Heaps

Prepared by Mahdi Ghamkhari

Priority Queues Based on Linked Lists

Priority Queues implemented by Linked Lists:

Fast-enQueueing: enQueue is O(n) deQueue is O(1)
Fast-deQueueing: enQueue is O(1) deQueue is O(n)

Priority Queues

Can we have a priority Queue for which enQueue and deQueue have a same time complexity?

Priority Queues

Can we have a priority Queue for which enQueue and deQueue have a same time complexity?

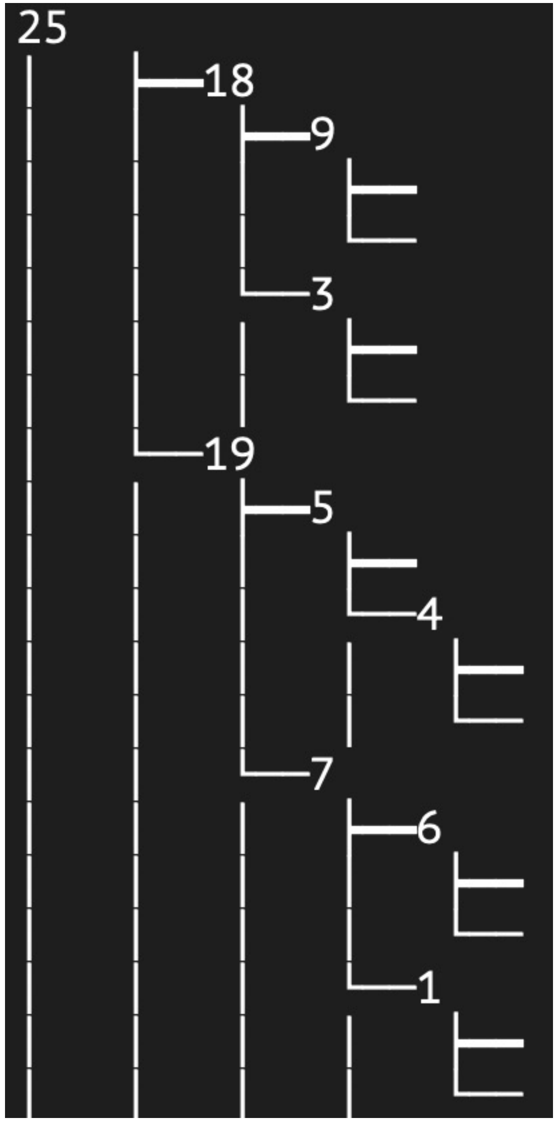
Yes! If we implement priority queues based on Binary Trees

Such a priority queues is called a Heap

Add() has a time complexity of log(n)
Delete() has a time complexity of log(n)

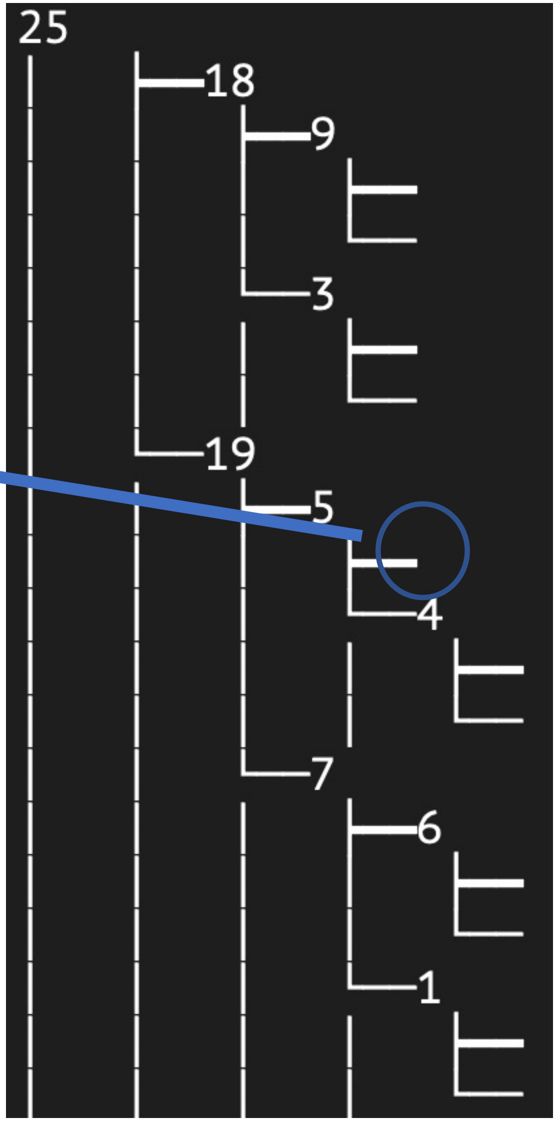
A Heap is a:

- Complete binary tree
  - All levels are full, except maybe the last level, which is filled in bottom-to-top
- At each node:
  - the parent has higher priority/value than its children



Add(12)

Since heap is a complete binary tree, a new node should be added right here



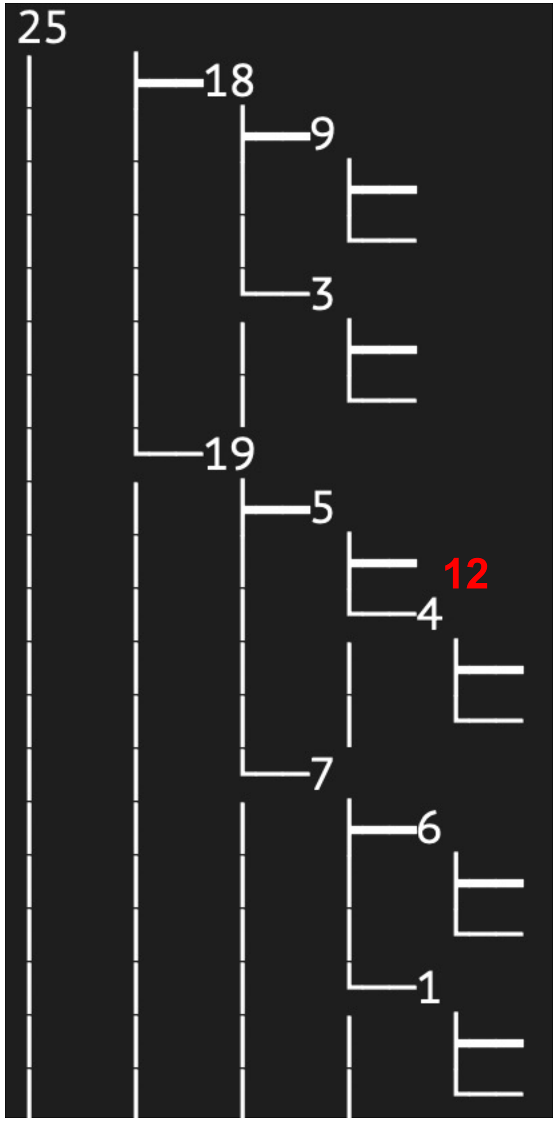
Add(12)

Adding 12 at the last level of the

Binary tree disturbs the order of the nodes

5 is parent of 12

5 $<$ 12

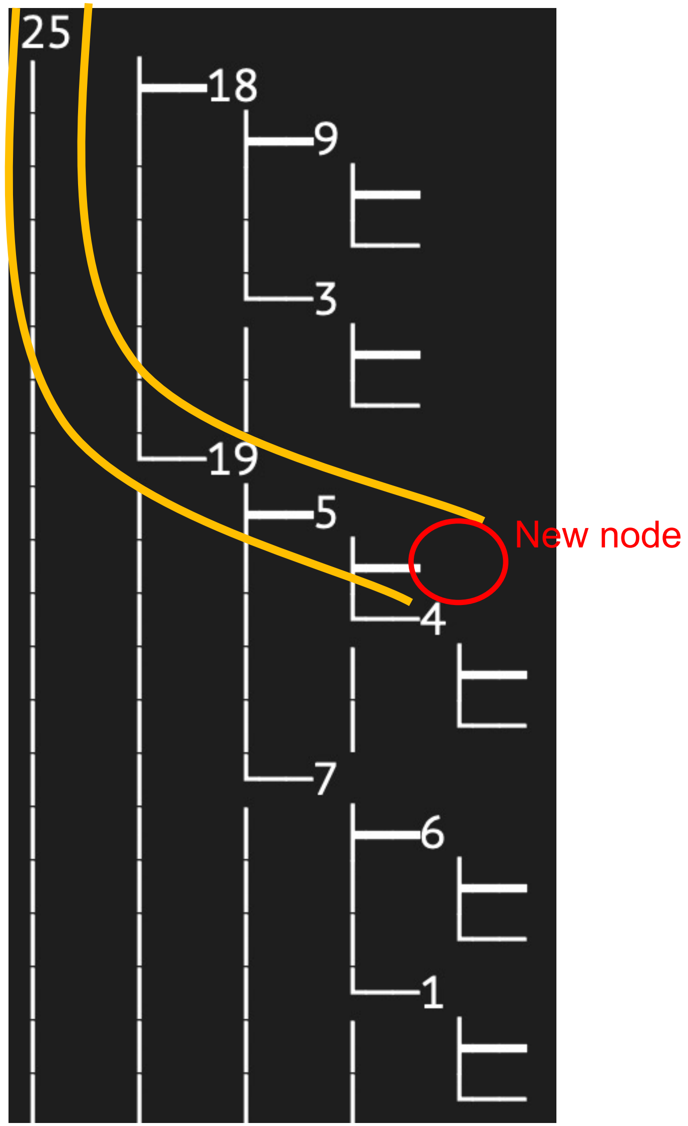


Add(12)

We need an algorithm to add new values to the heap without disturbing the order of nodes in the heap

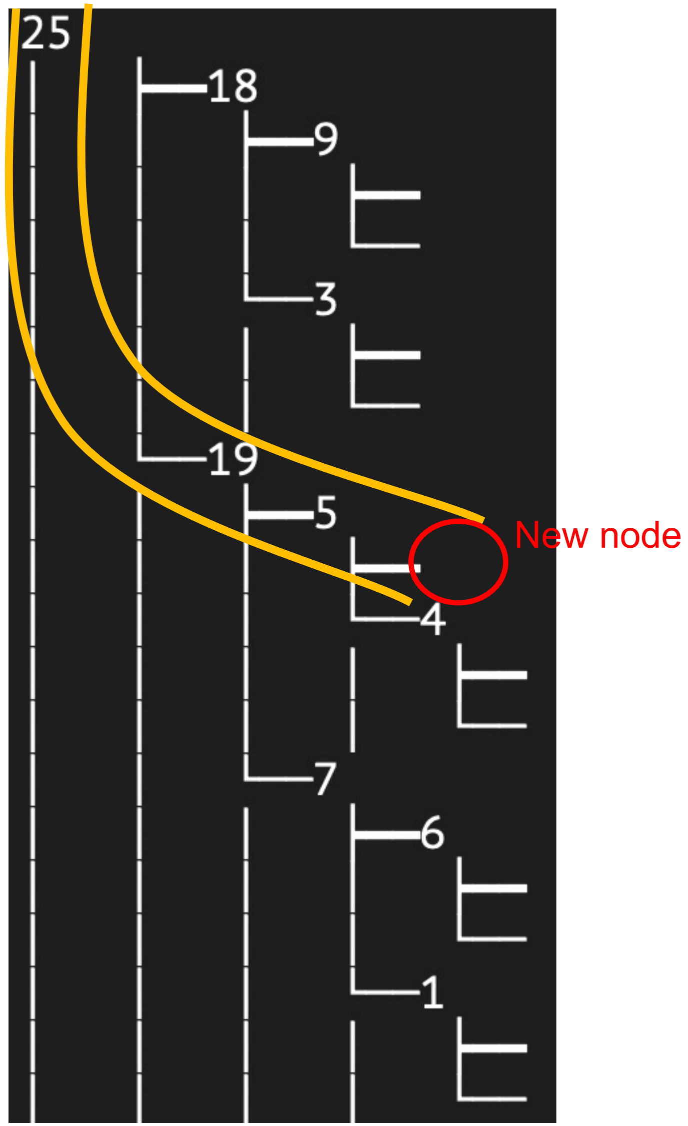
An Algorithm for Add(12)

- A heap is a binary tree
- There is one and only one route from the root to the new node
- The route is 25 → 19 → 5→ new node



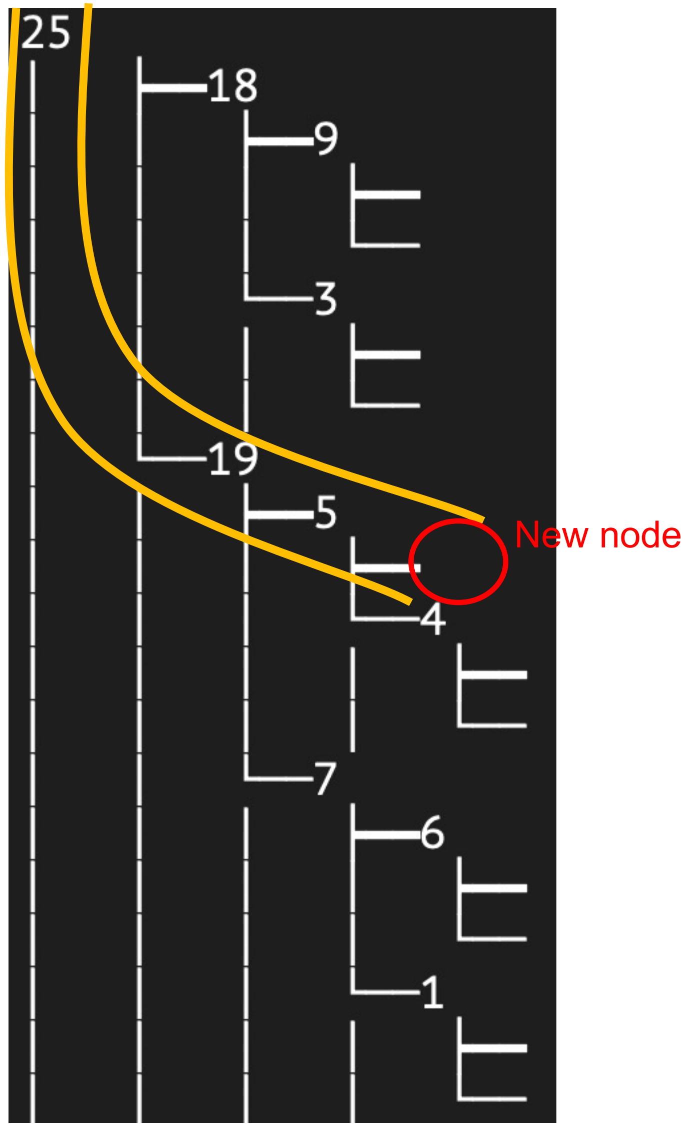
An Algorithm for Add(12)

- A heap is a binary tree
- There is <u>one and only one route</u> from the root to the new node
- The route is 25 → 19 → 5→ new node
- We start at the beginning of the route: root node
- 25 > 12



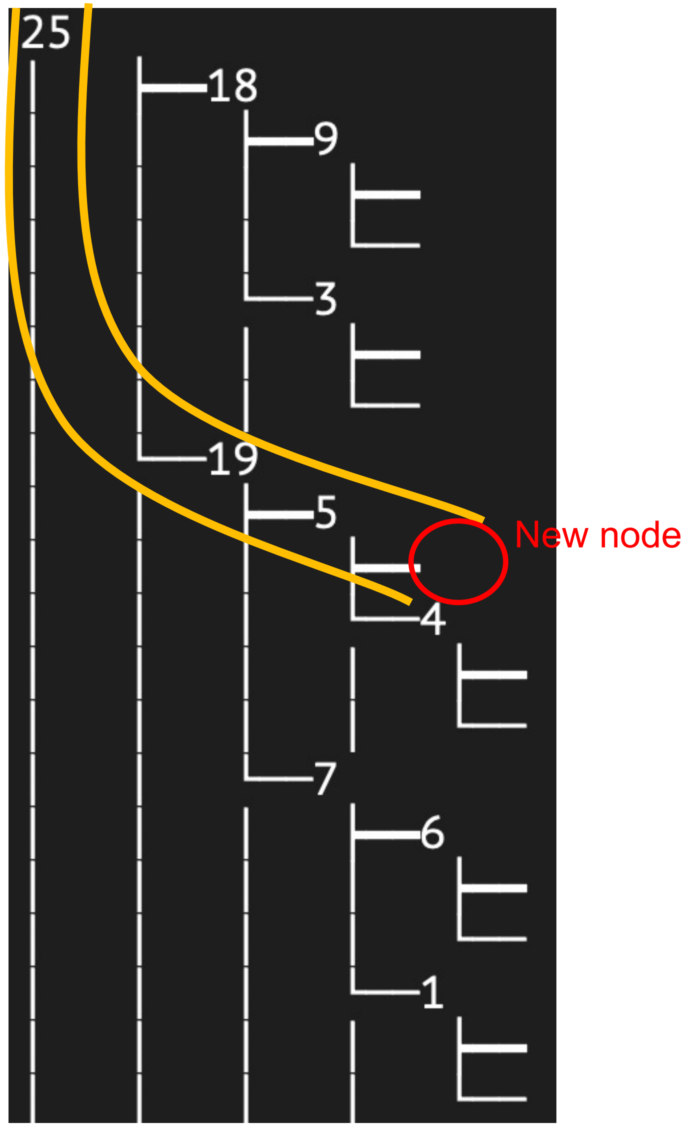
An Algorithm for Add(12)

- A heap is a binary tree
- There is <u>one and only one route</u> from the root to the new node
- The route is 25 → 19 → 5→ new node
- We start at the beginning of the route: root node
- 25 > 12
- We move to the next node of the route: 19
- 19 > 12



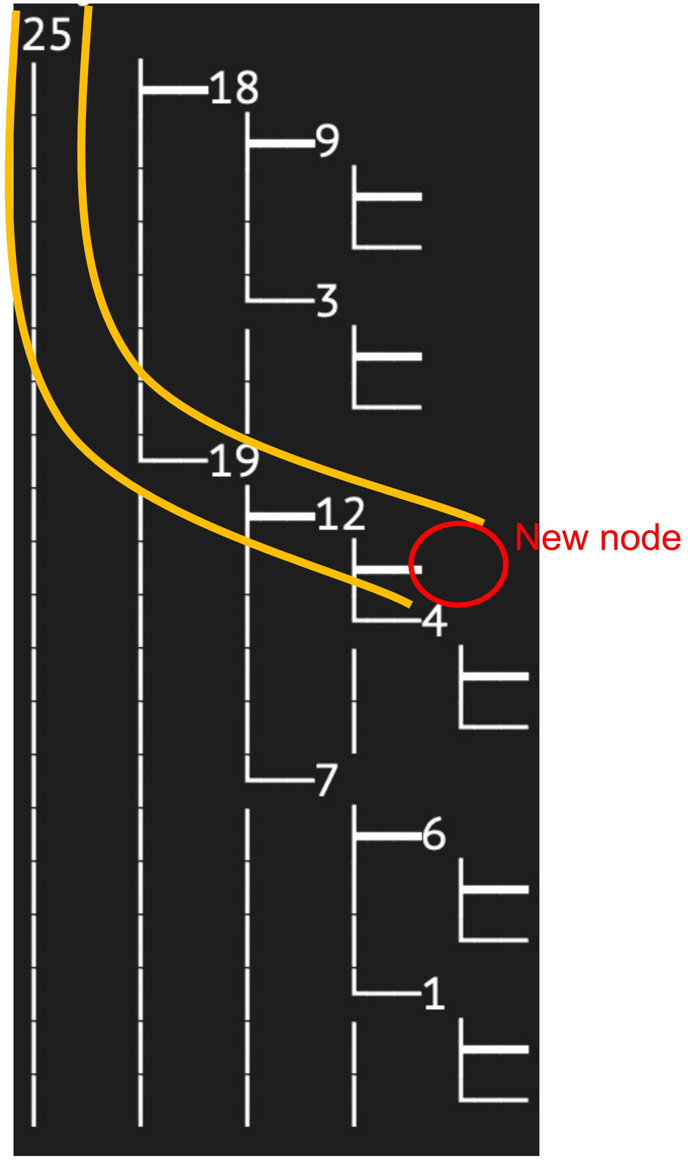
An Algorithm for Add(12)

- A heap is a binary tree
- There is <u>one and only one route</u> from the root to the new node
- The route is 25 → 19 → 5→ new node
- We start at the beginning of the route: root node
- 25 > 12
- We move to the next node of the route: 19
- 19 > 12
- We move to the next node of the route: 5
- 5 < 12 → We must take an action



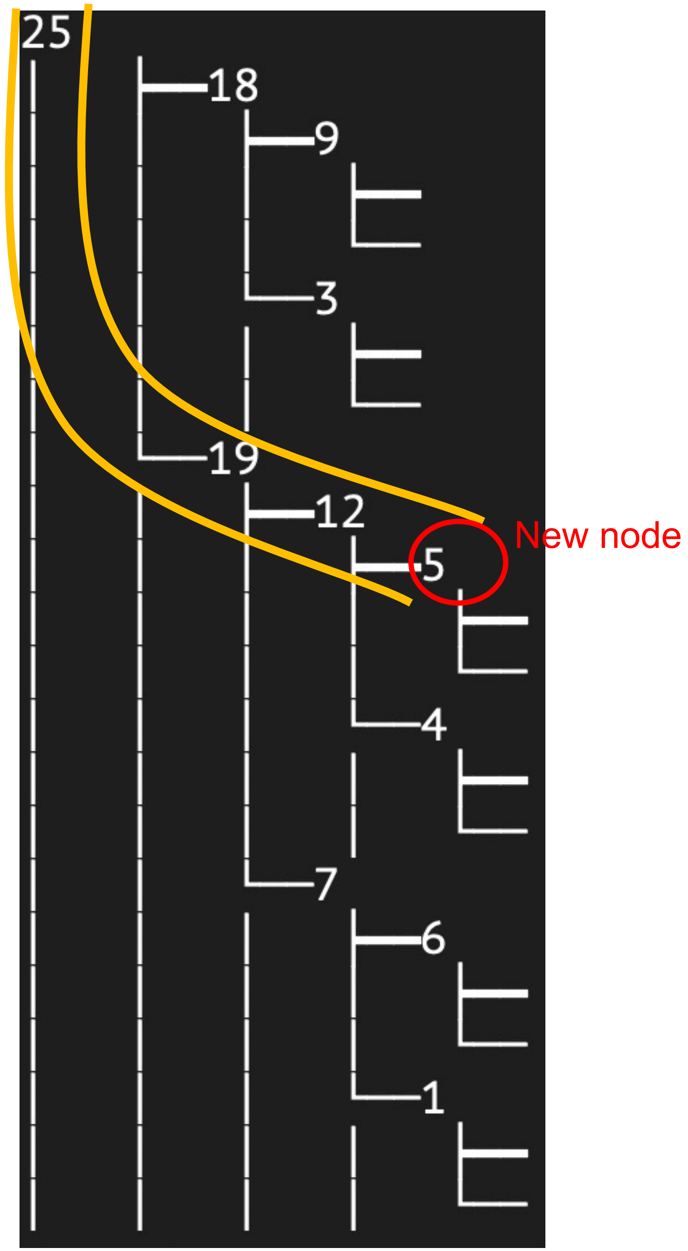
An Algorithm for Add(12)

- A heap is a binary tree
- There is one and only one route from the root to the new node
- The route is 25 → 19 → 5→ new node
- We start at the beginning of the route: root node
- 25 > 12
- We move to the next node of the route: 19
- 19 > 12
- We move to the next node of the route: 5
- 5 < 12 → We must take an action: We replace 5 with 12. Now 12 is placed in the Heap. We follow the rest of route to place 5 somewhere in the route



An Algorithm for Add(12)

- A heap is a binary tree
- There is one and only one route from the root to the new node
- The route is 25 → 19 → 5→ new node
- We start at the beginning of the route: root node
- 25 > 12
- We move to the next node of the route: 19
- 19 > 12
- We move to the next node of the route: 5
- 5 < 12 → We must take an action: We replace 5 with 12. Now 12 is placed in the Heap. We follow the rest of route to place 5 somewhere in the route
- We arrive at the new node and place 5 in the new node



Time complexity of add() algorithm

- When there are n nodes in the Heap, in average we have to traverse a route of log(n) nodes to add a new node to the heap:
- Time complexity of add() method is O(log(n))

Duplicate Values

- We can have duplicate values in a heap.
- The codes we see in the lab allow duplicate values in the heap.

How to Find the Route

The path from root node to the new node is:

25 → 19 → 5 → new node

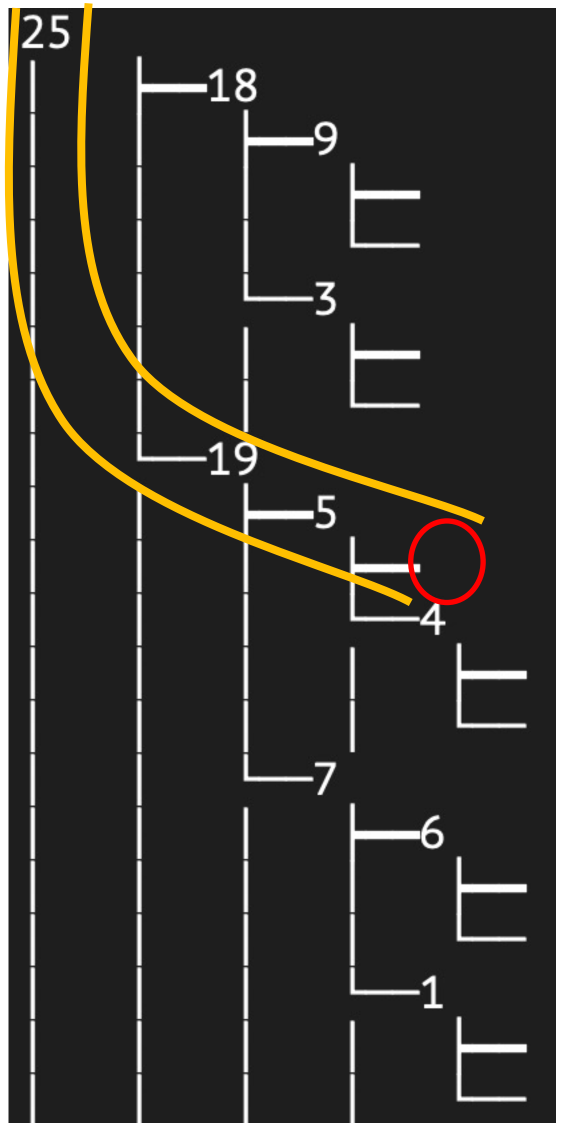
Starting from the root:

Level 1: DownWard

Level 2: UpWard

Level 3: Upward

How can we find the above directions/route?



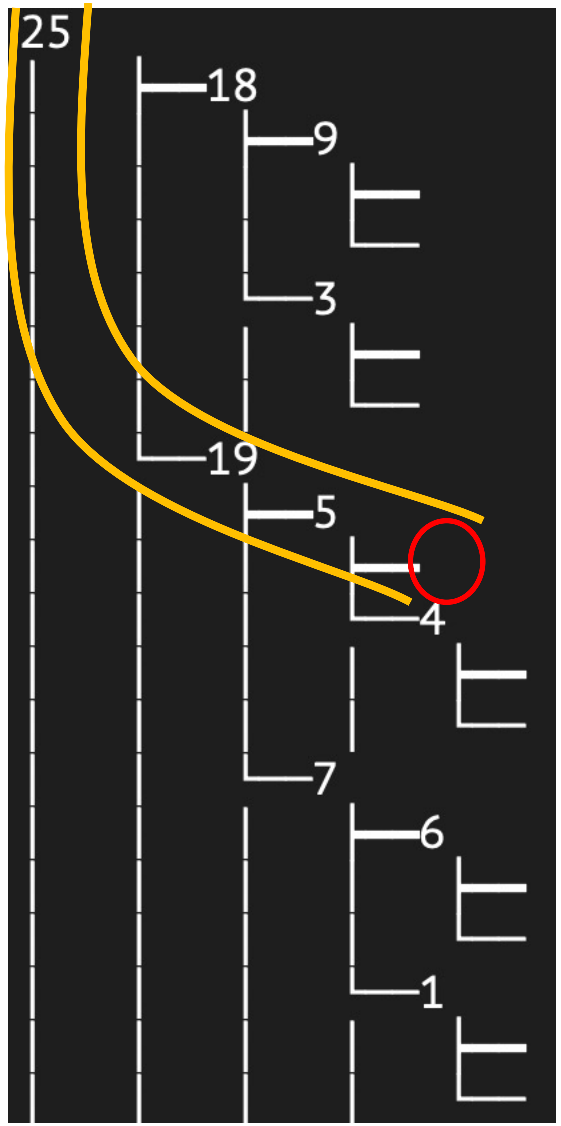
How to Find the Route

Size = 10

New node is at the Position = Size + 1 = 11

Public String [] route (int Position)

[”NON”, “DownWard”, “Upward”, “Upward”] = route (11)



How to Find the Route

```
public String[] route(int Position)

{
    String BinaryString = Integer.toBinaryString(Position);

    String[] myroute = new String[BinaryString.length()];

    myroute[0] = "NON";

    for (int i = 1; i < BinaryString.length(); i++) {
        if (BinaryString.charAt(i) == '0')
            myroute[i] = "DownWard";

        if (BinaryString.charAt(i) == '1')
            myroute[i] = "UpWard";
    }

    return myroute;

}
```

How to Find the Route

- The idea behind this algorithm is:

11 → Binary format → 1011 → Setting aside the first digit → 011

→ Converting zeros to “DownWard” and ones to “UpWard” →

→ “DownWard”, “UpWard”, “UpWard”

Time Complexity of the route() Method

- Time complexity of the route method is log(n).