

ACOL106: Data Structures and Algorithms

Ragesh Jaiswal, IIT Delhi

Data Structures

Heaps and Priority Queues

- **Priority Queue** ADT supports the following operations:
 - `insert( $k, v$ )`: Add an entry with **key** k and **value** v
 - `min()`: Returns (but does not remove) an entry (k, v) having smallest key; it returns null if there are no entries.
 - `removeMin()`: Removes and returns the entry (k, v) having smallest key; it returns null if there are no entries.
 - `size()`: returns the number of entries.
 - `isEmpty()`: returns true if there are no entries, else returns false.
- There is a data structure called **heap** where the running time of the operations are:
 - `insert( $k, v$ )`: $O(\log n)$
 - `min()`: $O(1)$
 - `removeMin()`: $O(\log n)$

Data Structures

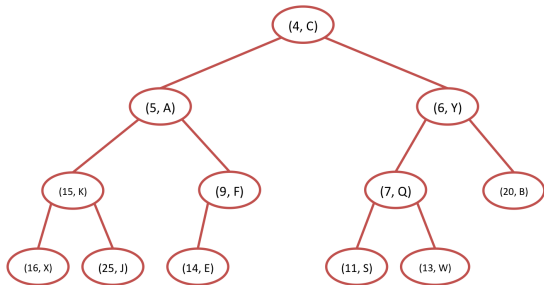
Heaps and Priority Queues

- (Minimum) Heap: A (minimum) Heap is a binary tree that stores entries at its nodes and satisfies the following two properties:
 - ① Heap-order property: for every node p other than the root node, the key stored at p is greater than equal to the key stored at p 's parent.
 - ② Complete binary tree property: it is a complete binary tree.
 - Complete binary tree: A binary tree with height h is a complete binary tree iff levels $0, 1, \dots, h - 1$ have maximum number of nodes possible (i.e., $1, 2, 2^2, \dots, 2^{h-1}$) and the remaining nodes at level h reside in the leftmost possible position.

Data Structures

Heaps and Priority Queues

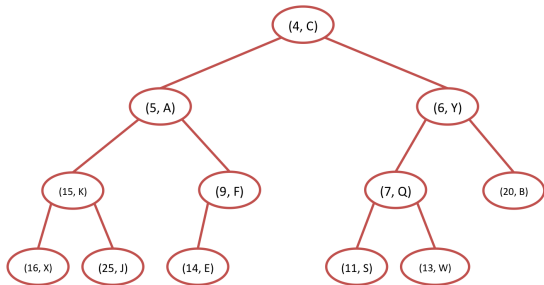
- (Minimum) Heap: A (minimum) Heap is a binary tree that stores entries at its nodes and satisfies the following two properties:
 - 1 **Heap-order property**: for every node p other than the root node, the key stored at p is greater than equal to the key stored at p 's parent.
 - 2 **Complete binary tree property**: it is a complete binary tree.
 - **Complete binary tree**: A binary tree with height h is a complete binary tree iff levels $0, 1, \dots, h-1$ have maximum number of nodes possible (i.e., $1, 2, 2^2, \dots, 2^{h-1}$) and the remaining nodes at level h reside in the leftmost possible position.
- Is this a min-heap?



Data Structures

Heaps and Priority Queues

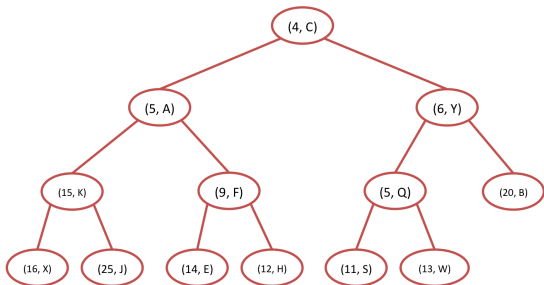
- (Minimum) Heap: A (minimum) Heap is a binary tree that stores entries at its nodes and satisfies the following two properties:
 - 1 Heap-order property: for every node p other than the root node, the key stored at p is greater than equal to the key stored at p 's parent.
 - 2 Complete binary tree property: it is a complete binary tree.
 - Complete binary tree: A binary tree with height h is a complete binary tree iff levels $0, 1, \dots, h-1$ have maximum number of nodes possible (i.e., $1, 2, 2^2, \dots, 2^{h-1}$) and the remaining nodes at level h reside in the leftmost possible position.
- Is this a min-heap? No



Data Structures

Heaps and Priority Queues

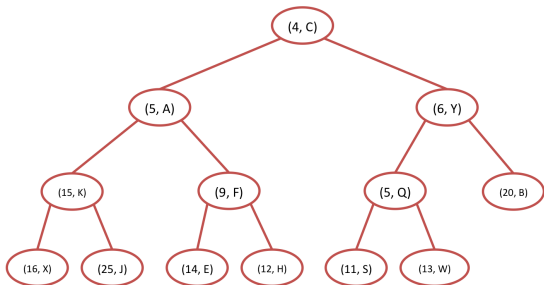
- (Minimum) Heap: A (minimum) Heap is a binary tree that stores entries at its nodes and satisfies the following two properties:
 - 1 Heap-order property: for every node p other than the root node, the key stored at p is greater than equal to the key stored at p 's parent.
 - 2 Complete binary tree property: it is a complete binary tree.
 - Complete binary tree: A binary tree with height h is a complete binary tree iff levels $0, 1, \dots, h-1$ have maximum number of nodes possible (i.e., $1, 2, 2^2, \dots, 2^{h-1}$) and the remaining nodes at level h reside in the leftmost possible position.
- Is this a min-heap?



Data Structures

Heaps and Priority Queues

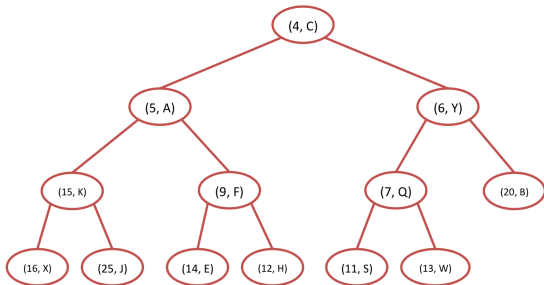
- (Minimum) Heap: A (minimum) Heap is a binary tree that stores entries at its nodes and satisfies the following two properties:
 - 1 Heap-order property: for every node p other than the root node, the key stored at p is greater than equal to the key stored at p 's parent.
 - 2 Complete binary tree property: it is a complete binary tree.
 - Complete binary tree: A binary tree with height h is a complete binary tree iff levels $0, 1, \dots, h-1$ have maximum number of nodes possible (i.e., $1, 2, 2^2, \dots, 2^{h-1}$) and the remaining nodes at level h reside in the leftmost possible position.
- Is this a min-heap? No



Data Structures

Heaps and Priority Queues

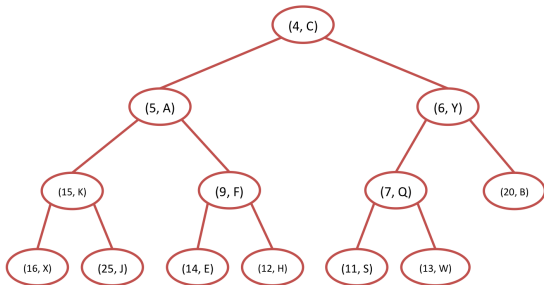
- (Minimum) Heap: A (minimum) Heap is a binary tree that stores entries at its nodes and satisfies the following two properties:
 - 1 Heap-order property: for every node p other than the root node, the key stored at p is greater than equal to the key stored at p 's parent.
 - 2 Complete binary tree property: it is a complete binary tree.
 - Complete binary tree: A binary tree with height h is a complete binary tree iff levels $0, 1, \dots, h-1$ have maximum number of nodes possible (i.e., $1, 2, 2^2, \dots, 2^{h-1}$) and the remaining nodes at level h reside in the leftmost possible position.
- Is this a min-heap?



Data Structures

Heaps and Priority Queues

- (Minimum) Heap: A (minimum) Heap is a binary tree that stores entries at its nodes and satisfies the following two properties:
 - 1 Heap-order property: for every node p other than the root node, the key stored at p is greater than equal to the key stored at p 's parent.
 - 2 Complete binary tree property: it is a complete binary tree.
 - Complete binary tree: A binary tree with height h is a complete binary tree iff levels $0, 1, \dots, h-1$ have maximum number of nodes possible (i.e., $1, 2, 2^2, \dots, 2^{h-1}$) and the remaining nodes at level h reside in the leftmost possible position.
- Is this a min-heap? Yes



Data Structures

Heaps and Priority Queues

- (Minimum) Heap: A (minimum) Heap is a binary tree that stores entries at its nodes and satisfies the following two properties:
 - ① Heap-order property: for every node p other than the root node, the key stored at p is greater than equal to the key stored at p 's parent.
 - ② Complete binary tree property: it is a complete binary tree.
 - Complete binary tree: A binary tree with height h is a complete binary tree iff levels $0, 1, \dots, h - 1$ have maximum number of nodes possible (i.e., $1, 2, 2^2, \dots, 2^{h-1}$) and the remaining nodes at level h reside in the leftmost possible position.
- Question: Show that any heap with n nodes has height $h = \lfloor \log n \rfloor$.

Data Structures

Heaps and Priority Queues

- Consider a pointer based implementation.

Code

```
class minHeap:
    """ The Min Heap class """
    #-----Nested class _Node -----#
    class _Node:
        def __init__(self, element, priority, parent, left_child, right_child):
            self._element = element
            self._priority = priority
            self._parent = parent
            self._left_child = left_child
            self._right_child = right_child
    #-----#
    def __init__(self):
        self._root = None
        self._size = 0

    def __len__(self):
        return self._size
```

Data Structures

Heaps and Priority Queues

- Consider a pointer based implementation.

Code

```
class minHeap:
    """ The Min Heap class """
    #-----Nested class -----#
    class _Node:
        def __init__(self, element, priority, parent, left_child, right_child):
            self._element = element
            self._priority = priority
            self._parent = parent
            self._left_child = left_child
            self._right_child = right_child
    #-----#
    def __init__(self):
        self._root = None
        self._size = 0

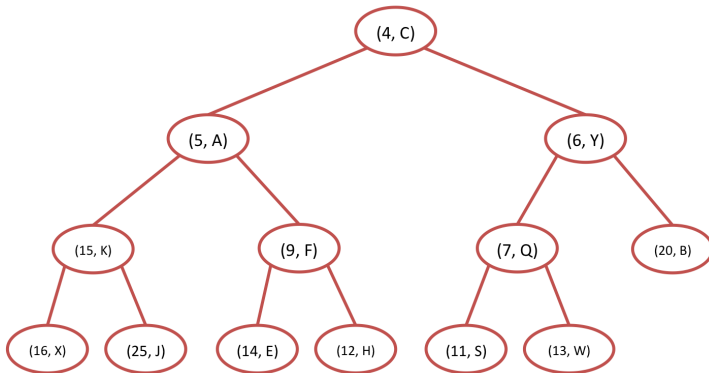
    def __len__(self):
        return self._size
```

- Question: How do we implement `min()`?

Data Structures

Heaps and Priority Queues

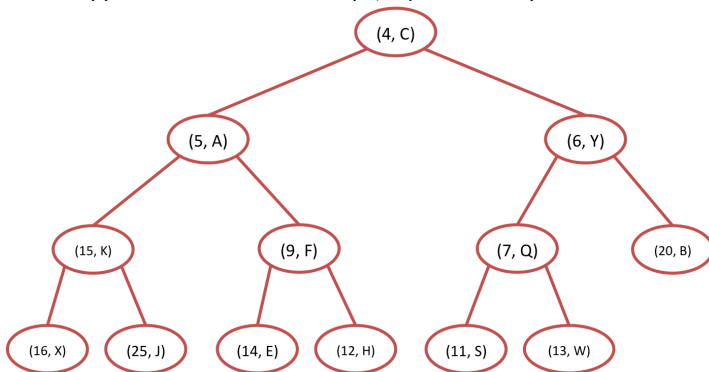
- Consider a pointer based implementation.
- Question: How do we implement `min()`?
- Question: How do we implement `insert( $k, v$ )`?



Data Structures

Heaps and Priority Queues

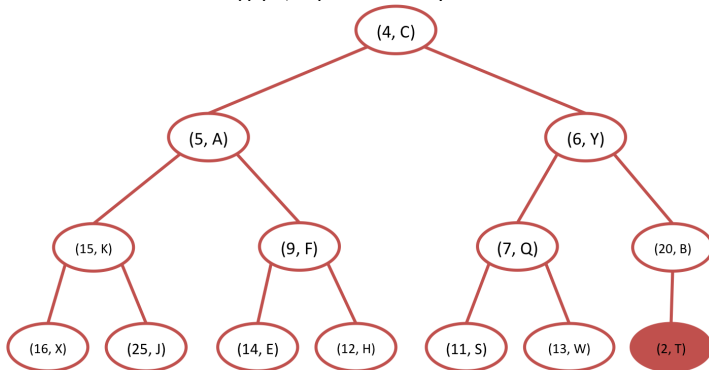
- Consider a pointer based implementation.
- Question: How do we implement `min()`?
- Question: How do we implement `insert( $k, v$ )`?
 - Suppose we want to insert $(2, T)$ in the heap below.



Data Structures

Heaps and Priority Queues

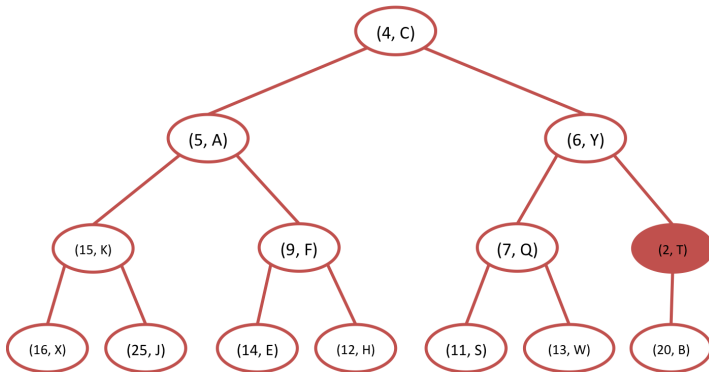
- Consider a pointer based implementation.
- Question: How do we implement `min()`?
- Question: How do we implement `insert( $k, v$ )`?
 - Consider inserting $(2, T)$ in the heap.



Data Structures

Heaps and Priority Queues

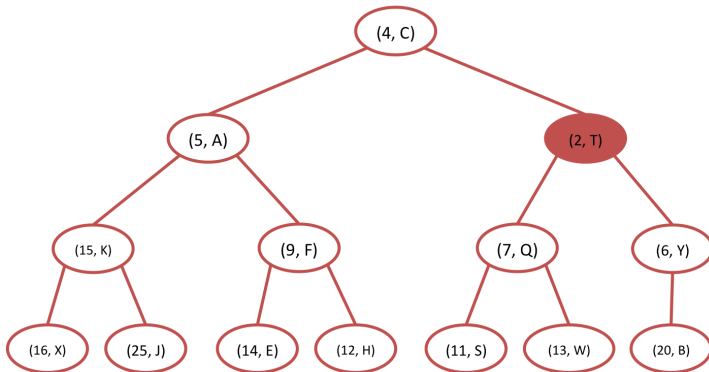
- Consider a pointer based implementation.
- Question: How do we implement `min()`?
- Question: How do we implement `insert( $k, v$ )`?
 - Consider inserting $(2, T)$ in the heap.



Data Structures

Heaps and Priority Queues

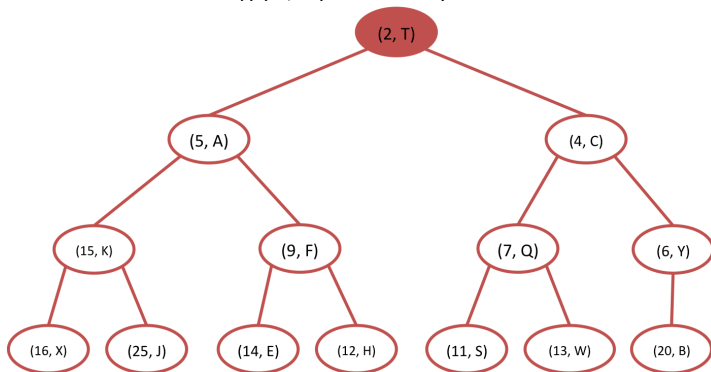
- Consider a pointer based implementation.
- Question: How do we implement `min()`?
- Question: How do we implement `insert( $k, v$ )`?
 - Consider inserting $(2, T)$ in the heap.



Data Structures

Heaps and Priority Queues

- Consider a pointer based implementation.
- Question: How do we implement `min()`?
- Question: How do we implement `insert( $k, v$ )`?
 - Consider inserting $(2, T)$ in the heap.



- This process is called **up-heap bubbling**.

Data Structures

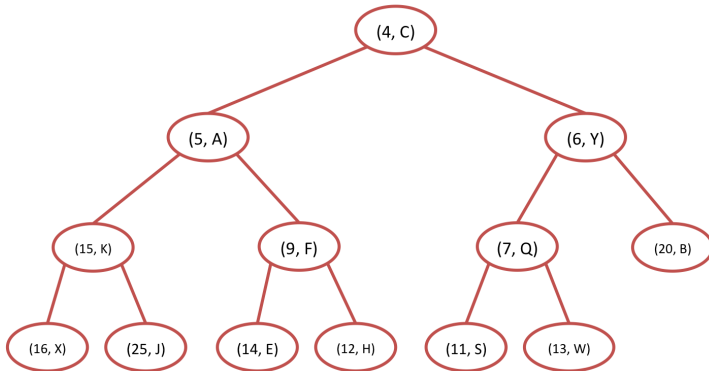
Heaps and Priority Queues

- Consider a pointer based implementation.
- Question: How do we implement `min()`?
- Question: How do we implement `insert( $k, v$ )`?
- Question: How do we implement `removeMin()`?

Data Structures

Heaps and Priority Queues

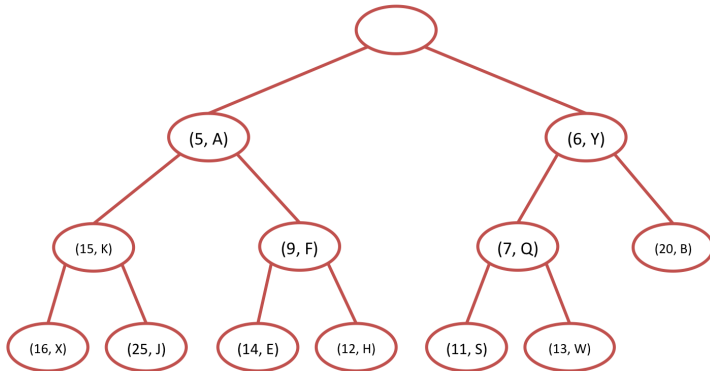
- Consider a pointer based implementation.
- Question: How do we implement `min()`?
- Question: How do we implement `insert(k, v)`?
- Question: How do we implement `removeMin()`?
 - Consider removing the entry with the minimum key from the heap.



Data Structures

Heaps and Priority Queues

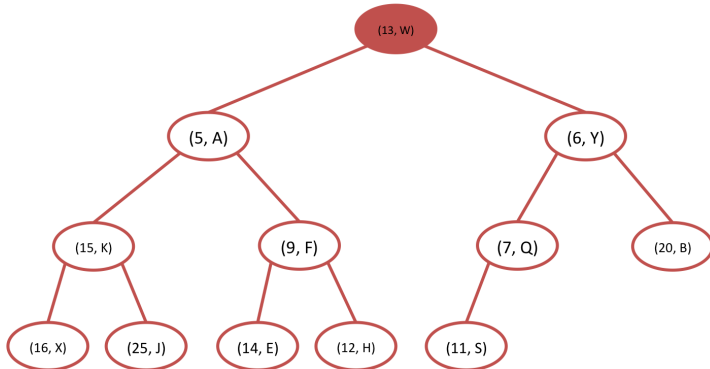
- Consider a pointer based implementation.
- Question: How do we implement `min()`?
- Question: How do we implement `insert(k, v)`?
- Question: How do we implement `removeMin()`?
 - Consider removing the entry with the minimum key from the heap.



Data Structures

Heaps and Priority Queues

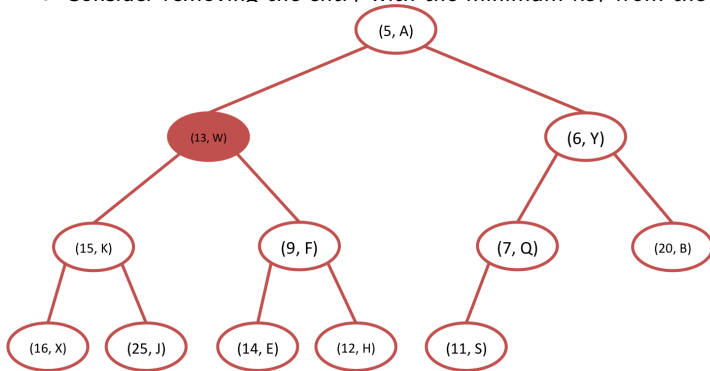
- Consider a pointer based implementation.
- Question: How do we implement `min()`?
- Question: How do we implement `insert(k, v)`?
- Question: How do we implement `removeMin()`?
 - Consider removing the entry with the minimum key from the heap.



Data Structures

Heaps and Priority Queues

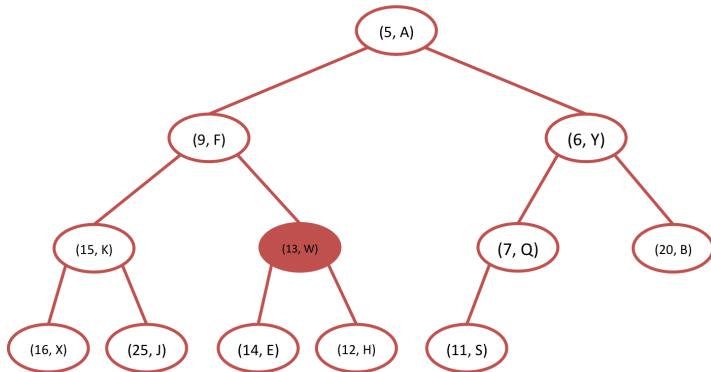
- Consider a pointer based implementation.
- Question: How do we implement `min()`?
- Question: How do we implement `insert(k, v)`?
- Question: How do we implement `removeMin()`?
 - Consider removing the entry with the minimum key from the heap.



Data Structures

Heaps and Priority Queues

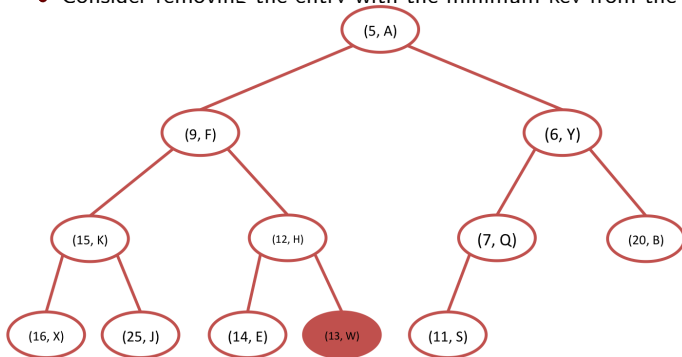
- Consider a pointer based implementation.
- Question: How do we implement `min()`?
- Question: How do we implement `insert(k, v)`?
- Question: How do we implement `removeMin()`?
 - Consider removing the entry with the minimum key from the heap.



Data Structures

Heaps and Priority Queues

- Consider a pointer based implementation.
- Question: How do we implement `min()`?
- Question: How do we implement `insert(k, v)`?
- Question: How do we implement `removeMin()`?
 - Consider removing the entry with the minimum key from the heap.



- This process is called **down-heap bubbling**.

Data Structures

Heaps and Priority Queues

- Let us write the methods for the pointer based implementation.

Code

```
class minHeap:
    """ The Min Heap class """
    #-----Nested class _Node -----#
    class _Node:
        def __init__(self, element, priority, parent, left_child, right_child):
            self._element = element
            self._priority = priority
            self._parent = parent
            self._left_child = left_child
            self._right_child = right_child

    #-----#
    def __init__(self):
        self._root = None
        self._size = 0

    def __len__(self):
        return self._size

    def min(self):
        """ This method returns the minimum element in the heap
        with the least priority.
        """

    def insert(self, element, priority):
        """ This method inserts an element with the given priority
        in the min heap.
        """

    def removeMin(self):
        """ This method removes and returns the element with the
        least priority.
        """
```

Data Structures

Heaps and Priority Queues

- Let us write the methods for the pointer based implementation.
- What is the running time of each operation:
 - `min()`:
 - `insert(k, v)`:
 - `removeMin()`:

Data Structures

Heaps and Priority Queues

- Let us write the methods for the pointer based implementation.
- What is the running time of each operation:
 - `min()`: $O(1)$
 - `insert(k, v)`: $O(\log n)$
 - `removeMin()`: $O(\log n)$

Data Structures

Heaps and Priority Queues

- What is the running time of each operation in the pointer based implementation:
 - `min()`: $O(1)$
 - `insert(k, v)`: $O(\log n)$
 - `removeMin()`: $O(\log n)$

Data Structures

Heaps and Priority Queues

- What is the running time of each operation in the pointer based implementation:
 - `min()`: $O(1)$
 - `insert(k, v)`: $O(\log n)$
 - `removeMin()`: $O(\log n)$
- Note that a **Max-Heap** can be defined in a similar manner as a Min-Heap.

Data Structures

Heaps and Priority Queues

- What is the running time of each operation in the pointer based implementation:
 - `min()`: $O(1)$
 - `insert(k, v)`: $O(\log n)$
 - `removeMin()`: $O(\log n)$
- Pointer manipulations can sometimes be intricate (recall the `incrementLastNode` methods in the homework).
- Question: Can we implement Min-Heap using an **Array**?

Data Structures

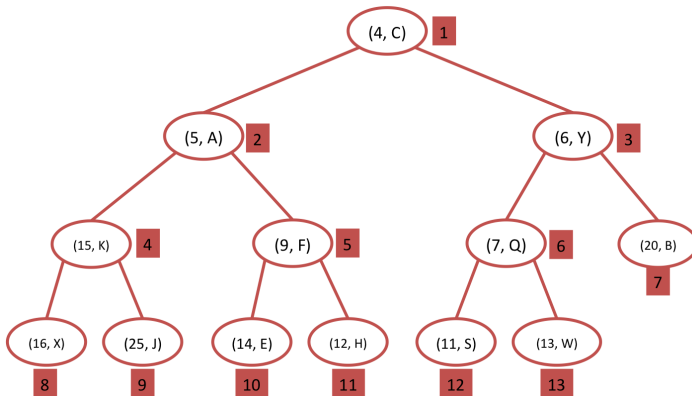
Heaps and Priority Queues

- What is the running time of each operation in the pointer based implementation:
 - `min()`: $O(1)$
 - `insert(k, v)`: $O(\log n)$
 - `removeMin()`: $O(\log n)$
- Pointer manipulations can sometimes be intricate (recall the `incrementLastNode` methods in the homework).
- Question: Can we implement Min-Heap using an **Array**?
 - Array elements are typically stored in contiguous locations in the memory and this has its advantages.

Data Structures

Heaps and Priority Queues

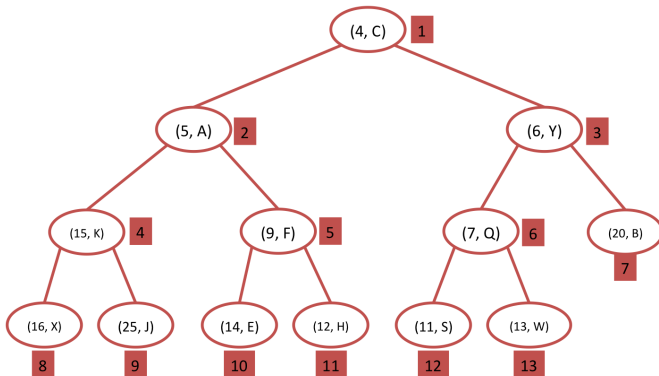
- Question: Can we implement Min-Heap using an **Array**?
- Consider the following example of a min-heap. Let us label the nodes of this **complete** binary tree level-wise as shown below.
- Do you observe some pattern in the labels?



Data Structures

Heaps and Priority Queues

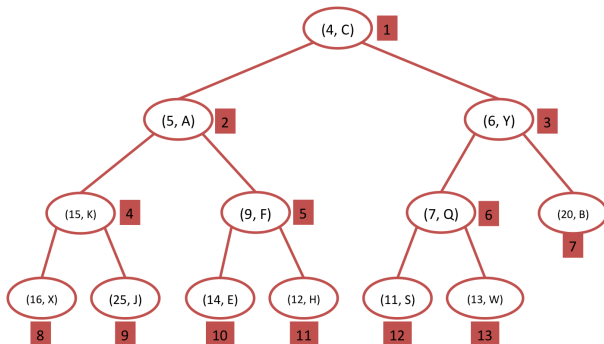
- Question: Can we implement Min-Heap using an **Array**?
- Consider the following example of a min-heap. Let us label the nodes of this **complete** binary tree level-wise as shown below.
- Do you observe some pattern in the labels?
 - Claim 1: Nodes of level i are labeled $2^i, \dots, 2^{i+1} - 1$.



Data Structures

Heaps and Priority Queues

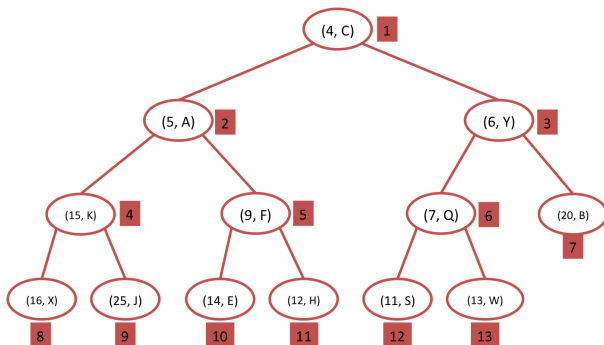
- Question: Can we implement Min-Heap using an **Array**?
- Consider the following example of a min-heap. Let us label the nodes of this **complete** binary tree level-wise as shown below.
- Do you observe some pattern in the labels?
 - Claim 1: Nodes of level i are labeled $2^i, \dots, 2^{i+1} - 1$.
 - Claim 2: For every node labeled i , the label of its children (if they exist) are _____ and _____ and the label of its parent is _____.



Data Structures

Heaps and Priority Queues

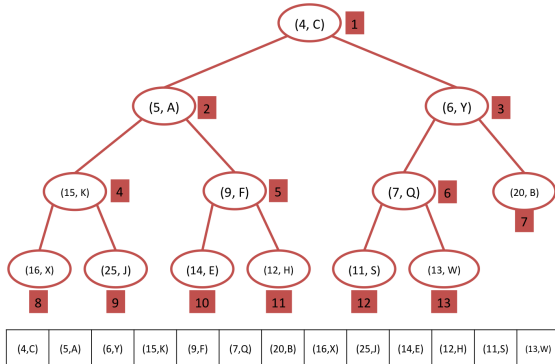
- Question: Can we implement Min-Heap using an **Array**?
- Consider the following example of a min-heap. Let us label the nodes of this **complete** binary tree level-wise as shown below.
- Do you observe some pattern in the labels?
 - Claim 1: Nodes of level i are labeled $2^i, \dots, 2^{i+1} - 1$.
 - Claim 2: For every node labeled i , the label of its children (if they exist) are $2i$ and $(2i + 1)$ and the label of its parent is $\lfloor i/2 \rfloor$.



Data Structures

Heaps and Priority Queues

- Question: Can we implement Min-Heap using an **Array**?
- Consider the following example of a min-heap. Let us label the nodes of this **complete** binary tree level-wise as shown below.
- Do you observe some pattern in the labels?
 - Claim 1: Nodes of level i are labeled $2^i, \dots, 2^{i+1} - 1$.
 - Claim 2: For every node labeled i , the label of its children (if they exist) are $2i$ and $(2i + 1)$ and the label of its parent is $\lfloor i/2 \rfloor$.
- Main idea: Store node labeled i at index i of the array.



End