EECS 2011 M: Fundamentals of Data Structures

Suprakash Datta
Office: LAS 3043

Course page: `http://www.eecs.yorku.ca/course/2011M`
Also on Moodle

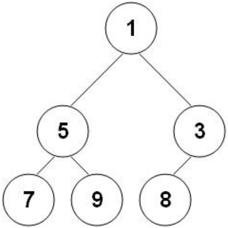
Heaps

Ch. 9.3

- Binary tree (stored in an array)
- Keys at nodes
- All nodes (except possibly the last) are complete

Note: Some slides in this presentation have been adapted from the authors' slides.

Array Representation



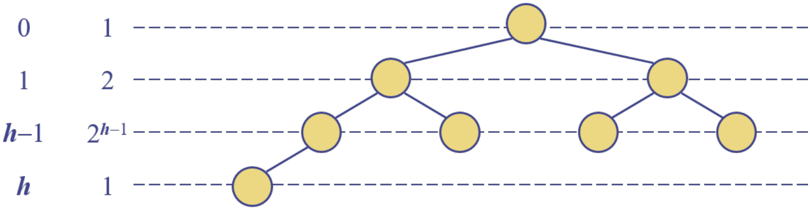
Node	1	5	3	7	9	8
Index	0	1	2	3	4	5

By Chris857. Transferred from en.wikipedia, CC0, `https://commons.wikimedia.org/w/index.php?curid=12768492`

- For the node at index/rank i
  - the left child is at index/rank $2i + 1$
  - the right child is at index/rank $2i + 2$
- add corresponds to inserting at rank $n + 1$
- Remove min – item at rank n moved to rank 1 and the heap adjusted

Heap Height

Theorem: A heap storing n keys has height $O(\log n)$
Proof: Let h be the height of a heap storing n keys
Since there are $2i$ keys at depth $i = 0, \ldots, h-1$ and at least one key at depth h, we have
$n \geq 1 + 2 + 4 + \ldots + 2^{h-1} + 1$.
Thus, $n \geq 2^h$, i.e., $h \leq \log_2 n$.

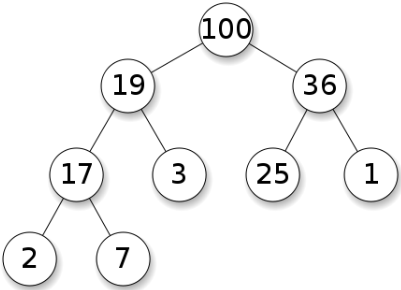


Heap Property

Min- Heaps:

- For every node v other than the root, $key(v) \leq key(parent(v))$
- The last node of a heap is the rightmost node of maximum depth

Max-heaps: $key(v) \geq key(parent(v))$



By Ermishin - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=12251273

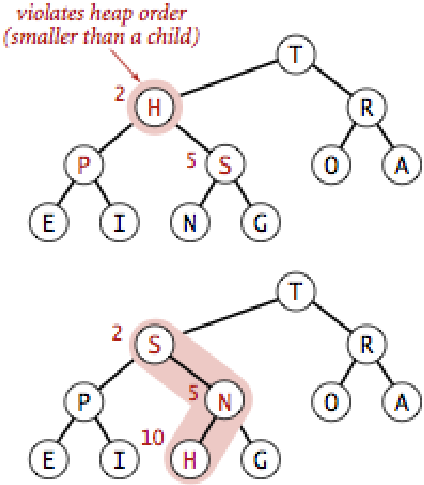
Heaps - Methods

- insert $O(\log n)$ time
- removeMin $O(\log n)$ time
- size
- isEmpty,
- extractMin

Maintaining Heap Property: Downheap

Min- Heaps:

- Restores heap when both children are heaps
- swap key k with min child
- terminates when key k reaches a leaf or a node where heap property holds



Downheap: Analysis

Correctness

- Pre-condition: Both children are heaps
- After swapping root with the min-child, same is true for the child root goes to, the other child is left unchanged.

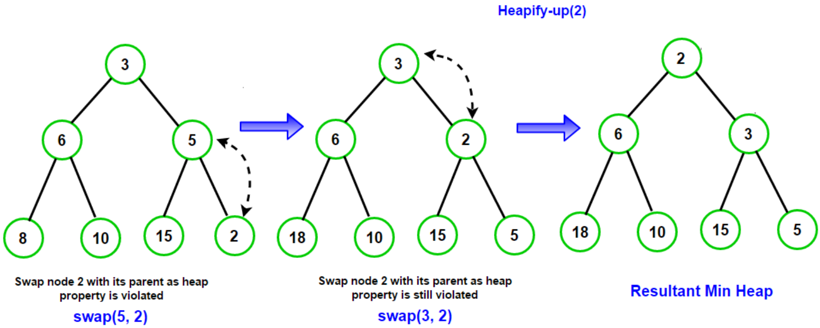
Running time

- After a constant number of steps an element travels down one level
- it travels at most the height of the tree
- Running time: $O(\log n)$

Maintaining Heap Property: Upheap

Min- Heaps:

- Restores heap when parent violates heap property
- swap key k with parent
- terminates when key k reaches the root or a node where heap property holds



Upheap: Analysis

Correctness

- Pre-condition: Both children are heaps
- After swapping node with the parent, same is true for the parent

Running time

- After a constant number of steps an element travels up one level
- it travels at most the height of the tree
- Running time: $O(\log n)$

Heap Insertion and ExtractMin

- Insert at the end of the heap
- Restore the heap using UpHeap

- Extract the top of the heap
- Delete the last node and put its key in the top node
- Restore the heap using DownHeap

Building Heaps

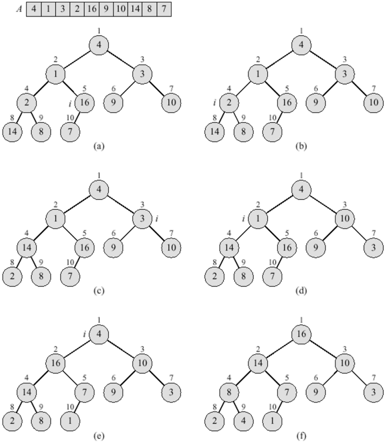
Bottom-up construction:

$\textsc{BuildHeap}(A)$

$n \leftarrow \mathit{length}[A]$

for $i = \lfloor n/2 \rfloor$ down to 1

do $\textsc{DownHeap}(A, i)$



Top-down construction:

$\textsc{RBHeap}(A, i)$

$n \leftarrow \mathit{length}[A]$

if $i \leq \lfloor n/2 \rfloor$

    then $\textsc{RBHeap}(A, 2i)$

        $\textsc{RBHeap}(A, 2i+1)$

        $\textsc{DownHeap}(A, i)$

BuildHeap: Analysis

- **Correctness:** induction on i, all trees rooted at $m > i$ are heaps
- **Running time:** less than n calls to DownHeap $= n \cdot O(\lg n) = O(n \lg n)$
    - Not a tight bound
    - Intuition: for most of the time DownHeap works on smaller than n element heaps

Bottom Up BuildHeap: Tighter Analysis

- Think of nodes at the same height as phases of the algorithm
- Assume $n = 2^k - 1$ (complete binary tree), $k = \lfloor \lg n \rfloor$
- Running time:

$$\sum_{h=1}^{k-1} h 2^{k-h} = 2^k \sum_{h=1}^{k-1} \frac{h}{2^h}$$

$\textsc{BuildHeap}(A)$

```
n ← length[A]
for i = ⌊n/2⌋ down to 1
    do DownHeap(A, i)
```

DownHeap(A, i) takes $O(ht(i))$ time, $ht(i) =$ height of subtree rooted at node i

Bottom Up BuildHeap: Analysis - 2

$$
\begin{aligned}
2^k \sum_{h=1}^{k-1} \frac{h}{2^h} &= (n+1) \sum_{h=1}^{k-1} \frac{h}{2^h} \\
&< (n+1) \sum_{h=1}^{\infty} \frac{h}{2^h} \\
&= (n+1) \left[\frac{1/2}{(1-1/2)^2} \right] \\
&= 2(n+1) \\
&\in O(n)
\end{aligned}
$$

Bottom Up BuildHeap: Analysis - 3

$$\sum_{h=0}^{\infty} x^h = \frac{1}{1-x} \text{ where } |x| < 1$$

$$\sum_{h=0}^{\infty} h x^{h-1} = \frac{1}{(1-x)^2} \text{ differentiating}$$

$$\sum_{h=0}^{\infty} h x^{h} = \frac{x}{(1-x)^2} \text{ multiplying both sides by } x$$

$$\sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{1/2}{(1-1/2)^2} = 2 \text{ substituting } x = 1/2$$

Using Heaps to Sort

- Gives an in-place sort
- $\theta(n \log n)$ running time
- Steps to sort in decreasing order:
    1. Build a min-heap from the unsorted array
    2. Keep swapping the minimum and the end of heap, decrement the size of the heap and reheapify (DownHeap).

Heapsort

```
HEAPSORT(A)
1   n ← length[A]
2   BUILDHEAP(A)
3   for i = 1 to n
4   do SWAP(A[1], A[i])
5       length(A) ← length(A) − 1
6       DOWNHEAP(A, 1)
```

Running time: line 2 takes $\theta(n)$ time. The lopp (lines 3-6) runs n times and each iteration takes $O(\log n)$ time. Total time: $O(n \log n)$.