Vlad Bogdan-Tudor, 917
VladB

1) search → AVL Tree: $O(\log n)$

→ Binary heap: $O(n)$

⇒ AVL is better for search

size → AVL Tree: $\Theta(1)$

→ Binary heap: $\Theta(1)$

} We can keep a "size" variable

remove a given element → AVL Tree: $O(\log n)$

$\Theta(\log n)$ only if it has 2 descendants and we have to find successor/predecessor

→ Binary heap: $O(h)$, which is $O(n)$

We have to apply bubble-down

⇒ AVL is better for remove

isFull → AVL Tree: $\Theta(1)$

→ Binary heap: $\Theta(1)$

} if we want a fixed size priority queue we can keep a variable for capacity and another one for size and compare them.

Vlad Bogdan-Tudor, 917
VladB

2) Print leaves of AVL tree in sorted order - worst case.

→ $\Theta(n)$

We can apply an in-order traversal and we only print the values of the nodes which do <u>not</u> have any descendants.

Vlad Bogdan-Tudor, 917
VlosB

3)

0	1	2	3	4	5	6	7	8	9	10
55	52	73	14			6	28	39	20	32

$h(k,i) = (k \% 11 + i) \% 11$, $i = \overline{0,10}$

An element is a candidate for being the first inserted if its position was computed with $i=0$ in the hash function.

$h(55,0) = 0$
55 is on pos 0 } ⇒ can be first ✓

$h(52,0) = 8$
52 is on pos 4 } ⇒ can't be first ×

$h(73,0) = 7$
73 is on pos 2 } ⇒ can't be first ×

$h(14,0) = 3$
14 is on pos 3 } ⇒ can be first ✓

$h(6,0) = 6$
6 is on pos 6 } ⇒ can be first ✓

$h(28,0) = 6$
28 is on pos 7 } ⇒ can't be first ×

Vlad Bogdan-Tudor, 917
VloBT

$h(39, 0) = 6$
39 is on pos 8 } ⇒ can't be first ✗

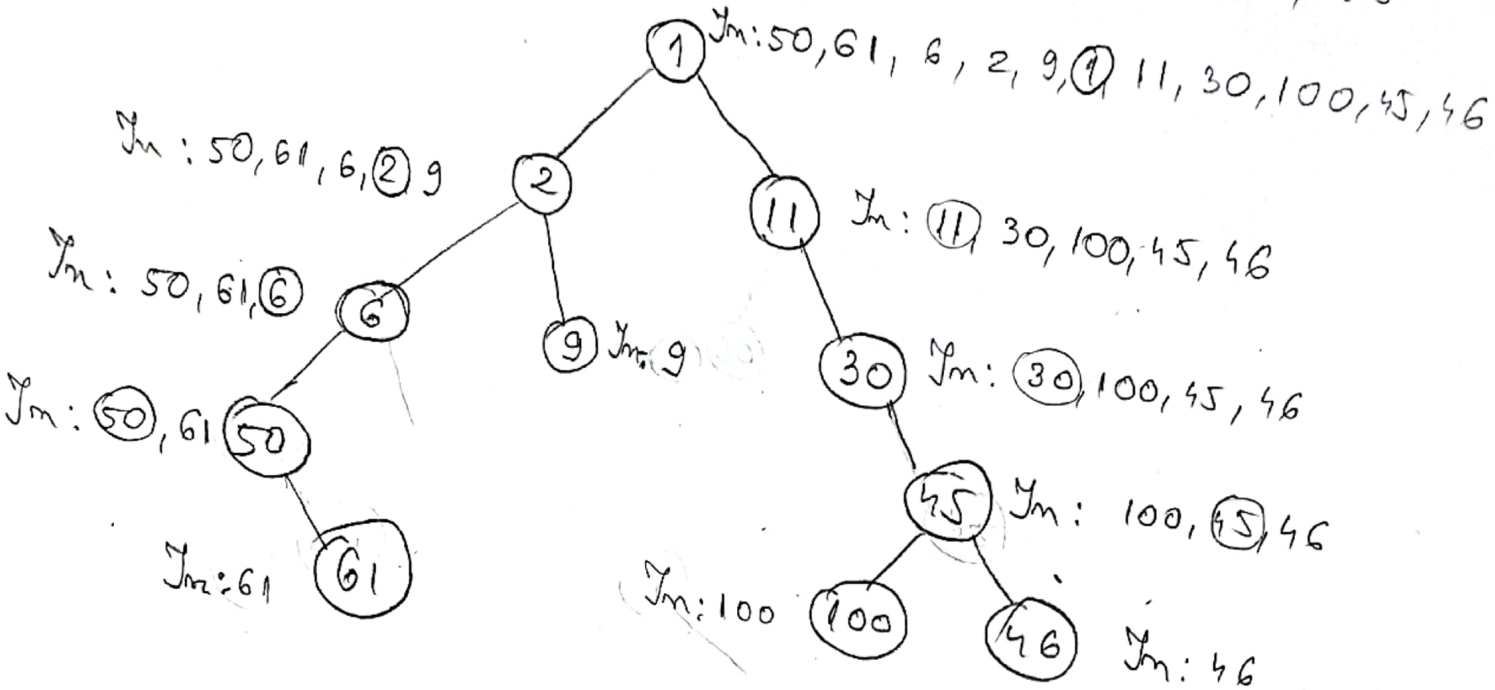
$h(20, 0) = 9$
20 is on pos 9 } ⇒ can be first ✓

$h(32, 0) = 10$
32 is on pos 10 } ⇒ can be first ✓

⇒ <u>55, 14, 6, 20, 32</u> can be first

Vlad Bogdan-Tudor, 917

4) Inorder: 50, 61, 6, 2, 9, 1, 11, 30, 100, 45, 46



It has min-heap property.

Post-order (left, right, visit): 61, 50, 6, 9, 2, 100, 46, 45, 30, 11, 1