Priority Queues (Binary Heaps)

CSE 2020 Computer Science II

Learning Objectives

- Define priority queue ADT.
- Compare the performance of different implementation of priority queue, including array, linked structure, and binary search tree.
- Define binary heap ADT
- Describe the process of creating a binary heap, deleting min element from binary heap.
- Design and implement binary heap.

Priority Queue ADT

- A priority queue ADT is a collection of prioritized elements that supports arbitrary element insertion but supports removal of elements in order of priority, that is, the element with the highest priority (the least priority value) is removed at any time.
- A special queue, not FIFO. When dequeuing an element, the element with the highest priority is dequeued, such as, OS scheduler runs a process, printer runs a print job.

Operations of Priority Queue

- Priority queue allows at least two operations:
  - `insert(const C & x)` inserts element x in priority queue
  - `deleteMin()` removes the element with the highest priority (the least priority value)

Other Operations

- isEmpty() returns true if the priority queue is empty
- getSize() returns the number of elements in priority queue
- makeEmpty() makes the priority queue empty state
- ……

Array Implementation

- Use array to implement priority queue
  - insert(x), add x at the end of array, O(1)
  - deleteMin(), find min element (linear search) and delete it (shift elements), O(n) + O(n) = O(n)
- Using sorted array (largest to smallest) to implement priority queue
  - insert(x), find the right position of x (binary search) and add x (shift elements), O(logn) + O(n) = O(n)
  - deleteMin(), delete min which is the last element in the array, O(1)

Linked Structure Implementation

- Use linked structure implement priority queue
  - insert(x), add x at the front of linked structure, O(1)
  - deleteMin(), find min element (linear search) and delete it, O(n) + O(1) = O(n)
- Use sorted linked structure (smallest to largest) implement priority queue
  - insert(x), find the right position of x (linear search) and add x, O(n) + O(1) = O(n)
  - deleteMin(), delete min which is the front element in the linked structure, O(1)

Binary Search Tree Implementation

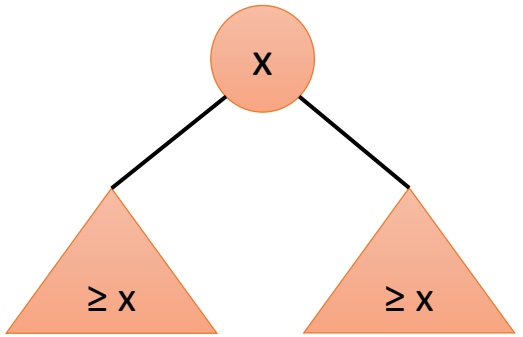
- Balanced binary search tree, height is $\log_2 n$
  - insert(x), insert x as a leaf node, balance the tree, O(logn)
  - deleteMin(), delete the most left node, balance the tree, O(logn)
- Disadvantages
  - always delete the most left node -> unbalanced BST, unless balance the tree everytime after deleteMin()
  - overqualified

Comparison of Implementation

	insert()	deleteMin()
array	O(1)	O(N)
sorted array	O(N)	O(1)
linked	O(1)	O(N)
sorted linked	O(N)	O(1)
binary search tree	O(logN)	O(logN)

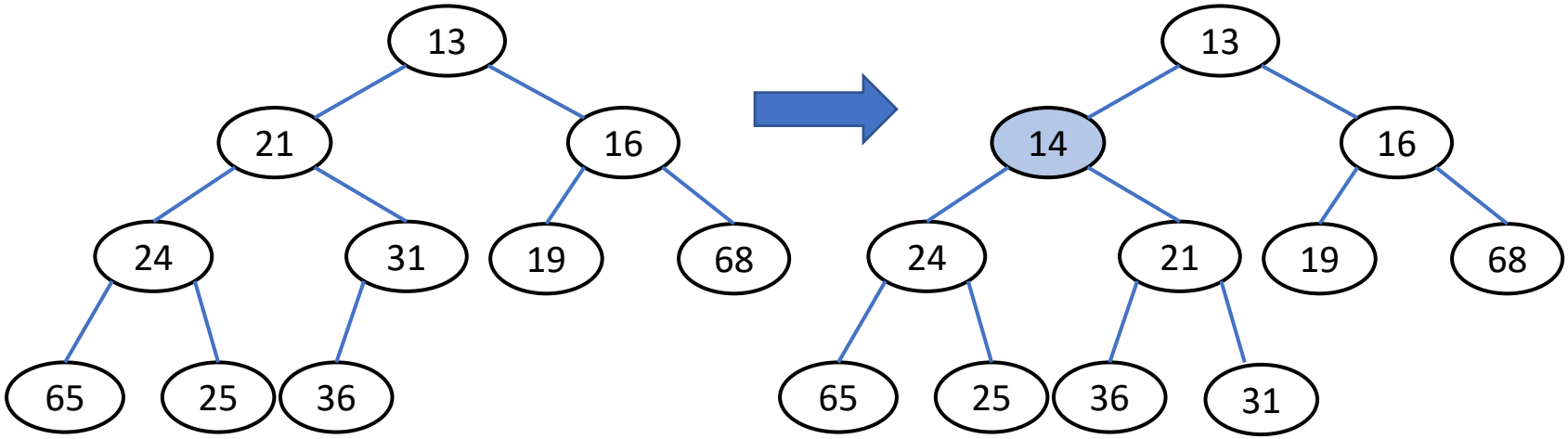
Binary Heap ADT

- It is very common to use a binary heap to implement priority queue. Priority queue is also called binary heap or heap.
- A binary heap has two properties
  - structure: complete binary tree
  - heap-order: root element is less than or equal to the elements in its both subtrees



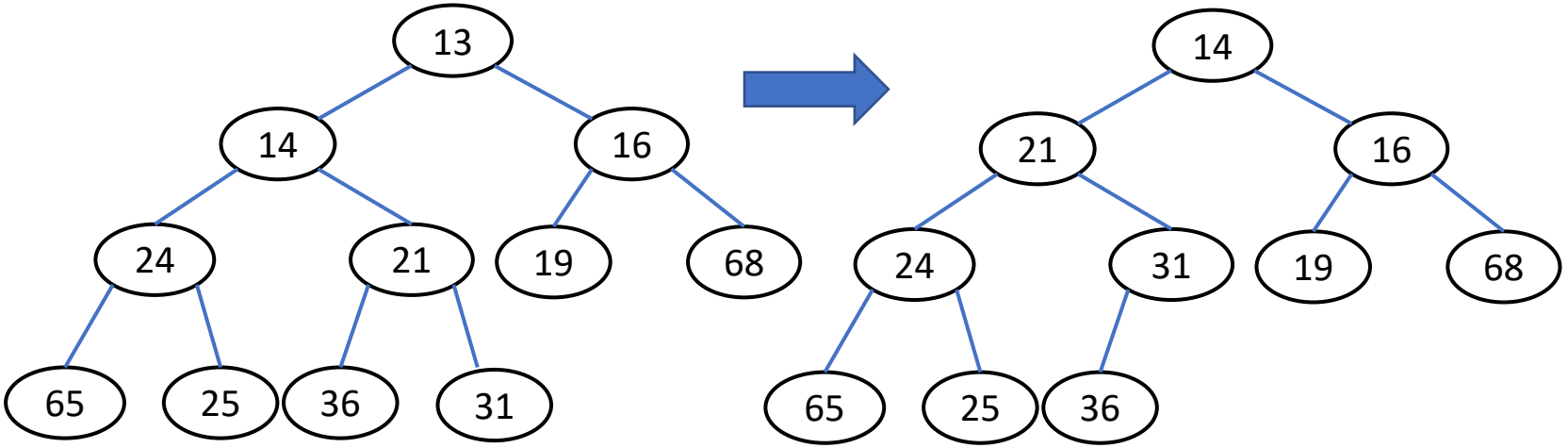
Binary Heap Operation *insert*

- insert 14
  - percolate up the hole to right position, add 14 in the hole



Binary Heap Operation *deleteMin*

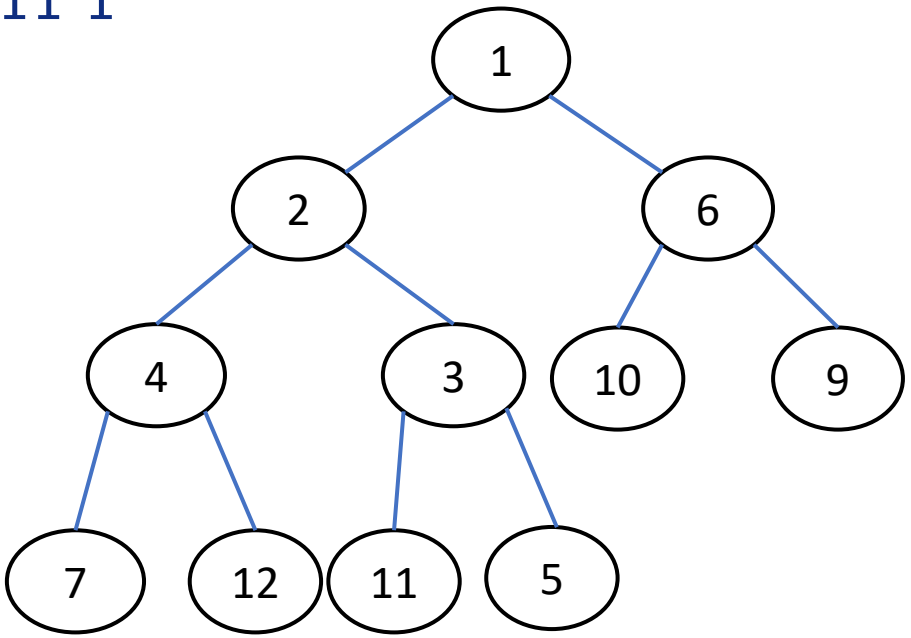
- deleteMin
  - percolate down the hole to the right position, add 31 in the hole



Build Binary Heap Example 1

- Build a binary heap by inserting a sequence of elements to an empty heap

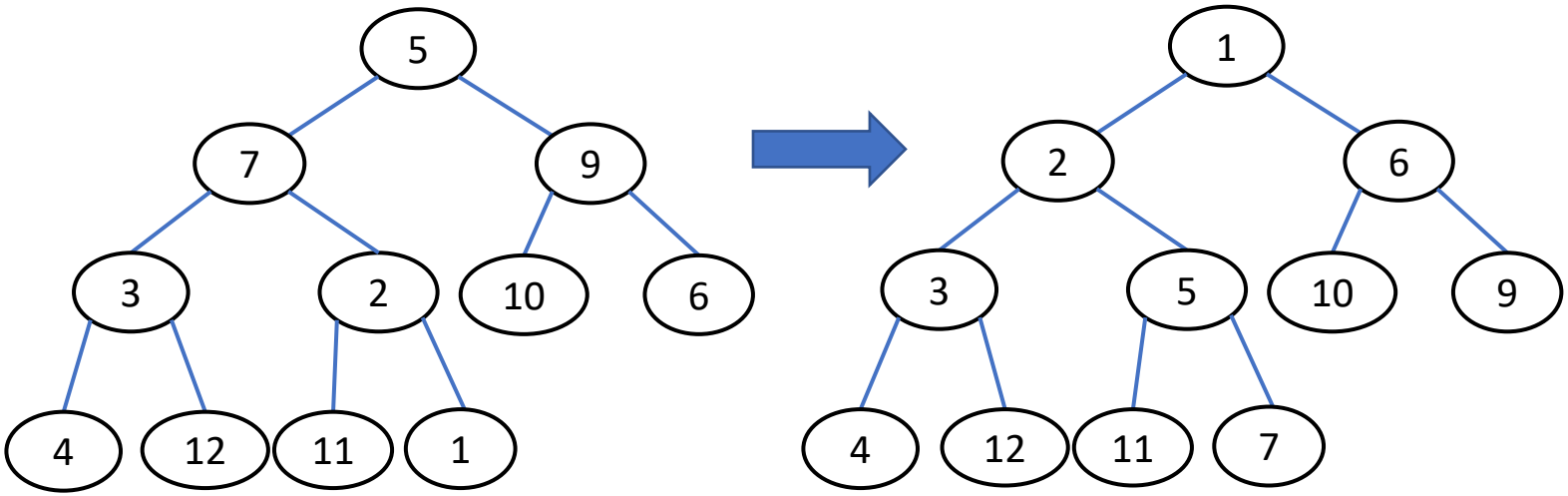
  5 7 9 3 2 10 6 4 12 11 1



Build Binary Heap Example 2

- Convert a complete binary tree into a binary heap

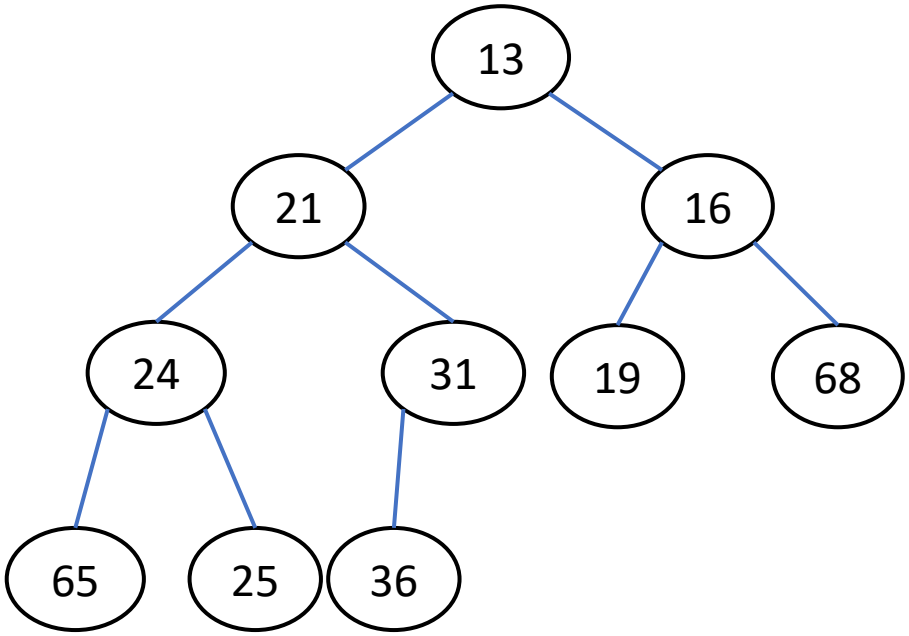
  level-order: 5 7 9 3 2 10 6 4 12 11 1



Use Array Implement Binary Heap

- The root element is stored at index 1
- The element at index i
  - left child at index $2i$
  - right child at index $2i + 1$
  - parent at index $\left\lfloor \frac{i}{2} \right\rfloor$ or $floor(i/2)$

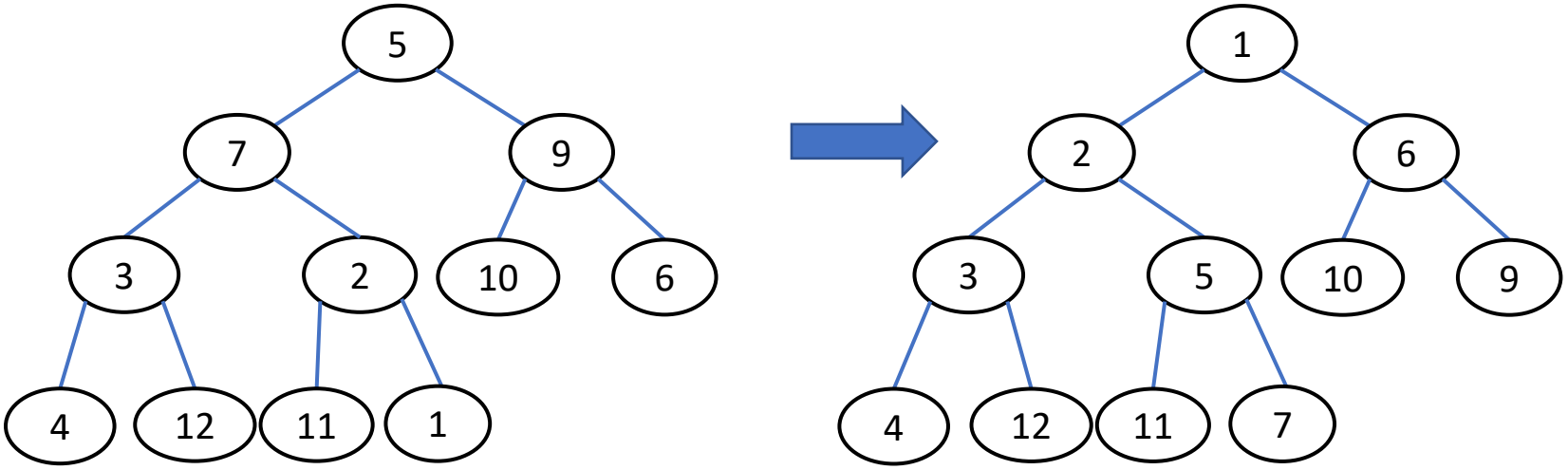
Example



0	1	2	3	4	5	6	7	8	9	10
	13	21	16	24	31	19	68	65	25	36

Practices

- Is the following level order traversal sequence a binary heap? if not, convert the sequence to a binary heap.
  - 5, 7, 9, 3, 2, 10, 6, 4, 12, 11, 1
  - ➔ 1, 2, 6, 3, 5, 10, 9, 4, 12, 11, 7



Implementation of Binary Heaps

- BinaryHeap class template is implemented in BinaryHeap.cpp
  - private attributes: int currentSize; vector<C> items;
  - private function:
    - void buildHeap(): converts an array of elements into a heap
    - void percolateDown(int hole): percolates down items[hole]
  - public function
    - void insert(x): insert x
    - void deleteMin( ): removes the minimal element
    - void deleteMin(minItem): removes and store the minimum in minItem
    - C findMin( ) const: returns the minimal element
    - bool isEmpty( ) const: returns true if empty; else false
    - void makeEmpty( ): removes all elements
- BinaryHeap.txt on Canvas

Time Complexity

- The height of a complete binary tree with N nodes is $H = \lfloor log_2 N \rfloor$
- insert(x) adds x at the end of the array and then percolates it up to the right place in the heap. The worst case is to percolate x up to the root. The $log_2 N$ times of comparisons should be done, O(logN)
- deleteMin() overwrites the minimal element with the last element and then percolates it down to the right place in the heap. The worst case is to percolate it down to the last level of the binary heap. The $2log_2 N$ times of comparisons should be done, O(logN)

Priority Queues in STL

- The binary heap is implemented in STL by the class template named *priority_queue* found in the standard header file *queue.*
- STL implements a **max-heap**, the largest item is the one that is accessed.
- The key member functions are:
  - void push(const C & x)
  - const C & top() const
  - void pop()
  - bool empty()
  - void clear()
- *https://www.geeksforgeeks.org/priority-queue-in-cpp-stl/*