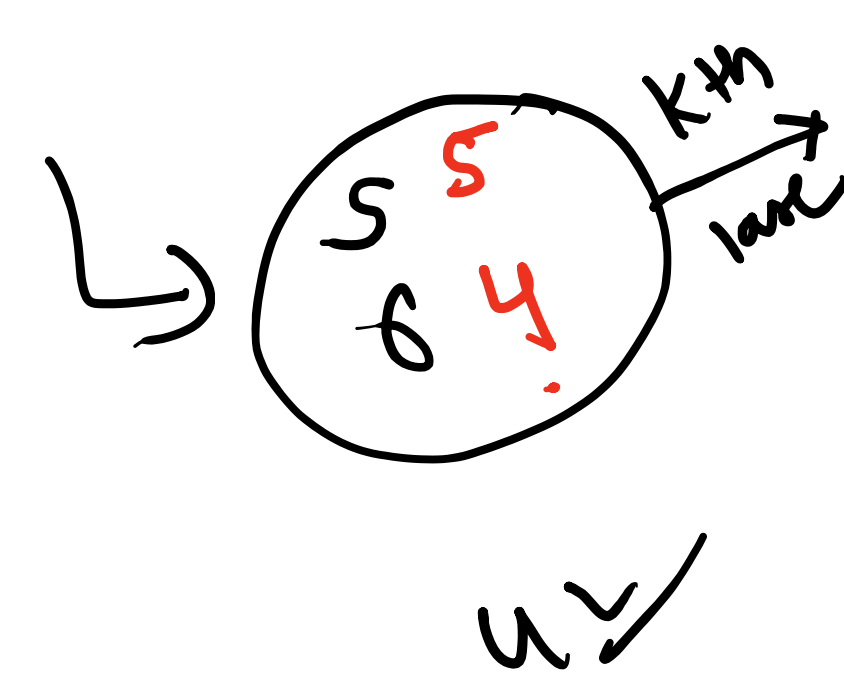
h = log N

2^h = N

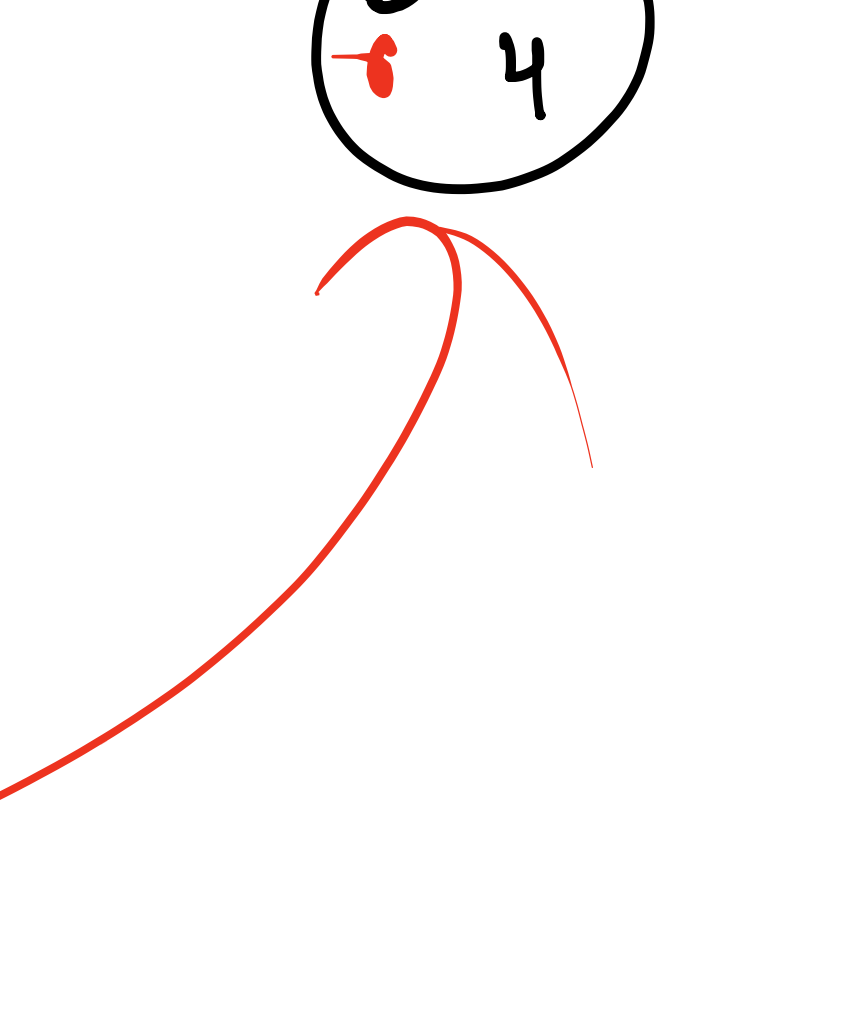
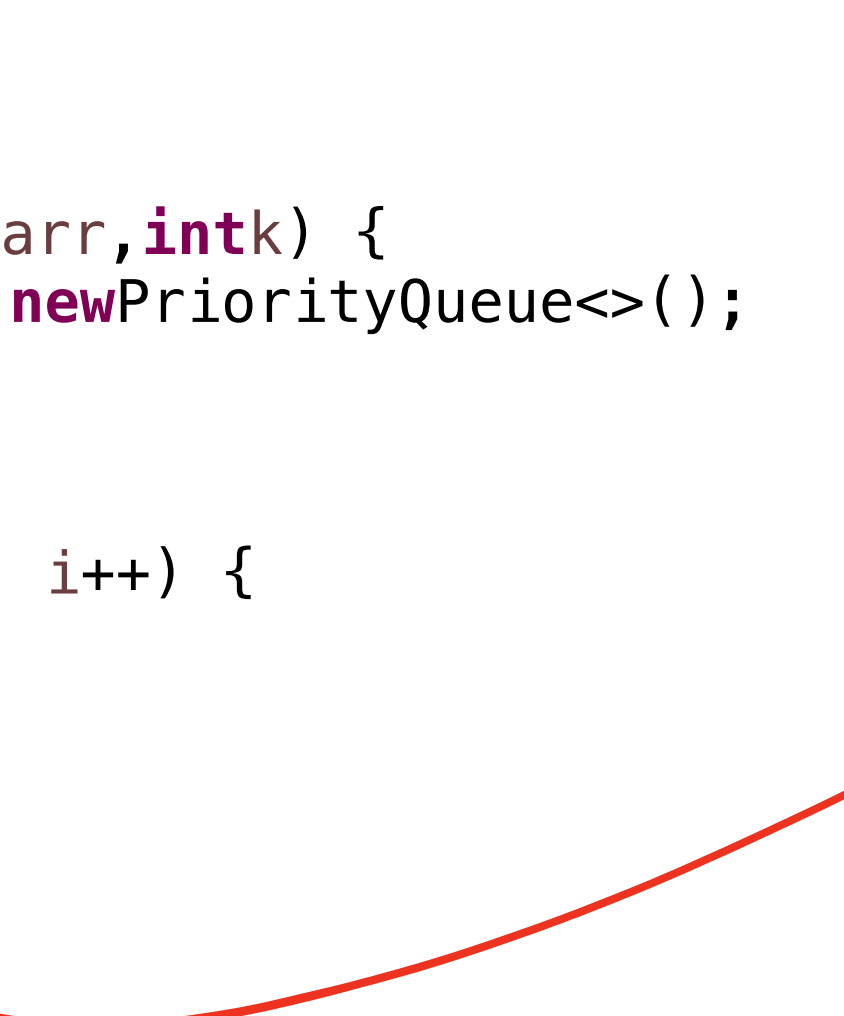


Private void downheapify(int pi) {
// TODOAuto-generated method stub
int lci = 2 * pi + 1;
int rci = 2 * pi + 2;
int mini = pi;
if (lci < ll.size() && ll.get(lci) < ll.get(mini)) {
mini = lci;
}
if (rci < ll.size() && ll.get(rci) < ll.get(mini)) {
mini = rci;
}
if (mini != pi) {
swap(mini, pi);
downheapify(mini);
}
}

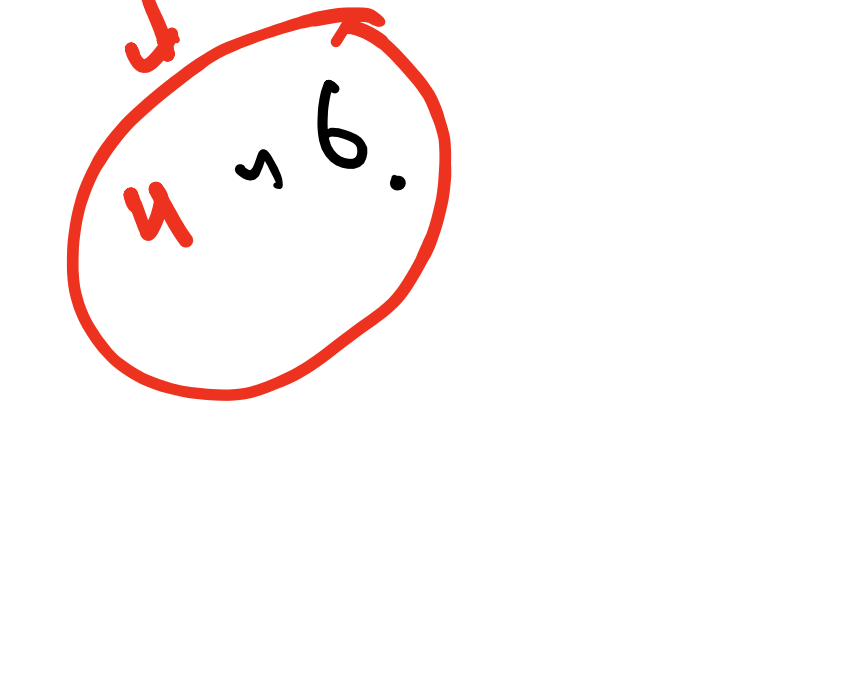
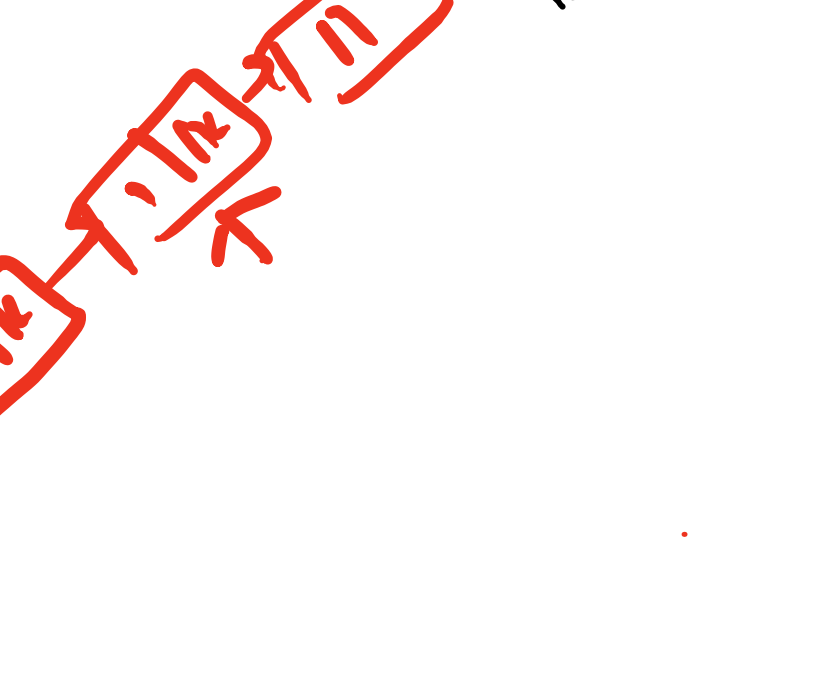
3, 2, 3, 1, 2, 4, 5, 5, 6, k = 4



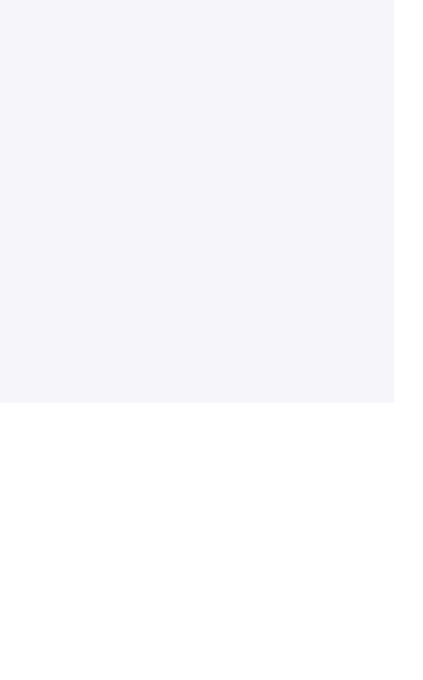
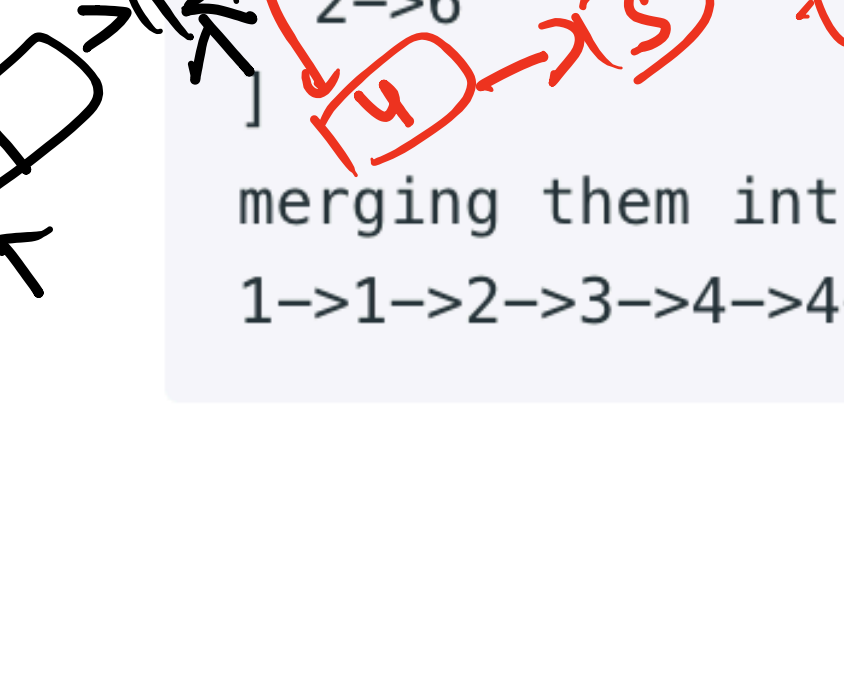
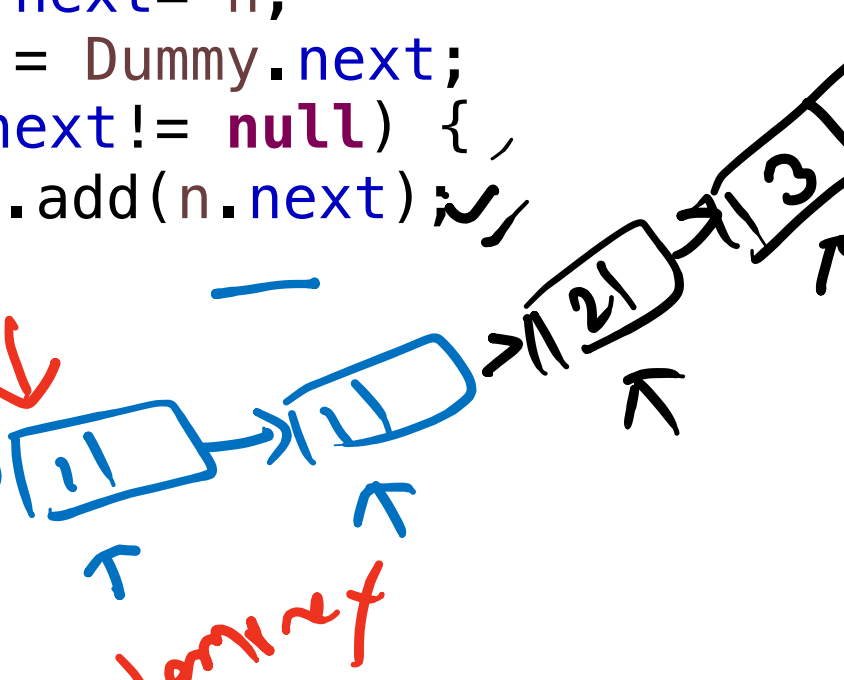
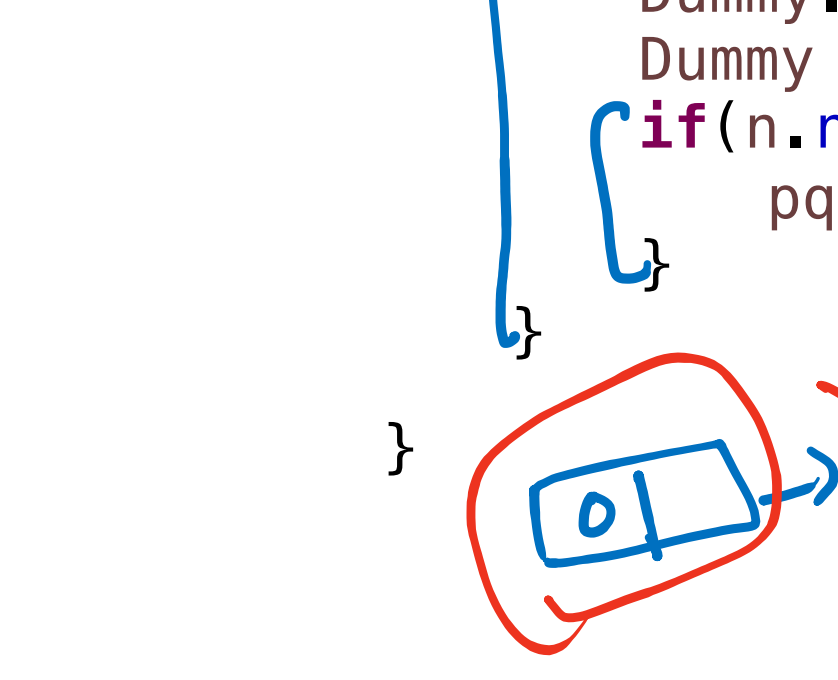
Public static int kthlargest(int[] arr, int k) {
PriorityQueue<Integer> pq = new PriorityQueue<>();
for (int i = 0; i < k; i++) {
pq.add(arr[i]);
}
for (int i = k; i < arr.length; i++) {
if (arr[i] > pq.peek()) {
pq.poll();
pq.add(arr[i]);
}
}
return pq.peek();
}



Public ListNode mergeKLists(ListNode[] lists) {
PriorityQueue<ListNode> pq = new PriorityQueue<>();
for (int i = 0; i < lists.length; i++) {
if (lists[i] != null) {
pq.add(lists[i]);
}
}
ListNode Dummy = new ListNode();
ListNode temp = Dummy;
while (!pq.isEmpty()) {
ListNode n = pq.poll();
Dummy.next = n;
Dummy = Dummy.next;
if (n.next != null) {
pq.add(n.next);
}
}
return Dummy.next;
}



Public ListNode mergeKLists(ListNode[] lists) {
PriorityQueue<ListNode> pq = new PriorityQueue<>();
for (int i = 0; i < lists.length; i++) {
if (lists[i] != null) {
pq.add(lists[i]);
}
}
ListNode Dummy = new ListNode();
ListNode temp = Dummy;
while (!pq.isEmpty()) {
ListNode n = pq.poll();
Dummy.next = n;
Dummy = Dummy.next;
if (n.next != null) {
pq.add(n.next);
}
}
return Dummy.next;
}



Input: lists = [[1,4,5],[1,3,4],[2,6]]
Output: [1,1,2,3,4,4,5,6]
Explanation: The linked-lists are:
[1,4,5]
[1,3,4]
[2,6]
merging them into one sorted list:
1->1->2->3->4->4->5->6

