

Design and Analysis of Algorithms

L28: Heapsort

Transform and Conquer Approach

Dr. Ram P Rustagi

Sem IV (2019-H1)

Dept of CSE, KSIT/KSSEM

rprustagi@ksit.edu.in

Resources

- Text book 1: Sec 9.1-5.4 - Levitin
- RI: Introduction to Algorithms
 - Cormen et al.

Transform and Conquer

- Secret to life: Replace one worry with another.
 - American cartoonist Charles M Shultz (1922-2000)
- Transform and conquer approach
 - A two stage process
 - Transformation stage: change the problem instance to another form, more amenable to solution
 - Conquering stage: Solve the problem
- Transformation can be done in 3 ways
 - Instance simplification: to a simpler or more convenient instance of the problem: presorted lists
 - Different representation: Heaps, Horner's rule
 - Problem reduction: transform to a different problem for which solution is available.

Priority Queue

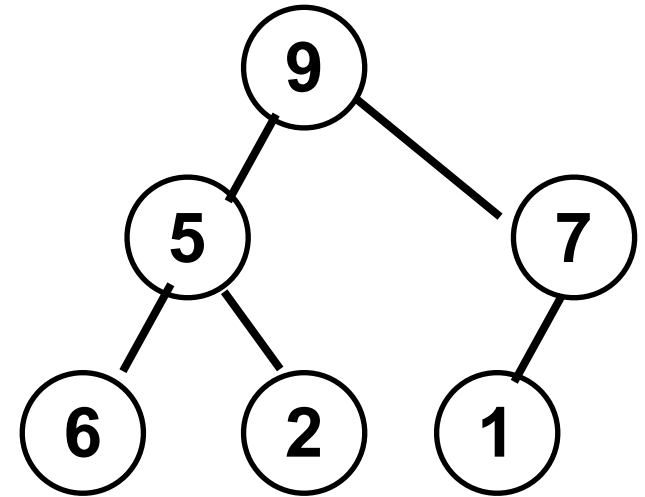
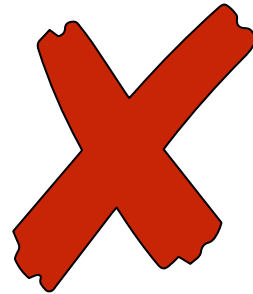
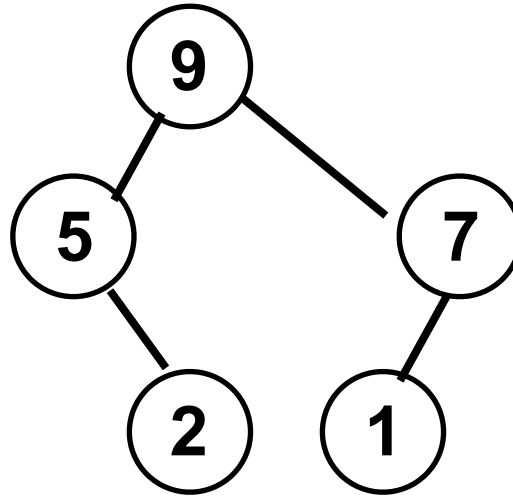
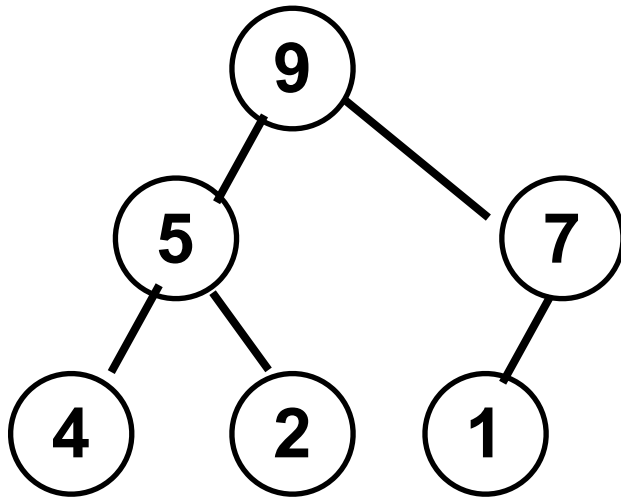
- Priority Queue:
 - A data structure with an orderable (called priority) characteristic on set of elements maintained by it
 - Allows 3 operations in an efficient way
 - FindMin (or even FindMax):
 - Find an item with highest priority (e.g. max, min)
 - DeleteMin:
 - Delete an item with highest priority
 - Insert:
 - Add a new item to the data structure
- Heaps makes these 3 operations interesting and useful
- Heapsort: a cornerstone of theoretical sorting problem

Heap

- Definition:
 - Heap is defined as binary tree with keys assigned to nodes (one key per node) with following conditions
 - Binary tree is a a complete tree except possibly at the last level
 - Few rightmost leaves may be missing
 - The key of a parent is greater than or equal to keys of its children and hence descendants
 - Also, known as parental dominance.

Examples

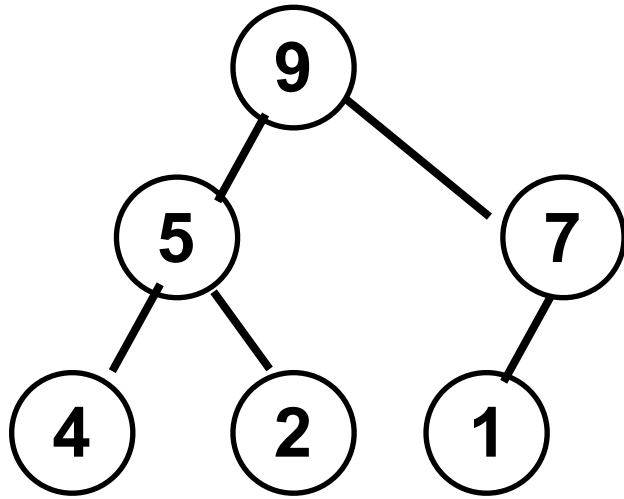
Q: Identify if given binary tree is a heap?



Heap Properties

- There exists only 1 complete binary tree with n nodes.
- The root of the heap is always the largest element
- A node of heap taken together with all its descendants is also a heap
- Heap implementation
 - Can be an array $H[]$ with top-down and left to right
 - Store heap elements in positions thru 1 to n .
 - Element $H[0]$ can either be unused or a sentinel
 - Its value can be greater than every element of heap
 - Parental nodes are in first $\lfloor n/2 \rfloor$ positions of the array
 - Leaf nodes will be last $\lceil n/2 \rceil$ positions of the array

Example: Heap Implementation



- Left child of node at j is at $2j$
- Right child (if exists) of node at j is at $2j+1$
- Parent of node at j is at $\lfloor j/2 \rfloor$
- Parental nodes are in first $\lfloor n/2 \rfloor$ positions of the array
- Leaf nodes are in last $\lceil n/2 \rceil$ positions of the array

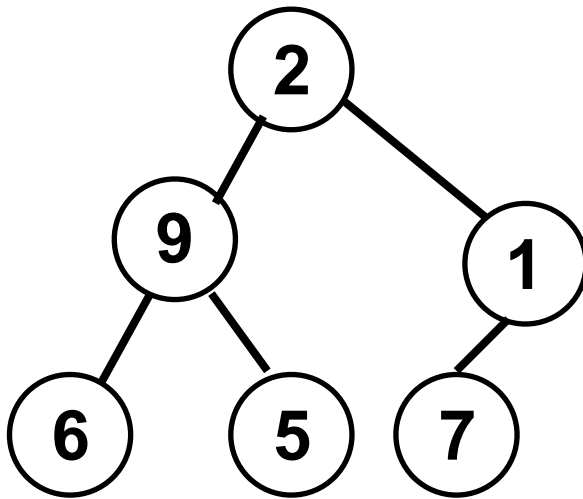
0	1	2	3	4	5	6
	9	5	7	4	2	1

Heap Construction

- S0: Initialize heap structure with keys in order
- S1: Start with the last (right most) parental node
 - Fix the heap rooted at it.
 - If it fails the heap condition, then exchange with larger child
 - Repeat the process till heap condition satisfies
- S2: Repeat the previous step (s1) for preceding parental nodes.

Heap Construction

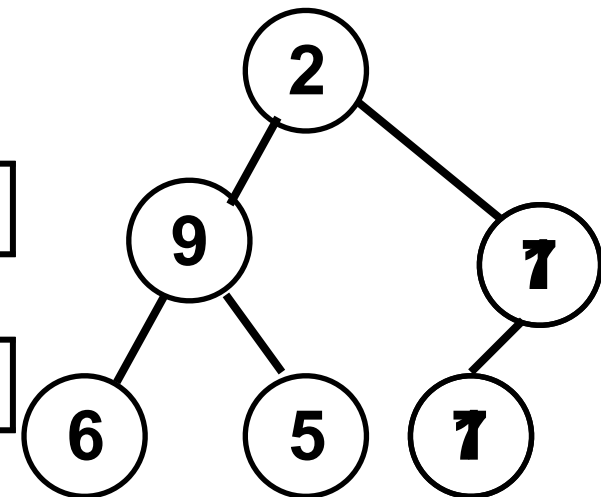
- Consider the data: 2, 9, 1, 6, 5, 7
- Construct the heap in order.



- Let us heapify
- Last parental node (at $\lfloor 6/2 \rfloor = 3$ is 1
 - Smaller than child node 7 at pos 6
 - Exchange it
 - Heap property satisfies

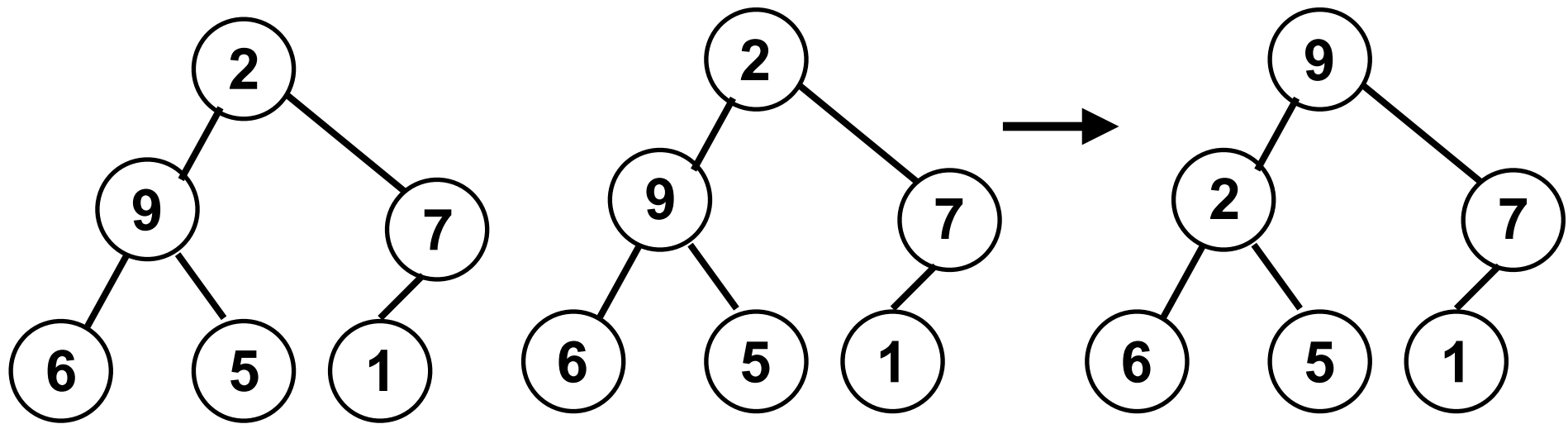
-	2	9	1	6	5	7
---	---	---	---	---	---	---

-	2	9	7	6	5	1
---	---	---	---	---	---	---



Heap Construction

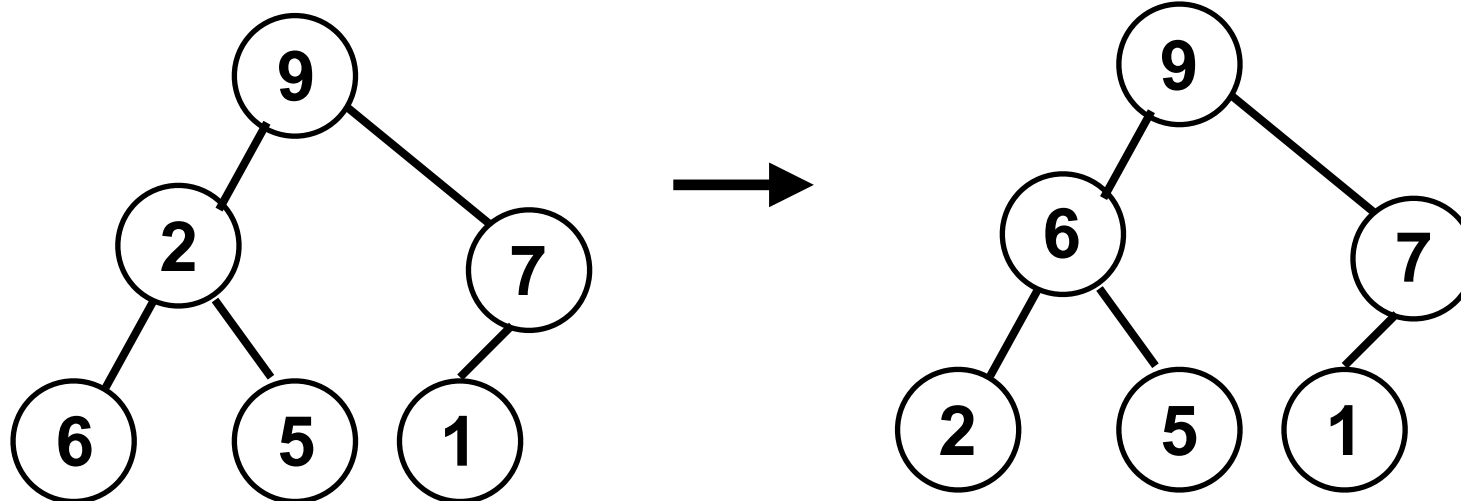
- Consider preceding parental node 9 (at pos $3-1=2$)
- It is in order. No exchange required
- Next parental node 2 (at pos 1). Needs heapification.
- Exchange it with 9, and repeat the process



keys: 2, 9, 7, 6, 5, 1

Heap Construction

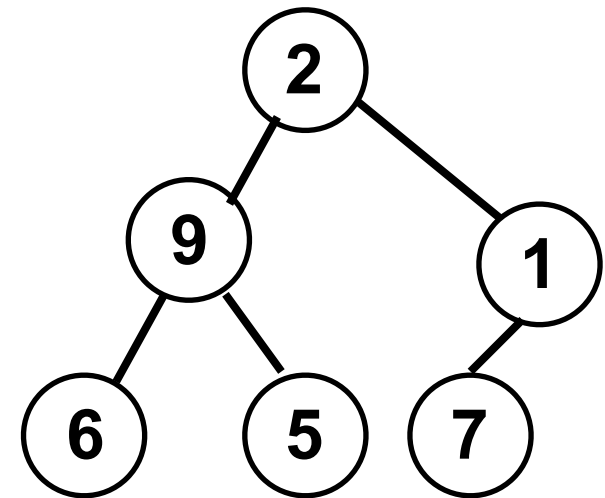
- Since exchanged node 2 is not in heap order
- This needs to be exchanged with 6.



- Now heap is in order.

Heap Algorithm

- **Algo** HeapBottomUp ($H[1:n]$)
// i/p: an array $H[1:n]$ of items to be ordered
// o/p: Heap $H[1:n]$ of ordered items
for $i \leftarrow \lfloor n/2 \rfloor$ **to** 1 **do**
 $k \leftarrow i$; $v \leftarrow H[k]$; heap \leftarrow False
 while not heap **and** $2*k \leq n$ **do**
 $j \leftarrow 2*k$
 if $j < n$ // there are two children
 if $H[j] < H[j+1]$
 $j \leftarrow j+1$
 if $v \geq H[j]$
 heap \leftarrow True
 else
 $H[k] \leftarrow H[j]$; $k \leftarrow j$
 $H[k] \leftarrow v$



Complexity Analysis

- Consider the tree height
 - height of a node: length of the path from it to leaf
 - n -element heap has height $\lfloor \lg_2 n \rfloor$
 - Number of nodes at height h is $\lceil n/2^{h+1} \rceil$
 - e.g. $n=15$, $h=3$,
 - nodes at $h=0$ is 8, at $h=1$ is 4, at $h=2$ is 2
- Generalized analysis
 - Moving $\lfloor n/2 \rfloor$ nodes i.e. considering parent nodes
 - Each node may move $h = \lfloor \lg_2 n \rfloor$ times.
 - Thus complexity for heapifying array is $\Theta(n \lg_2 n)$

Complexity Analysis: Improved

- Node at height 1 moves at most 1 times
- Node at height 2 moves at most 2 times
- i.e. Node at height h moves at most h times.
- Total number of moves are

$$\sum_{i=0}^h \left\lceil \frac{n}{2^{i+1}} \right\rceil * i = O\left(n \sum_{i=0}^{\lg_2 n} \left\lceil \frac{i}{2^i} \right\rceil\right) \quad (1)$$

Some basic mathematics

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \quad \text{for } x < 1$$

Differentiating both sides

$$\sum_{k=1}^{\infty} kx^{k-1} = \frac{1}{(1-x)^2} \Rightarrow \sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2} \quad (2)$$

Complexity Analysis: Improved

Taking $x = 1/2$ in eqn (2) gives

$$\sum_{k=0}^{\infty} k \left(\frac{1}{2}\right)^k = \frac{\frac{1}{2}}{\left(1 - \frac{1}{2}\right)^2}$$

$$\Rightarrow \sum_{k=0}^{\infty} \frac{k}{2^k} = \frac{\frac{1}{2}}{\frac{1}{4}}$$

$$\Rightarrow \sum_{i=0}^{\infty} \frac{i}{2^i} = 2$$

Thus eqn (1) becomes $O\left(n \sum_{i=0}^{\lg_2 n} \left\lceil \frac{i}{2^i} \right\rceil\right) \leq O(n \cdot 2) = O(n)$

That is heap from the array can be built in $O(n)$ time

Exercises:

- Consider the array
 - 3, 5, 6, 7, 20, 8, 2, 9, 12, 15, 30, 17
 - Draw the Complete Binary Tree
 - Heapify the tree.
 - Workout the updates in array when heapifying.
- In the above heap, insert the following items, one at a time.
 - 16, 21, 45
- Perform 3 DeleteMax operations
 - Show the heap structure after each delete

Summary

- Priority queue
- 3 Operations
 - FindMin
 - DeleteMin
 - Add
- Heap
- Heapification (building an heap)
- Time complexity analysis