

CS483-12 Transform-and-Conquer

Instructor: Fei Li

Room 443 ST II

Office hours: **Tue. & Thur. 1:30pm - 2:30pm** or by appointments

`lifei@cs.gmu.edu` with **subject: CS483**

http://www.cs.gmu.edu/~lifei/teaching/cs483_fall07/

Based on *Introduction to the Design and Analysis of Algorithms* by Anany Levitin and Professor

Jyh-Ming Lien's notes.

Outline

- ▶ Transform and Conquer Techniques (which allow us to handle **dynamic** data / information)
 - Binary search tree
 - AVL tree via **rotations** (or red-black tree or splay tree)
 - 2 – 3 tree (or 2 – 3 – 4 tree or B tree)
 - **Heap**

Priority Queue

- ▶ Consider problems that require you to:
 - schedule tasks (e.g., CPU)
 - match n men to n women (eHarmony.com)
 - route mails (Internet package routing)
- ▶ All these problems need to deal with dynamic data/information and contain information about priority/ordering/preference.

Priority Queue

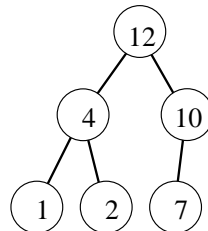
- ▶ Consider problems that require you to:
 - schedule tasks (e.g., CPU)
 - match n men to n women (eHarmony.com)
 - route mails (Internet package routing)
- ▶ All these problems need to deal with dynamic data/information and contain information about priority/ordering/preference.
- ▶ A **priority queue** is needed in these problems to perform the following operations:
 - Find the element with the highest priority
 - Delete the element with the highest priority
 - Insert element

Priority Queue

- Options for building a priority queue
 - a pointer points to the highest priority (what's the drawback?)
 - a sorted array (what's the drawback?)
 - a sorted list (what's the drawback?)
 - a balanced binary search tree (what's the drawback?)

Heap

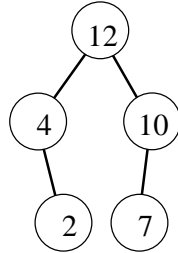
- **Heap** is a binary tree with keys at its nodes (one key per node) such that:
 - It is essentially **complete**, i.e., all its levels are full except possibly the last level, where **only some rightmost keys may be missing**
 - For each node n in a heap, n 's **key** is always **larger** than the keys of n 's **kids** (so, the largest value is in the root)



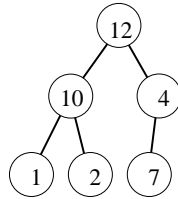
- Heap is data structure good for building priority queue.

Heap

- ▶ Only right most leaves are allowed empty (easier to expand, delete, and store). See the non-heap.



- ▶ Heap's elements are **ordered top down** (along any path down from its root), but they are **not ordered left to right**.



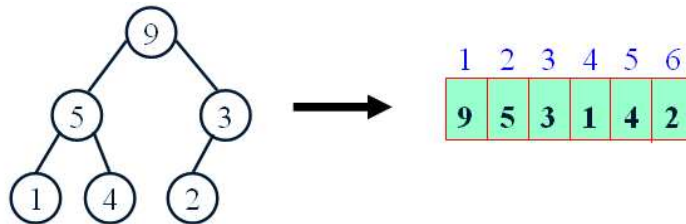
Properties of Heaps

- ▶ Given n , there exists a **unique binary tree** with n nodes that is essentially complete, with $h = \lceil \log_2 n \rceil$
- ▶ The **root** contains the **largest key**
- ▶ The **subtree** rooted at any node of a heap is **also a heap**
- ▶ A heap can be **represented as an array**

Heap's Array Representation

- ▶ Store heap's elements in an array (whose elements indexed, for convenience, 1 to n) in top-down left-to-right order
 - The kids of a node with index i have indices $2i$ and $2i + 1$
 - The parent of a node with index i has index $\lfloor \frac{i}{2} \rfloor$
 - Parental nodes are represented in the first $n/2$ locations

▶ Example:



Heap: Insertion

- ▶ Assuming that we have a heap, and given a value with key k , insert the value to the heap.

Algorithm 0.1: HEAPINSERT(H, k)

Place k at $n + 1$

Let $i = n + 1$

while $H[i] > H[\lfloor \frac{n+1}{2} \rfloor]$ **and** $i > 0$

do $\begin{cases} \text{Swap}(H[i], H[\lfloor \frac{n+1}{2} \rfloor]) \\ i \leftarrow \lfloor \frac{n+1}{2} \rfloor \end{cases}$

- ▶ Time efficiency: $O(\log n)$

▶ Example:

Heap: Top-Down Construction

- ▶ Problem: Given an array $A[1 \dots n]$ of orderable items, output a heap $H[1 \dots n]$.
- ▶ A heap can be constructed by successive insertions of a new key into a previously constructed heap. That is, we can call HEAPINSERT iteratively over all the keys.

Algorithm 0.2: HEAPTODOWN($A[1 \dots n]$)

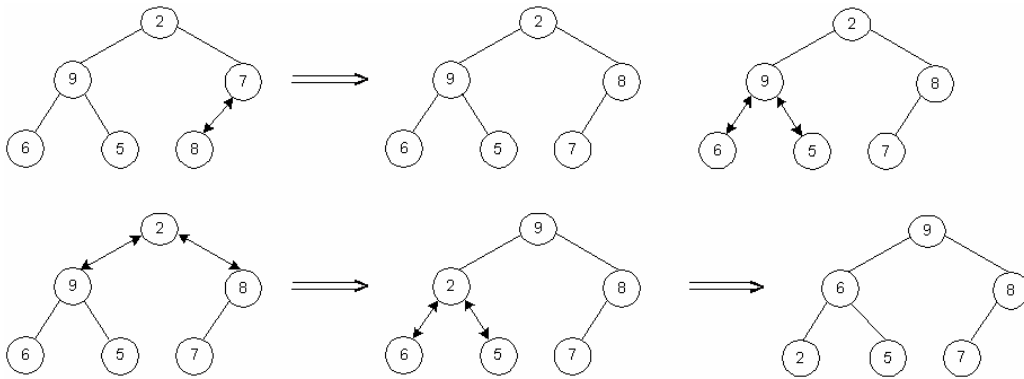
```
 $H \leftarrow A[1]$   
for  $i \in \{2 \dots n\}$   
  do HEAPINSERT( $H, A[i]$ )
```

Heap: Bottom-Up Construction

- Step 0: Initialize the structure with keys in the order given
- Step 1: Starting with the last (rightmost) parental node, fix the heap rooted at it, if it does not satisfy the heap condition: keep exchanging it with its largest child until the heap condition holds
- Step 2: Repeat Step 1 for the preceding parental node

Heap: Bottom-Up Construction

Example: Construct a heap for the list 2, 9, 7, 6, 5, 8



Heap: Bottom-Up Construction Algorithm

Algorithm 0.3: HEAPBOTTOMUP($A[1 \dots n]$)

```

for  $i \leftarrow \{ \lfloor n/2 \rfloor \dots 1 \}$ 
  do  $k \leftarrow i; v \leftarrow A[k]$ 
  heap  $\leftarrow$  false
  while not heap and  $2 \cdot k \leq n$ 
    do  $j \leftarrow 2 \cdot k$ 
    if  $j < n$  there are two children
      do if  $A[j] < A[j + 1]$ 
        do  $j \leftarrow j + 1$ 
    if  $v \geq A[j]$ 
      do heap  $\leftarrow$  true
    else  $A[k] \leftarrow A[j]; k \leftarrow j$ 
   $A[k] \leftarrow v$ 

```

Heap: Deletion

- Deleting the maximum key from a heap is similar to inserting a key

Algorithm 0.4: HEAPDELMAX(H)

swap ($H[1], H[n]$)

Let $i = 1$

while ($H[i] < H[2i]$ **or** $H[i] < H[2i + 1]$) **and** $i < n$

do $\left\{ \begin{array}{l} \text{if } H[2i] > H[2i + 1] \\ \text{then } \left\{ \begin{array}{l} \text{swap}(H[i], H[2i]) \\ i = 2i \end{array} \right. \\ \text{else } \left\{ \begin{array}{l} \text{swap}(H[i], H[2i + 1]) \\ i = 2i + 2 \end{array} \right. \end{array} \right.$

- What's the time complexity?
- Example: Build a heap from this list: $\{2, 9, 7, 6, 5, 8\}$ and delete the root's key

Heapsort

1. Construct a heap for a given list of n keys
2. Repeat operation of root removal $n - 1$ times:
 - Exchange keys in the root and in the last (rightmost) leaf
 - Decrease heap size by 1
 - If necessary, swap new root with larger child until the heap condition holds
3. In-space sorting

Stage 1 (heap construction) Stage 2 (root/max removal)

1 9 7 6 5 8	9 6 8 2 5 7
2 9 8 6 5 7	7 6 8 2 5 9
2 9 8 6 5 7	8 6 7 2 5 9
9 2 8 6 5 7	5 6 7 2 8 9
9 6 8 2 5 7	7 6 5 2 8 9
	2 6 5 7 8 9
	6 2 5 7 8 9
	5 2 6 7 8 9
	5 2 6 7 8 9
	2 5 6 7 8 9

Heapsort

- ▶ Pop the largest element from the heap, i.e., call HEAPDELMAX ($n - 1$) times
- ▶ What's the complexity?

$O(n \log n)$