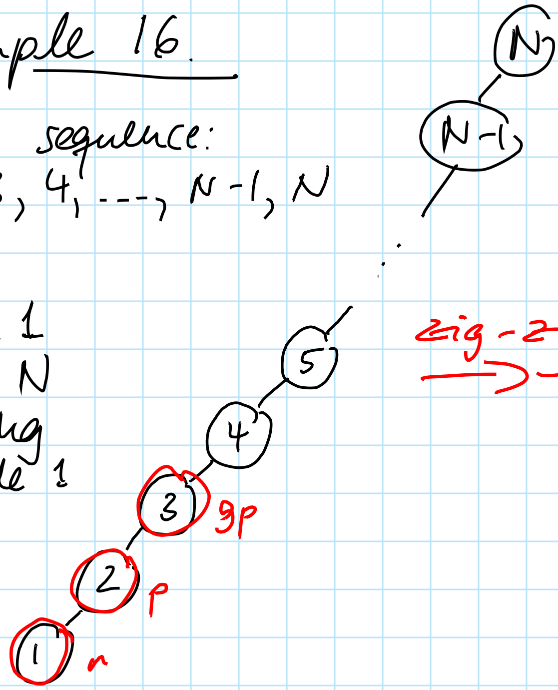
lecture 8 - Sep 21

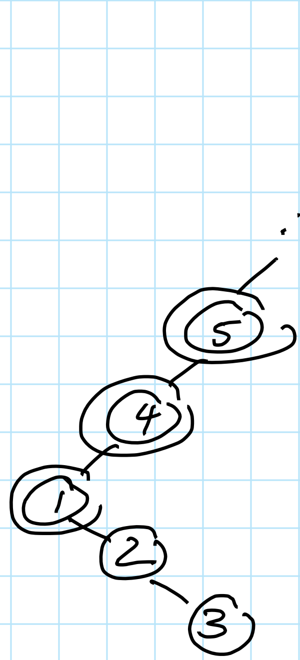
Example 16.

access sequence:
$1, 2, 3, 4, \dots, N-1, N$

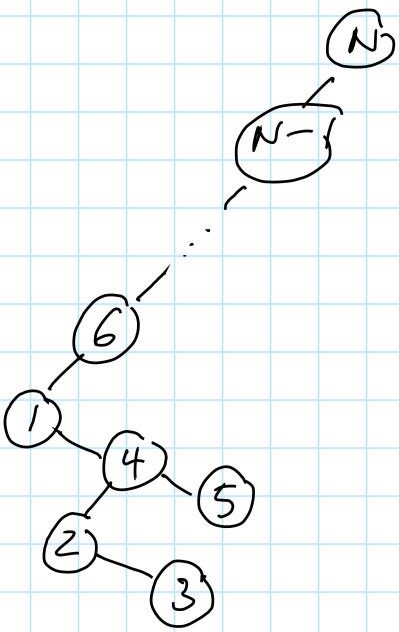
access: 1
cost: N
then: splaying at node 1



zig-zig $\longrightarrow$

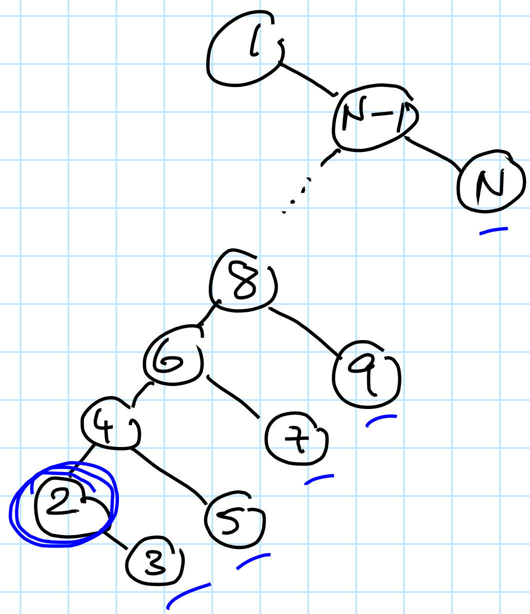


zig-zig $\longrightarrow$



assume N odd

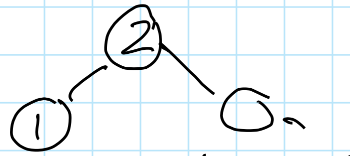
a total of $\lfloor \frac{N}{2} \rfloor$ zig-zig rotations $\longrightarrow$

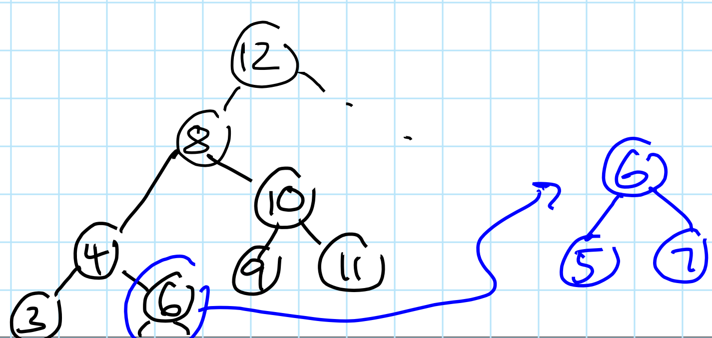


access 2:
cost $\frac{N}{2}$
then splaying at 2

$\rightsquigarrow$

$\frac{N}{4}$ rotations





access 3: cost $\frac{N}{4}$ then splaying at 3

access 4 cost $\frac{N}{8}$

in total:

access	N	$\frac{N}{2}$	$\frac{N}{4}$	$\frac{N}{8}$	
splay	$\frac{N}{2}$	$\frac{N}{4}$	$\frac{N}{8}$	$\frac{N}{16}$	...

$$\rightsquigarrow \quad N + 2\frac{N}{2} + 2\frac{N}{4} + 2\frac{N}{8} + \dots$$

$$= 2N + \underbrace{\frac{N}{2} + \frac{N}{4} + \frac{N}{8} + \dots}_{\leq N}$$

$$\leq 3N = \Theta(N)$$

for a sequence of N operations

$\rightsquigarrow$ amortized cost of $\Theta(1)$ (for this particular sequence)

Operations on Splay Trees

- search k
  1) as in BST
  2) splay at node k
- insert k
  1) as in BST
  2) splay at node k
- delete k
  1) splay at node k
  2) delete root ($\to k$) as for BST

▷ easier to implement than AVL trees,

▷ saves space (no balance information needs to be stored)

Analysis : the running time of all operations is dominated by cost of splaying

NOTE : In the worst case, the running time of a single splay operation in a splay tree with N nodes is $\Theta(N)$

Theorem 5. The amortized time to splay a node in a splay tree with N nodes is $O(\log(N))$

Corollary 2. The amortized running time of any standard splay tree op. for a splay tree with N nodes is $O(\log(N))$

2. PRIORITY QUEUES

queue : FIFO, e.g., most printers
line-up in a movie theatre

priority queue: "most important" dequeued first, e.g.,
clever printers
OS w/ multiple users
line-up in a medical clinic

2.1 The ADT Priority Queue

operations
- insert (cf. enqueue)
- deleteMin (cf. dequeue)
  removes the highest priority (= minimal) element

applications :
- OS
- external sorting
- greedy algorithms

Implementation:

- linked list?

  insert at front $\Theta(1)$, delete min $\Theta(N)$

  or keep sorted, then delete min $\Theta(1)$, but insert $\Theta(N)$

- BST? overkill: we do not need all the properties

⇝ better: a very simple DS that supports "insert" and "delete min" with $T_{worst}(N) = O(\log(N))$

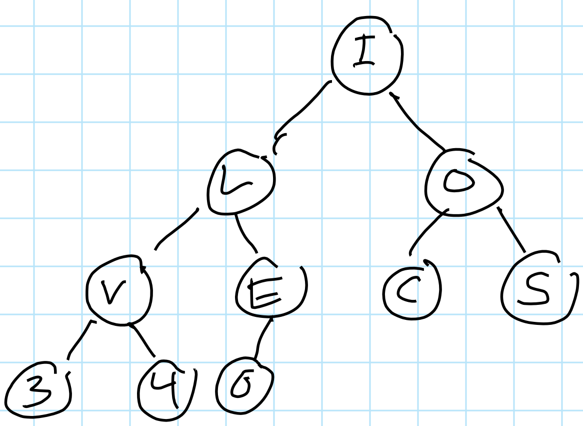
(even $T_{avg}(N) = O(1)$ for insertion!!!)

2.2 (Binary) Heap

"Definition" 12 A complete BT is a BT with the following properties:

(1) If there is a node at depth d, $d \geq 1$, then there are 2^{d-1} nodes at depth $d-1$, i.e., level $d-1$ is "full".

(2) The bottom level of the tree is filled from left to right.



⇝ complete BTs can be easily stored as arrays!

0	1	2	3	4	5	6	7	8	9	10	11	12	
	I	L	O	V	E	C	S	3	4	O			....

node n in position i

→ left child of n in pos. $2*i$

→ right child $2*i+1$

→ parent of n $\lfloor i/2 \rfloor$

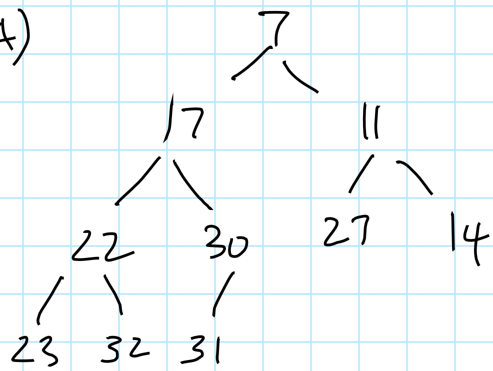
⇝ no links needed! fast traversal operations

Definition 5. A (binary) heap is a complete BT in which, for every node n except for the root, is at least as large as the entry at its parent

Fact
- Every (subtree of a) heap has its smallest entry at the root.
- Every complete subtree of a heap is a heap.

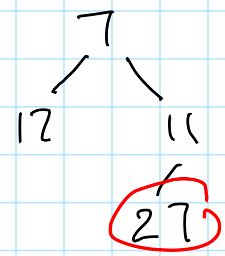
Example 17.

(A)



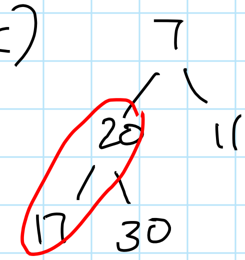
heap ✓

(B)



not a heap (not a complete BT)

(C)



not a heap