Implement a queue using two stacks

enque(1)

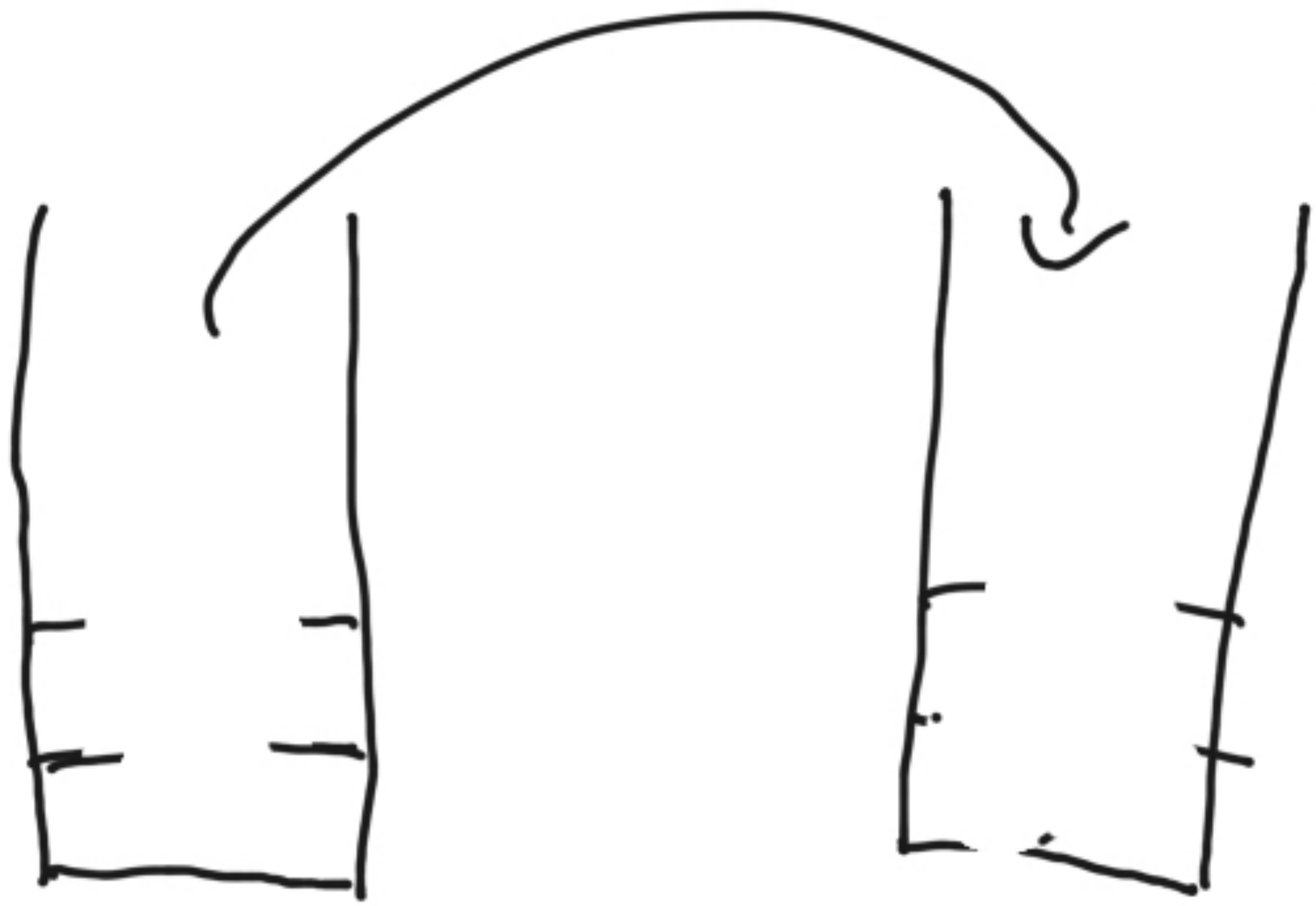
enque(2)

deque() → 1 ✓

enque(3) ✓

deque() → 2 ✓

deque() → 3 ✓



in

out

```
enque(x)
    in.push(x)                      | O(1)

deque()
    if outstack is empty:

        # move all entries from in to out
        while in stack is not empty:          } O(N)
            out.push(in.pop())

    return out.pop()        O(1)
```

Selection problem

8	3	1	9	2	7	4	6	5

select the k-th largest entry

Idea 1: sort, then simply take 3rd-last element.
$O(N \log N)$

Idea 2: find max 3 times. remove largest element each time. $O(k \cdot N)$

Idea 3:

$$li = [\,8 \mid 3 \mid 1 \mid 9 \mid 2 \mid 7 \mid 4 \mid 6 \mid 5\,] \qquad O(N)$$

$$tmp = [\,8 \mid 3 \mid 9\,]$$

if element at index p is greater than min of list tmp, replace that min with $li[p]$ $\qquad O(k)$

finally return min of tmp.

total: $O(N \cdot k)$

finding median:
$k = \frac{N}{2} \Rightarrow O(N^2)$

$$O(N \cdot \log k) \Rightarrow O(N \log N)$$

Heap (a type of priority queue)

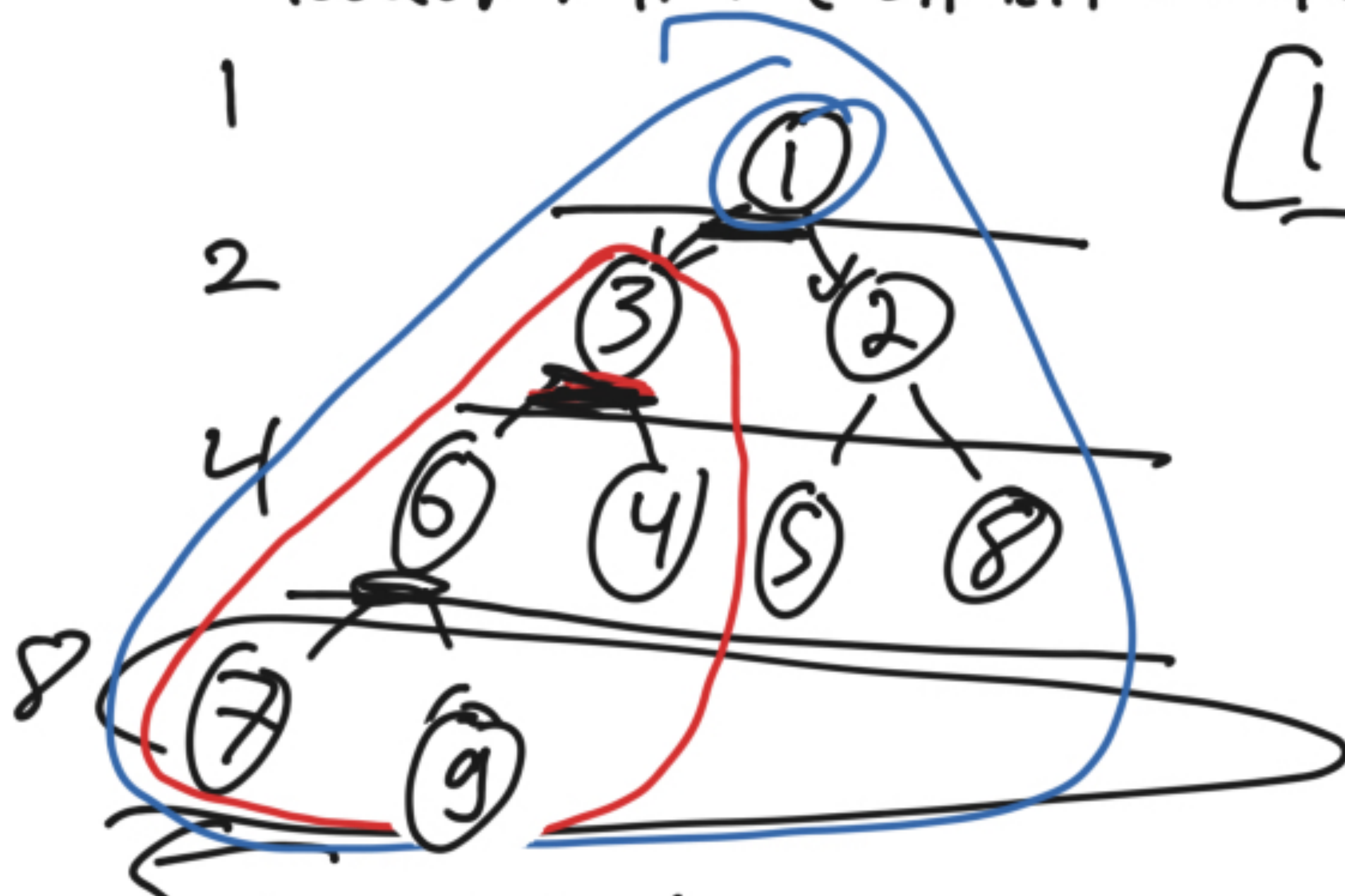
stack: LIFO

queue: FIFO

Heap: first in best out

Binary Heap

- insert(x) a.k.a. heappush $O(\log n)$
- deleteMin() a.k.a. heappop $O(\log n)$
  lookupMin (without delete) $O(1)$



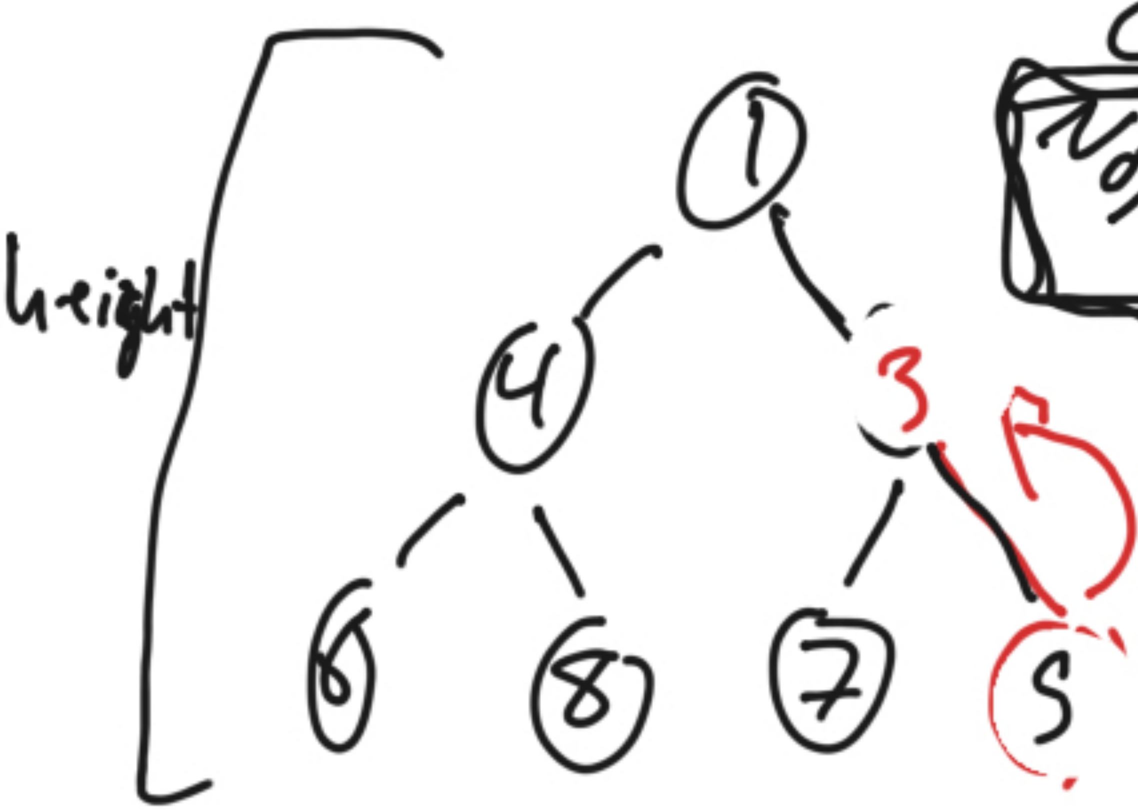
Complete binary tree

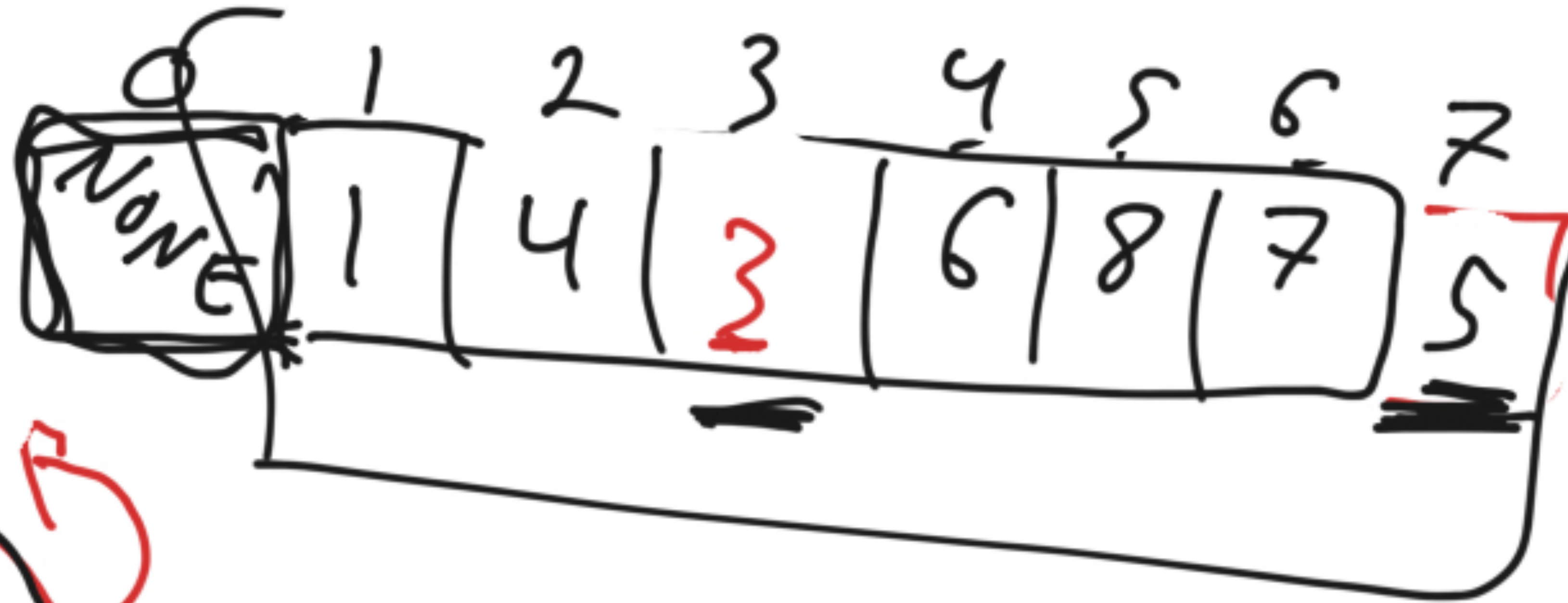
[1 3 2 6 4 5 8 7 9]

Heap-order property

for each node, all entries in the subtree under the node must be greater than the node.

insert



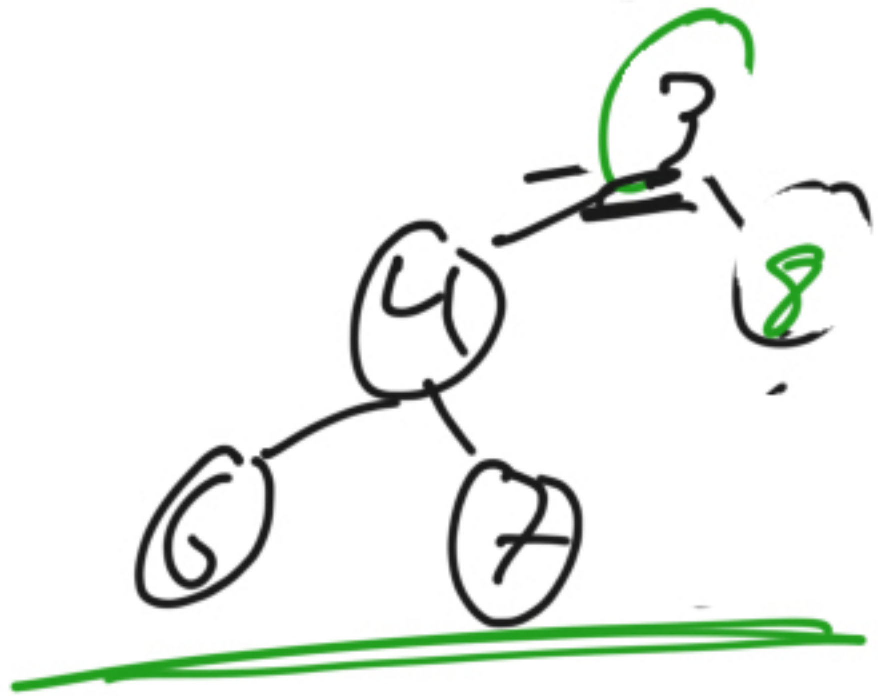


insert 3

$\text{left\_child}(i) = 2i$

$\text{right\_child}(i) = 2i+1$

$\text{Parent}(i) = \left\lfloor \frac{i}{2} \right\rfloor$



	1	2	3	4	5	6
None	1	4	3	6	7	

deleteMin() → 1

Move last entry to root

swap that value with the smaller of its children until both children are greater

height of complete binary tree
with N nodes is $O(\log N)$

so insert and deleteMin run in $O(\log N)$

Heap Sort:

insert N entries in $O(N \cdot \log N)$ time.

Then deleteMin until empty in $O(N \log N)$.

write result to a new list.

Total: $O(N \log N)$