CSC3100 Data Structure
Tutorial_9 Heap

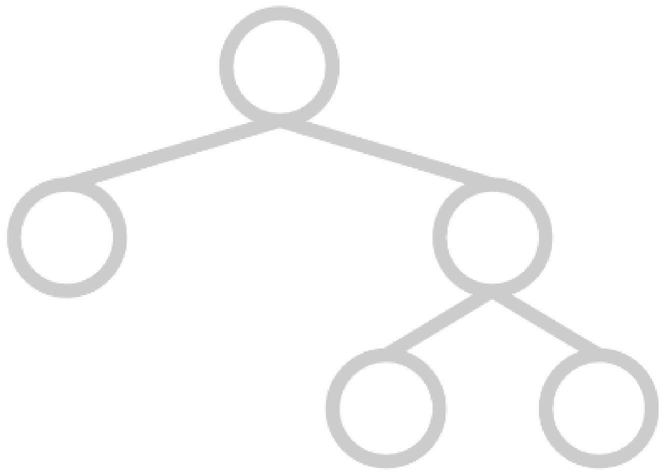
Chen Shi
224040349@link.cuhk.edu.cn

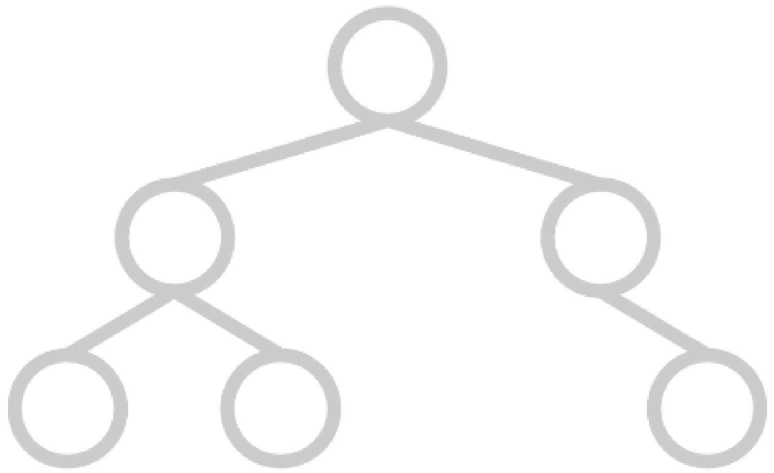
Review

- Heaps are **tree-like structures** that follow two additional invariants that will be discussed more later.
- Normally, elements in a heap can have any number of children, but in this course we will restrict our view to **binary heaps**, where each element will have at most two children.
- Thus, binary heaps are essentially binary trees with two extra invariants.

Invariant 1: Completeness

- We will define balance in a binary heap's underlying tree-like structure as *completeness*.
- A **complete tree** has all available positions for elements filled, except for possibly the last row, which must be filled left-to-right. A heap's underlying tree structure must be complete.





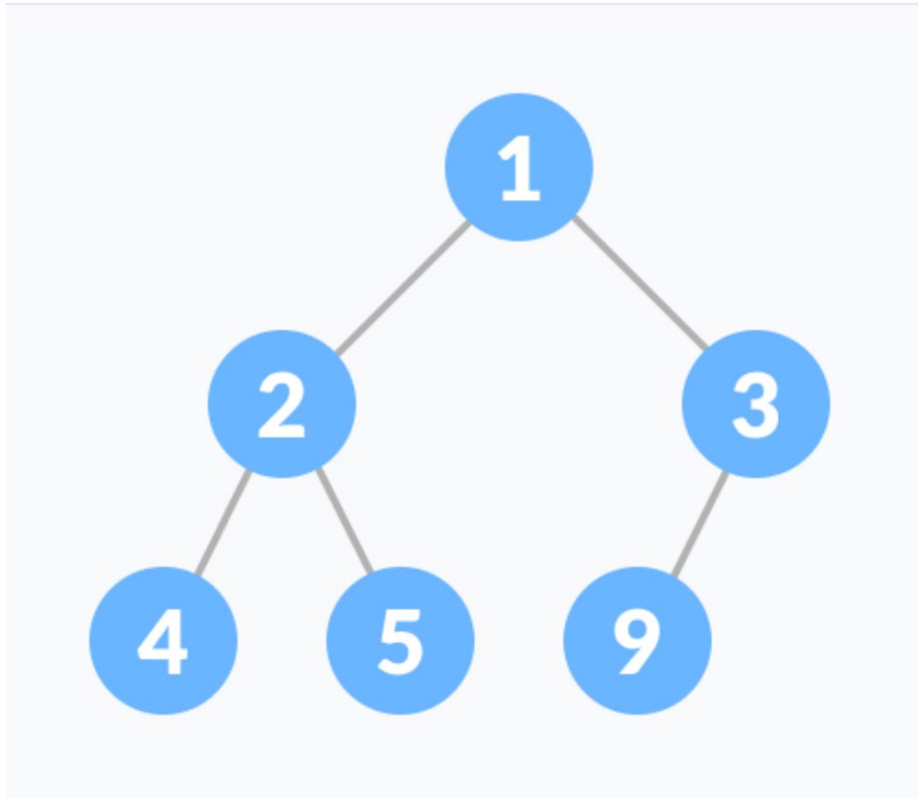
Invariant 2: Heap Property

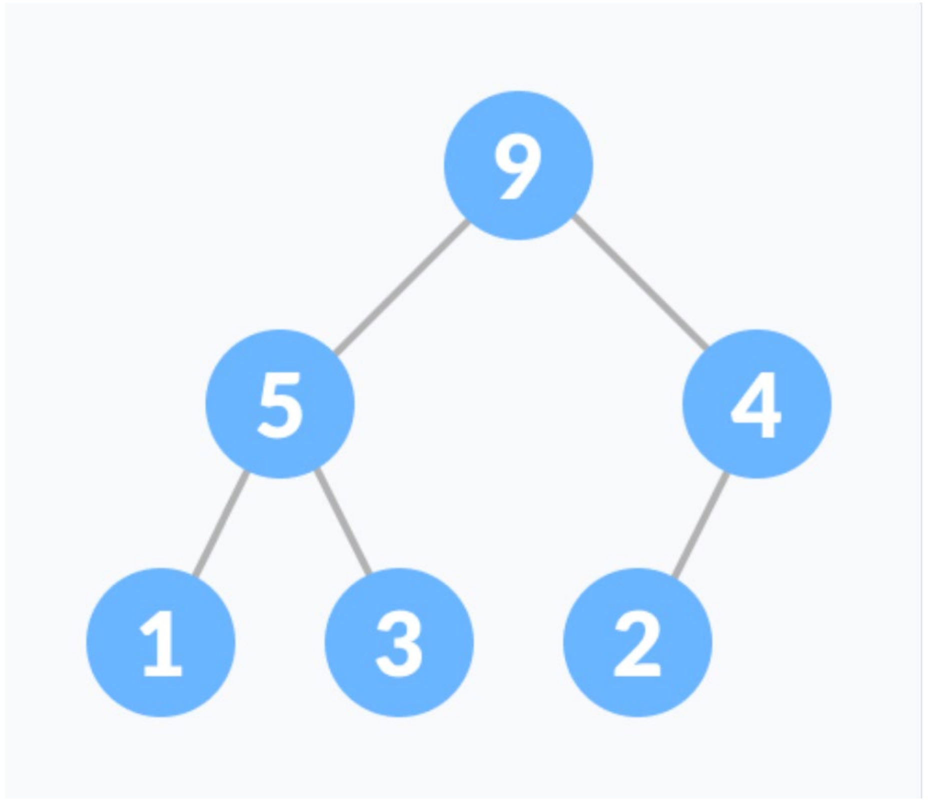
- Each element must be smaller than or equal to all of the elements in its subtree. This is known as the *heap property*
- If we have a heap, this guarantees that the element with the lowest value will always be at the root of the tree. If the elements are our priority values, then we are guaranteed that the element with the lowest priority value (the minimum) is at the root of the tree. This helps us access that item quickly, which is what we need for a (minimum) priority queue!

Min Heap & Max Heap

- The heap we described is also known more specifically as a “min-heap”, because the heap property has the minimum element at the root at the root of the tree.

- There is a variant of the heap data structure that is very similar: the max heap. Max heaps have the same completeness invariant, but have the opposite heap property. In a max heap, each element must be larger than or equal to all of the elements in its subtree. This means that the element with the highest priority value (the maximum) is at the root of the tree.



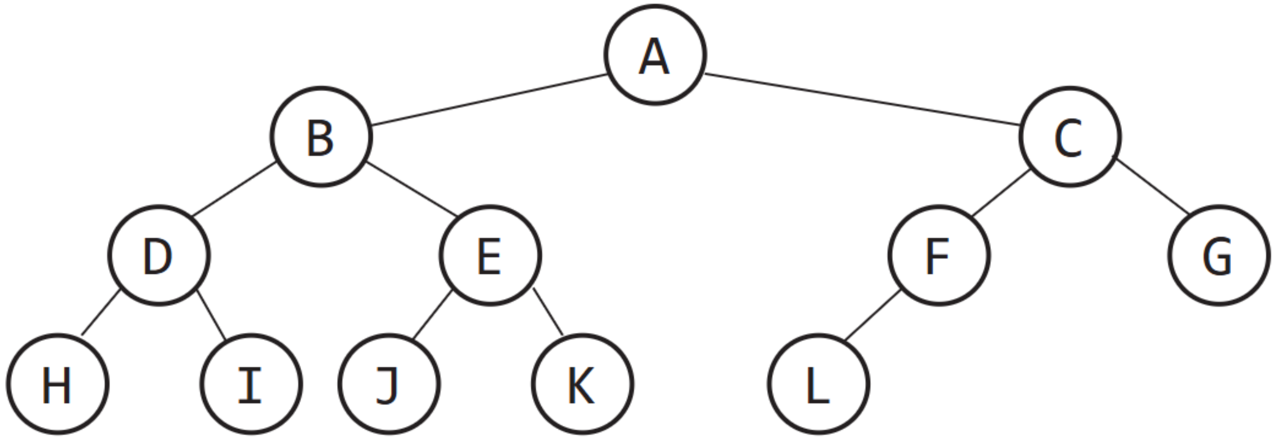


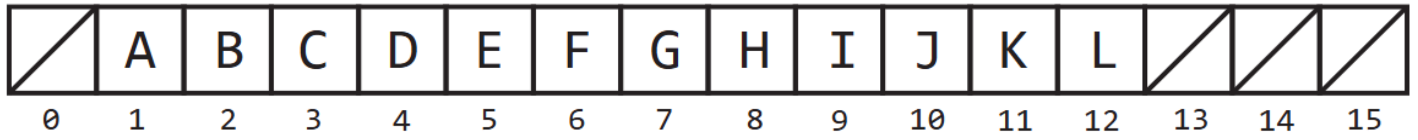
Min heap

Max heap

Heap Representation

we can represent a binary tree using an array:





- The root of the tree will be in position 1 of the array (nothing is at position 0).
- The left child of a node at position N is at position 2N.
- The right child of a node at position N is at position 2N+1.
- The parent of a node at position N is at position N/2.

Binary Heap

- Some functions
  - void destroy ( );
  - void makeEmpty ( );
  - void insert (ElementType X);
  - ElementType deleteMin ( );
  - ElementType findMin ( );
  - boolean isEmpty ( );
  - boolean isFull ( );

$\rightarrow$

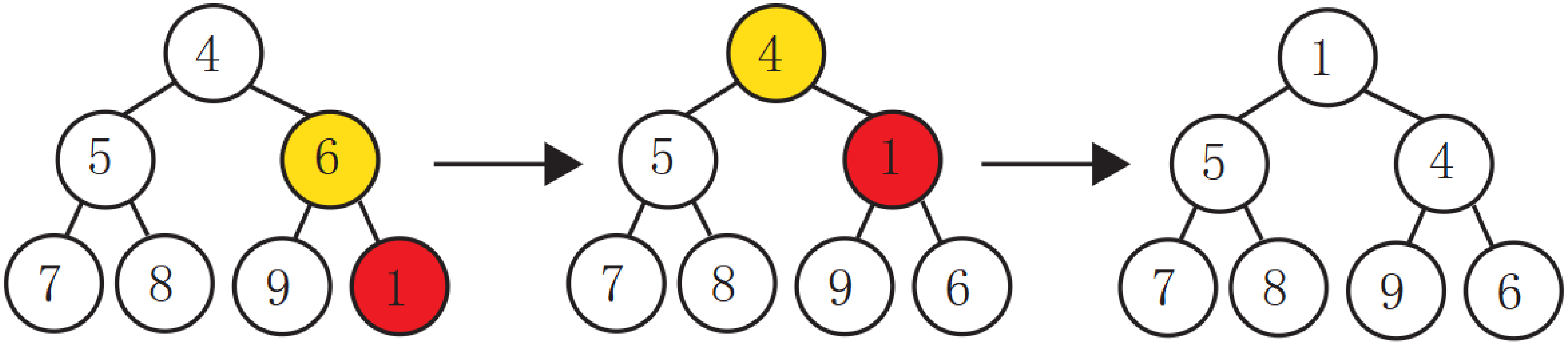
Bubble up/Percolate up

Bubble down/Percolate down

If you want to see an online visualization of heaps, take a look at the USFCA interactive animation of a min heap. You can type in numbers to insert, or remove the min element (ignore the BuildHeap button for now) and see how the heap structure changes.

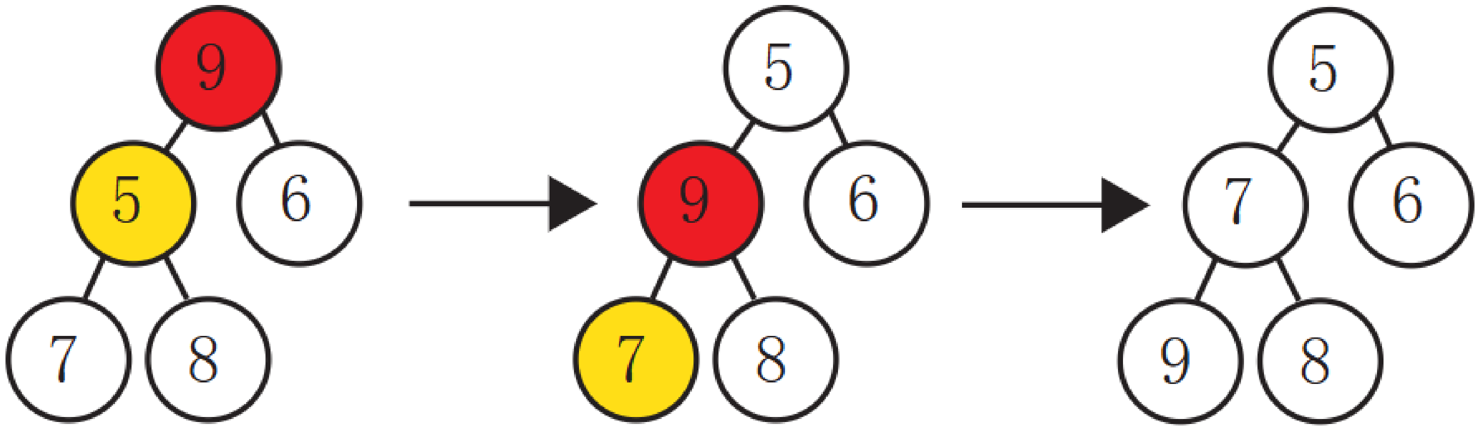
Exercise 1:

Try to write pseudocode for the percolate up



```
Percolate_Up(index) {
    while (index is not the root and array[index] is smaller than parent) {
        swap array[index] with array[parent]
        update index to parent
    }
}
```

What about percolate down? Similar to the code we write before



```
Percolate Down(index) {
    while (there is a child and arr[index] is greater than either child) {
        swap arr[index] with the SMALLER child
        update index to the index of the swapped child
    }
}
```

Exercise 2: Min heap operations

Assume that Heap is a binary min-heap (smallest value on top) data structure that is a properly-implemented heap. Draw the heap and its corresponding array representation after all of the operations below have occurre r.

```
Heap<Character> h = newHeap<>();
        h.insert('f');
        h.insert('h');
        h.insert('d');
        h.insert('b');
        h.insert('c');
        h.removeMin();
        h.removeMin();
```

```
    d
   / \
  f   h

Heap as an array: [ _,'d','f','h']
(_ denotes the absence of the first element)
```

✗

```
    d
   / \
  h   f

Heap as an array: [ _,'d','h','f']
(_ denotes the absence of the first element)
```

Heap Brainteaser: Heap and BST

Consider a binary tree that is both max heap and binary search tree.
How many nodes can such a tree have? Choose all that apply.

A. 1 node
B. 2 nodes
C. 3 nodes
D. 4 nodes
E. 5 nodes
F. Any number of nodes
G. No trees exist

Exercise 3:

You are given an integer array nums and an integer k. You want to find a subsequence of nums of length k that has the largest sum. Return any such subsequence as an integer array of length k.
A subsequence is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.
Example 1:Input: nums = [2,1,3,3], k = 2
Output: [3,3]
Explanation:The subsequence has the largest sum of 3 + 3 = 6.
Example 2:Input: nums = [-1,-2,3,4], k = 3
Output: [-1,3,4]
Explanation: The subsequence has the largest sum of -1 + 3 + 4 = 6.
Example 3:Input: nums = [3,4,3,3], k = 2
Output: [3,4]
Explanation:The subsequence has the largest sum of 3 + 4 = 7. Another possible subsequence is [4, 3].

2099. 找到和最大的长度为 K 的子序列 - 力扣（LeetCode）

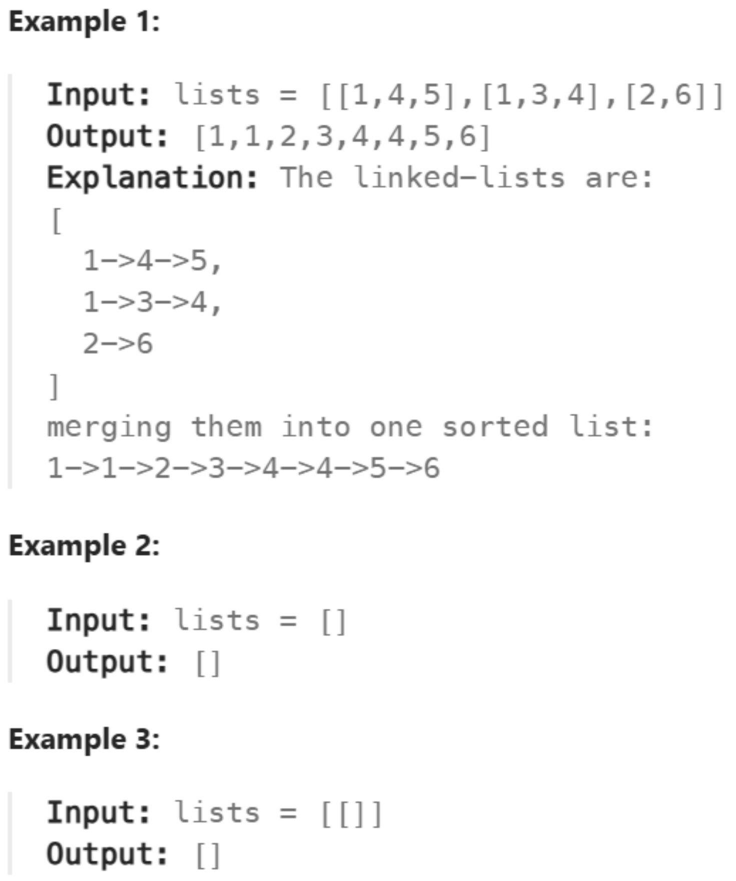
Exercise 3:

You are given an integer array nums and an integer k. You want to find a subsequence of nums of length k that has the largest sum. Return any such subsequence as an integer array of length k.
A subsequence is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

- Initialize the min heap with the first K elements
  - Each node contains value and index
- Iterate through the remaining elements, compare it with the smallest element currently in heap
  - If smaller, continue
  - If larger, remove the smallest element from the heap and add this new element
- The final heap contains the the indices of the k largest elements
- Sort indices to preserve the subsequence order
- Complexity: O(n logk)

Exercise 4: Merge K sorted Lists

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.
Merge all the linked-lists into one sorted linked-list and return it.



Example 1:

```
Input: lists = [[1,4,5],[1,3,4],[2,6]]
Output: [1,1,2,3,4,4,5,6]
Explanation: The linked-lists are:
[
  1->4->5,
  1->3->4,
  2->6
]
merging them into one sorted list:
1->1->2->3->4->4->5->6
```

Example 2:

```
Input: lists = []
Output: []
```

Example 3:

```
Input: lists = [[]]
Output: []
```

23. 合并 K 个升序链表 - 力扣（LeetCode）

Exercise 4: Merge K sorted Lists

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order. Merge all the linked-lists into one sorted linked-list and return it.

- Initialize the min heap with the first element of K linked lists
  - Each node contains value and pointer to the node
- Iterate extract the smallest value node from the heap
  - Append this node to result list
  - If the node we just processed has a next node, we add that next node to the heap
- Return results
- Complexity: O(N logk), N is the number total of nodes

Thanks