Binary Heap: Intuition and Implementation

Course on Basic Data Structures (C++)

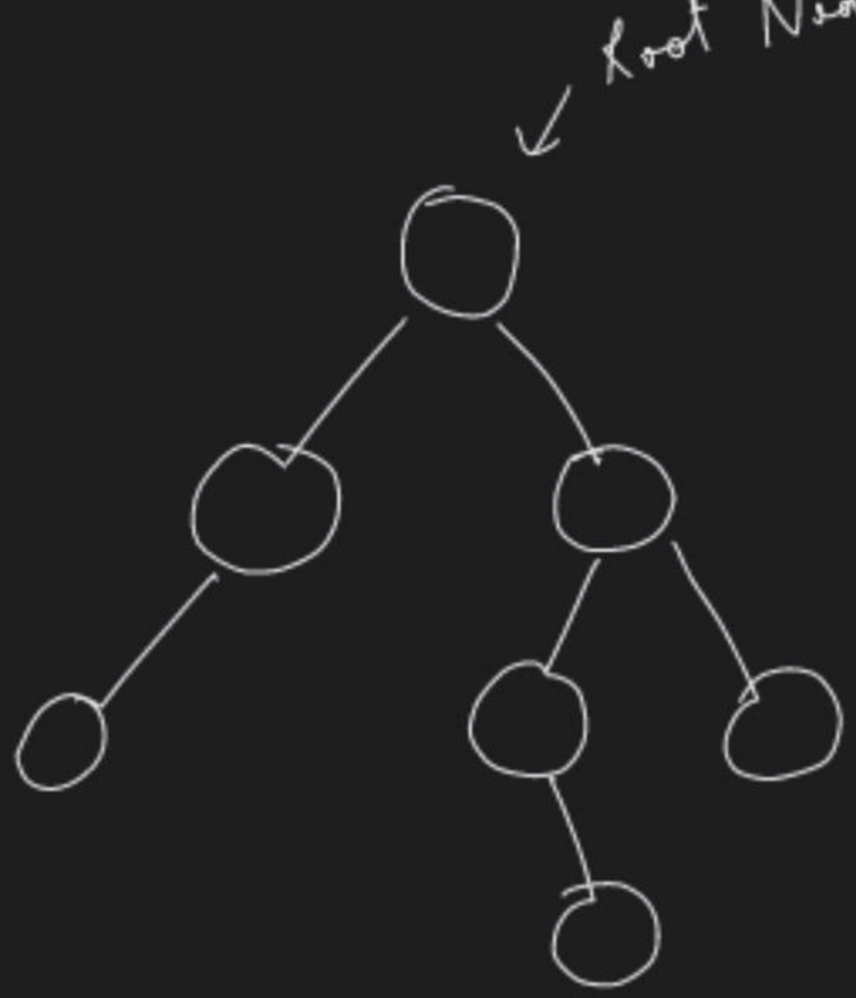
Pulkit Chhabra • Lesson 17 • Feb 18, 2021

Linear { Vector, Linked List, Stack, Queue }

HashMap

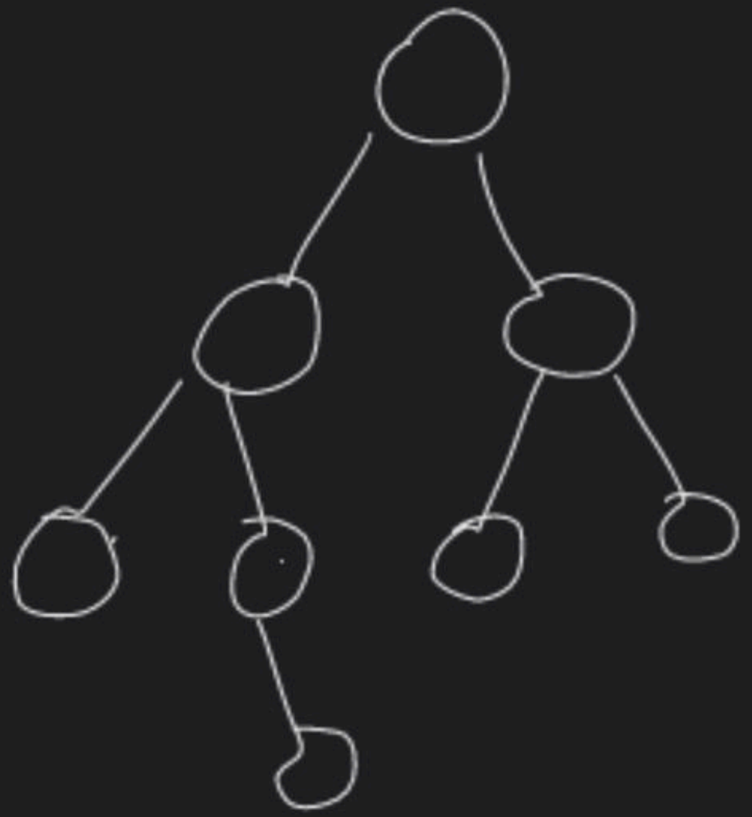
Non-Linear { Binary Tree, Binary Heap }

Binary Tree



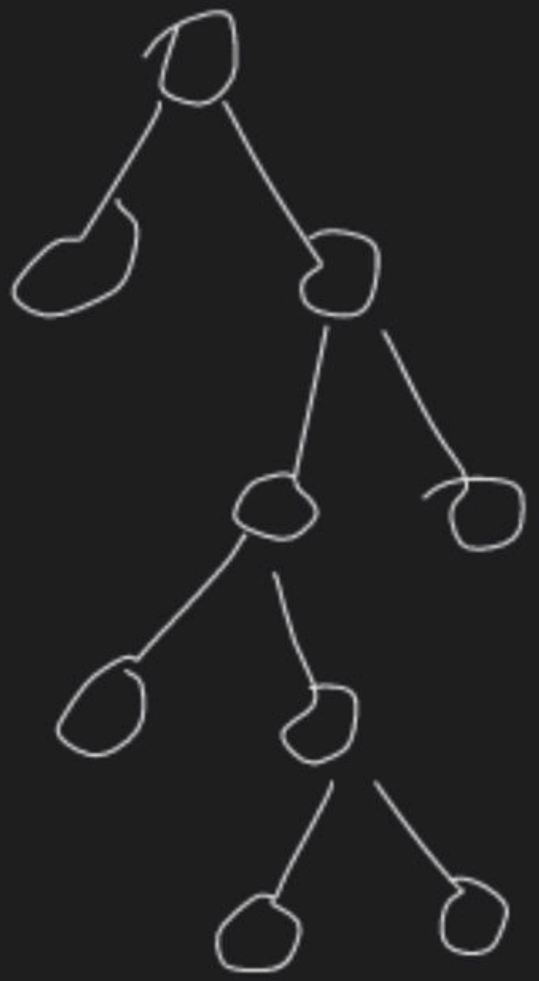
No. of children ≤ 2

Balanced Bin. Tree



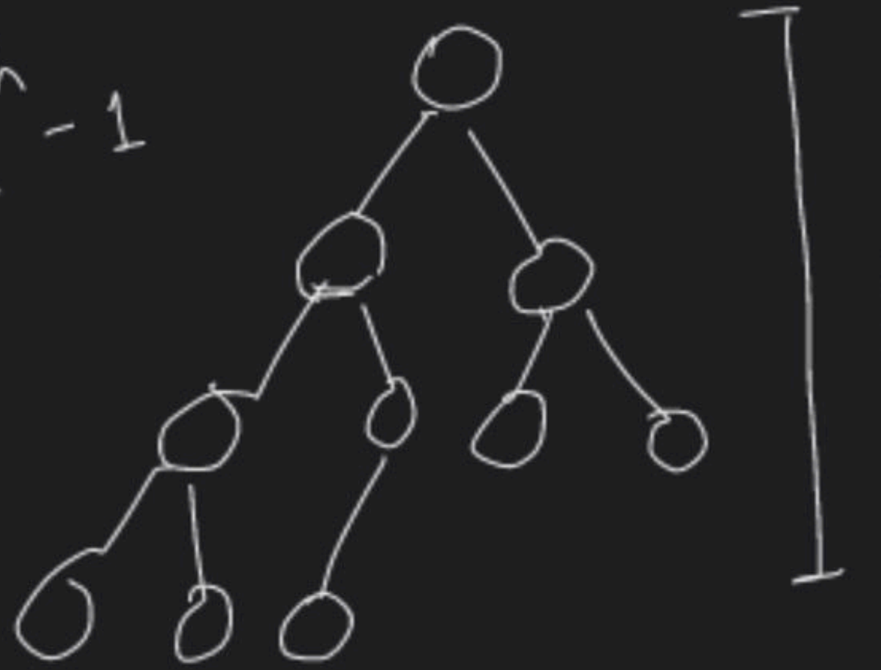
Full Binary Tree

Every has either 2 children or 0



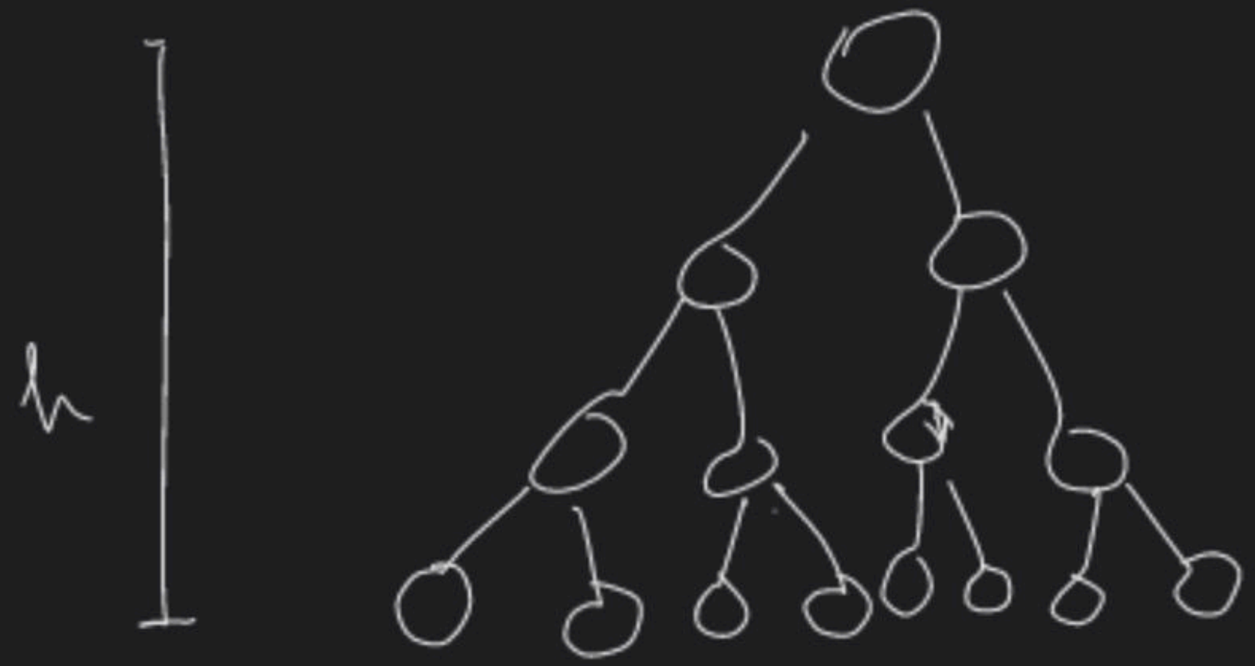
Complete Binary Tree

$$2^{h-1} - 1 < N \leq 2^{h} - 1$$



Till 2nd last level, all slots are full, in the last level, some prefix of slots are filled, after that slots may be empty

Perfect Binary Tree



$$N = 2^h - 1$$

$$\text{no\_of\_nodes}(l) = 2^{l-1}$$

$$(0 \le l < h)$$

$$N = 2^0 + 2^1 + 2^2 \cdots\cdots 2^{h-1} \Rightarrow 2^h - 1$$

1) Binary Search Tree

max(left_subt) < node.val < min(right_subt)

2) Binary Heap (Min)

→ Complete Binary Tree

→ node.val ≤ min(left.val, right.val)

Design a DS which can support following operations.

1.) Insert

2) GetMin

3) DeleteMin

vector

1) Insert ⇒ O(1)

2) Get Min / Delete Min

⇒ O(N)

Vector sorted in reverse.

1) Insert ⇒ O(N)

2) Get Min / Delete Min

↳ O(1)

L.L.

1) O(N) get/Delete Min

2) O(1) Insert

Sorted L.L.

1) O(N) Insert

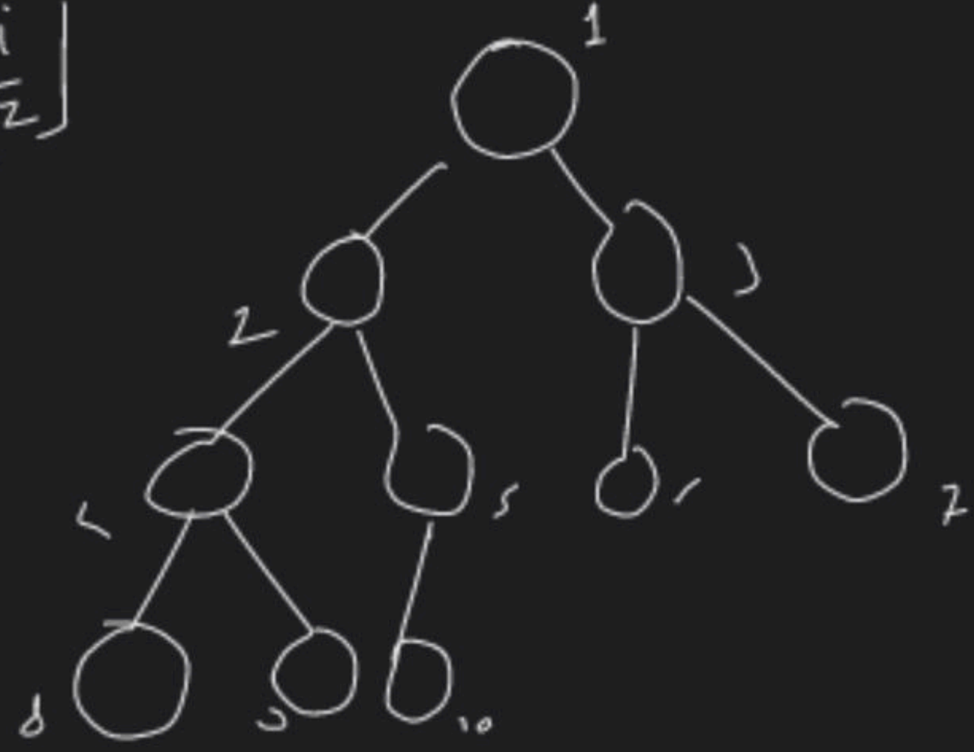
2) O(1) get/Delete Min

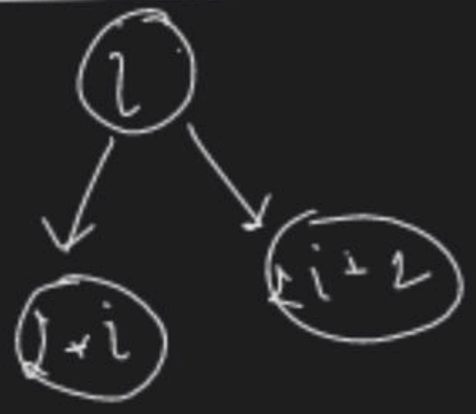
How to get down all 3 operations to $O(\log N)$?

Tree

{ class node
↳ int val
node * left, * right }

Complete BT parent (i) → $\lfloor \frac{i}{2} \rfloor$





$2*i > n$
$\Rightarrow i > n/2$
↓
no left child

$2i+1 > n$
$i > \frac{n-1}{2}$
↓
no right child.

Min Heap

[x, 9, 10, 21, 18, 13, 30, 47, 102, 19, 32]

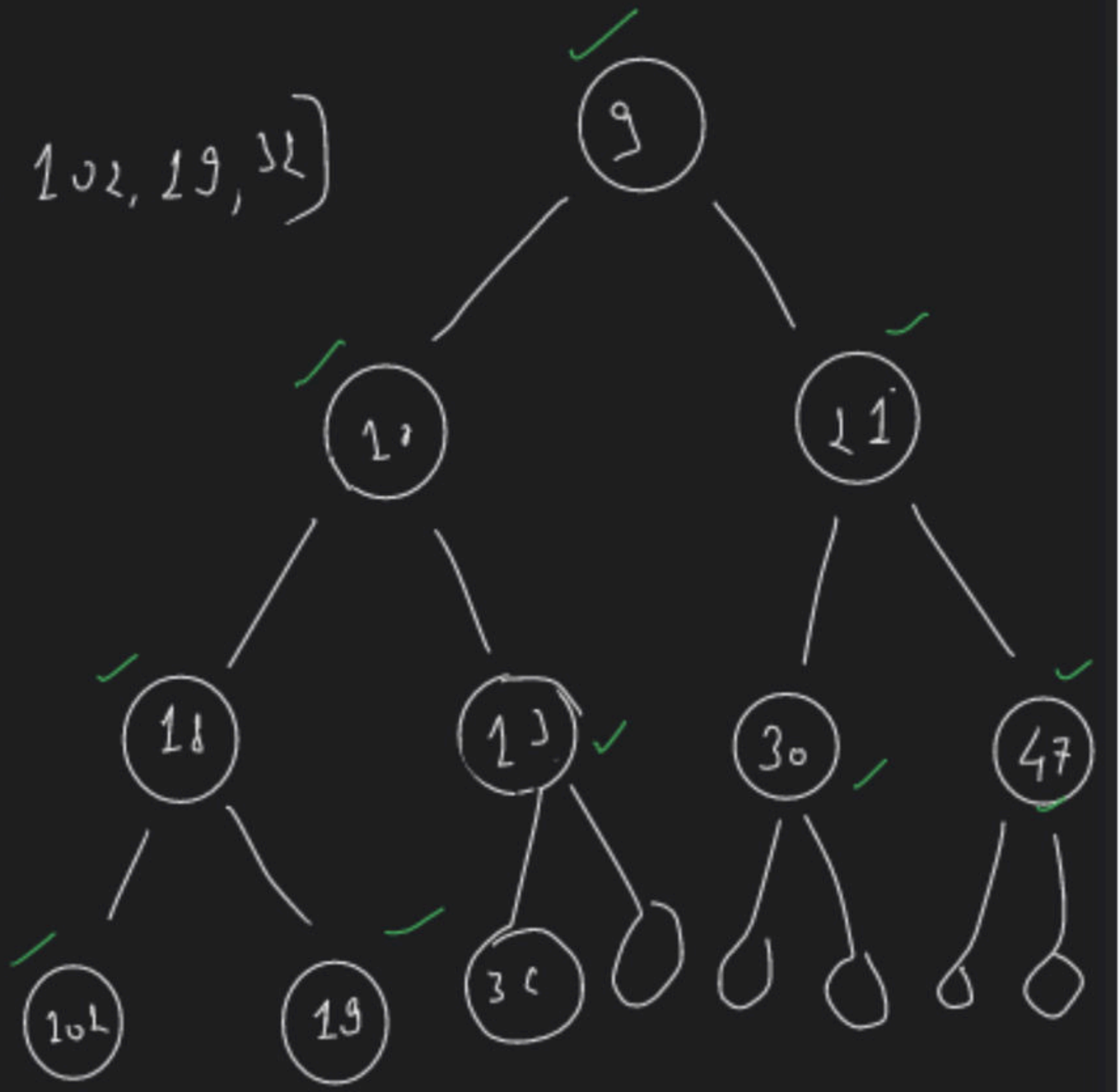
1.) Insert ⟶ O(log N)

2.) Delete Min → O(log N)

3.) Get Min ⟶ O(1)

4) Sift Up ⟶ O(log N)

5) Sift Down → O(log N)



$$\sum_{h=1}^{\log N} \frac{n}{2^h} \times h$$

$$\Rightarrow n * \sum_{h=2}^{\infty} h * \left(\frac{1}{2}\right)^h$$

$$T(N) = O(N)$$

$$\sum_{h=1}^{\infty} \frac{1}{2^h} = 1$$

$$\sum_{h=2}^{\infty} x^h = \frac{1}{2-x}$$

$$\sum_{h=0}^{\infty} h x^h = \frac{x}{(1-x)^2}$$

$\Rightarrow (1/2)/(1/2) = 1.$

$$T(N) = \sum_{h=1}^{\log N} \frac{N}{2^h} * h$$

$$N * \sum_{h=1}^{\infty} h * \left(\frac{1}{2}\right)^h$$

$$T(N) = O(N)$$

$$\sum_{h=1}^{\infty} x^h = \frac{1}{1-x}$$

$$\sum_{h=1}^{\infty} h x^h = \frac{x}{(1-x)^2}$$

$$\sum_{h=1}^{\infty} h * \left(\frac{1}{2}\right)^h = \frac{1/2}{1/4}$$

$$\Rightarrow 2$$

Heaps: Intuition and Implementation

With Pulkit Chhabra

Let's crack Competitive Programming together!

1. What is the time complexity of building a heap from an unordered array?

A. O(1)
B. O(log n)
C. O(n)
D. O(n log n)

1. What is the time complexity of building a heap from an unordered array?

A. O(1)
B. O(log n)
C. O(n)
D. O(n log n)

2. A heap data structure is represented by a ____ binary tree?

A. degenerate
B. perfect
C. full
D. complete

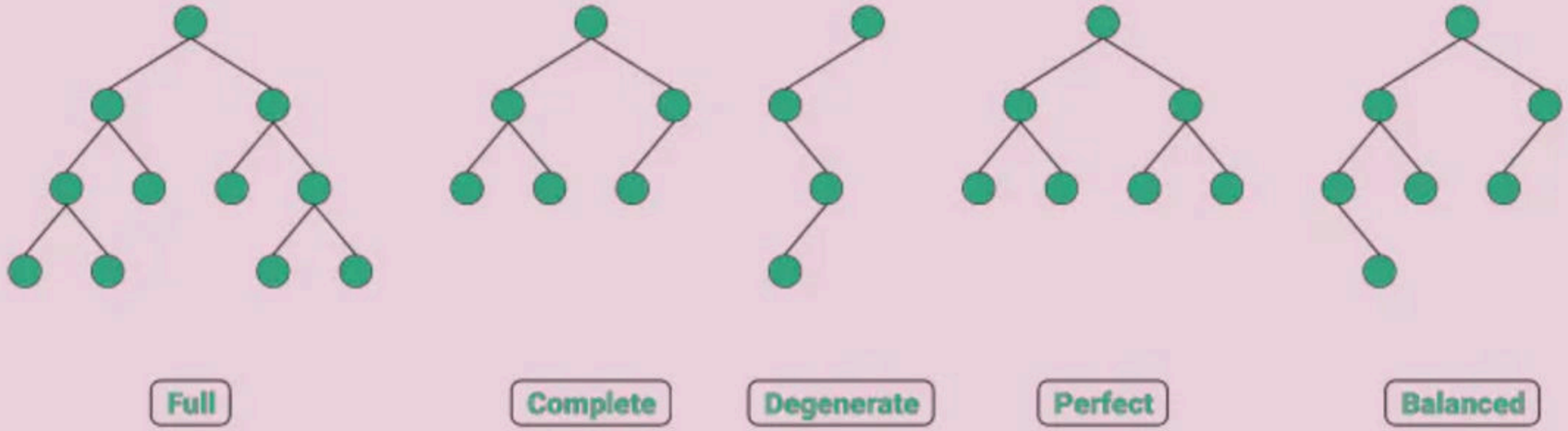


Image courtesy: Anand K Parmar

2. A heap data structure is represented by a ____ binary tree?

A. degenerate
B. perfect
C. full
D. complete

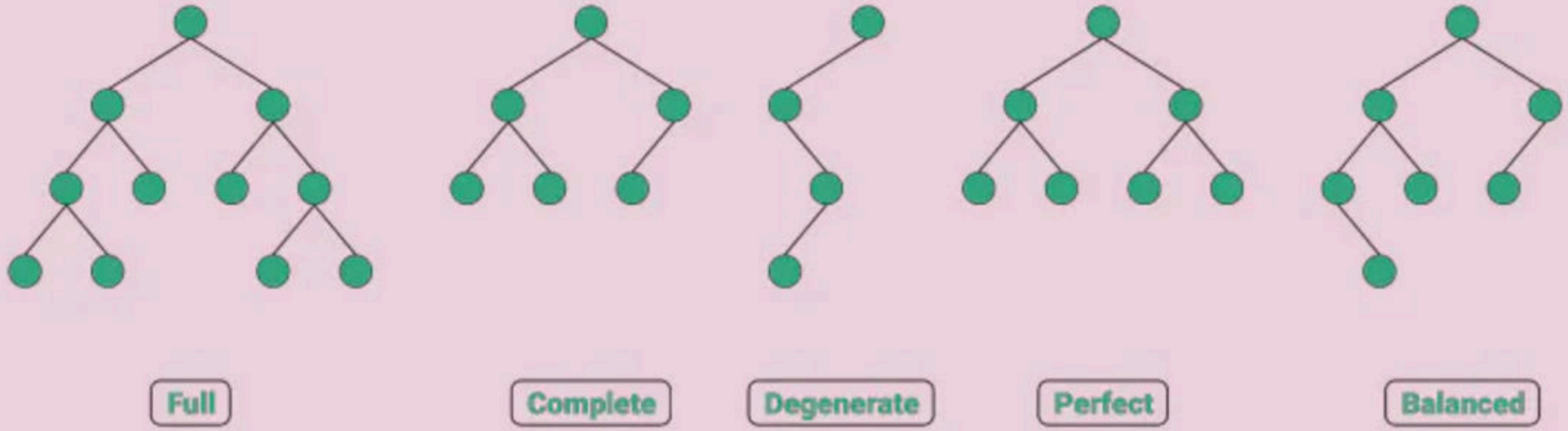


Image courtesy: Anand K Parmar

3. A max-heap of ***n*** elements, would have ___ leaf nodes?

A. n - 1
B. ceil(n/2)
C. floor(n/2)
D. floor(log N)

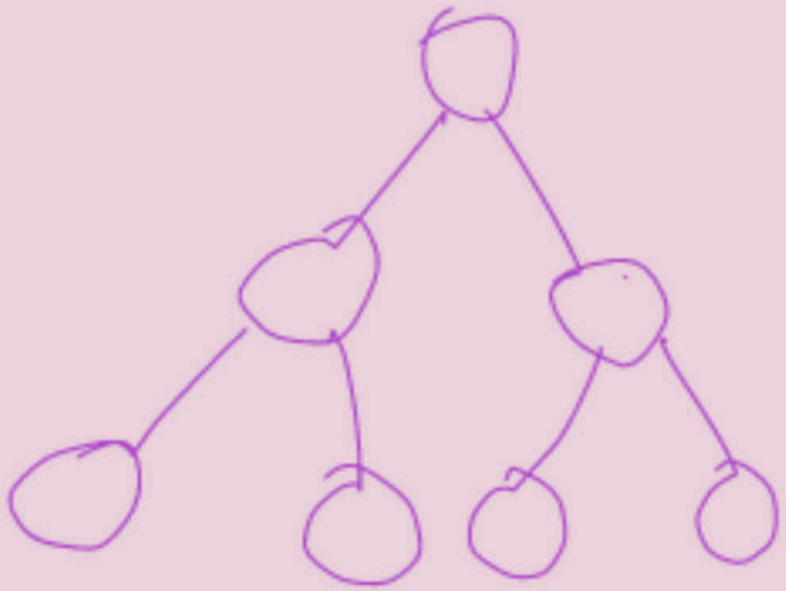
3. A max-heap of ***n*** elements, would have ___ leaf nodes?

A. n - 1

B. ceil(n/2)

C. floor(n/2)

D. floor(log N)



We can still optimize by ignoring the non-leaf nodes.

4. What is the time complexity of finding the min-element in max-heap?

A. O(1)
B. O(log n)
C. O(n)
D. O(n log n)

4. What is the time complexity of finding the min-element in max-heap?

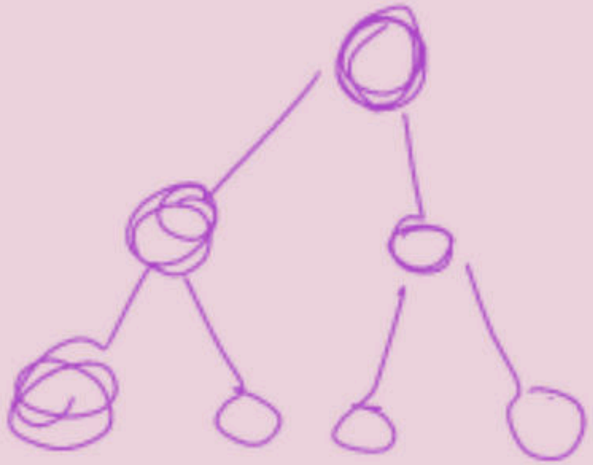
A. O(1)
B. O(log n)
C. O(n)
D. O(n log n)

We can still optimize by ignoring the non-leaf nodes.

5. Assume we have a max-heap of size ***n*** and we wish to add ***n*** more elements to it, what is the order of time complexity?

A. $O(n)$
B. $O(n \log n)$
C. $O(n^2)$
D. $O(n^2 \log n)$

5. Assume we have a max-heap of size ***n*** and we wish to add ***n*** more elements to it, what is the order of time complexity?



A. O(n)
B. O(n log n)
C. $O(n^2)$
D. $O(n^2 \log n)$

Create a new array with ***2n*** elements, use the *buildheap* operation with time complexity ***O(n)***.

Thank You

Let’s crack Competitive Programming together!