CS472 Module 7 Part C - A Heaping Helping of Heaps

Athens State University

March 7, 2016

Outline

Contents

1 Heaps and Priority Queues

Heaps: The Notion of a Heap

- *Heap*: A binary tree with keys assigned to its nodes, one key per node, such that the following conditions hold:
  - *shape property*: the tree is essentially complete - that is all its levels are full except for possibly the last level, where only some rightmost leaves are missing
  - *heap property*: the tree shows *parental dominance*, that is the key in each node is greater than or equal to the keys in its children

Heaps: The Notion of a Heap

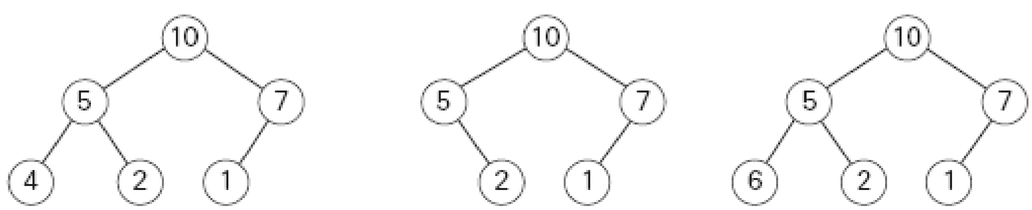

FIGURE 6.9 Illustration of the definition of heap: only the leftmost tree is a heap.

Heaps: The Notion of a Heap

- Note that key values in a heap are ordered top down
  - A sequence of values from the root to a leaf is decreasing (nonincreasing, if equal keys are allowed)

– But thre is no left-to-right order in key values

- There exists exactly one essentially complete binary tree with n nodes
  – The height of this tree is equal to the floor of $log_2(n)$
- A node of a heap and all of its descendants is also a heap

Heaps: Array representation

We can use a 1-d array to store a heap by recording the elements of the heap into the array in top-down, left-to-right fashion

- For conveinence sake, we start this process in the first element of the array, leaving the 0th element unused
- The parental node keys will be in the lower half of the array and the leaf node keys in the upper half of the array
- The children of a key in position i will be in positions $2i$ and $2i+1$ while parent of a key in position i will be found at the floor of $n/2$

Heaps: Priority queues

Note how heaps implement the priority queue abstract data type.

A *priority queue* is a multiset of items with orderable characteristic called the item's *priority*, with the following operations:

- Finding an item with the highest priority.
- Deleting an item with the highest priority
- Adding a new item to the multiset

Priority queues are painfully useful, with applications to operating systems, netowrking, and important graph algorithms

2 Heap Building

Heap Building: Bottom up

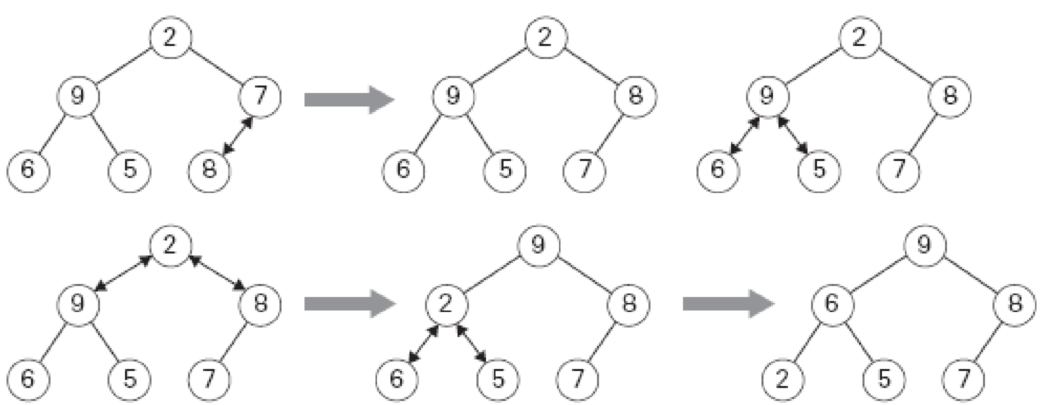

FIGURE 6.11 Bottom-up construction of a heap for the list 2, 9, 7, 6, 5, 8. The double-headed arrows show key comparisons verifying the parental dominance.

Heap Building: Bottom up

Algorithm 1: Build a heap from the bottom up

```
Input: An array H[1..n] of orderable items
Output: That array, as a heap
for i ← [floor(n/2)..1] do
    k ← i;
    v ← H[k];
    heap ← false ;
    while not heap and 2*k ≤ n do
        j ← 2*k;
        if j < n then
            if H[j] < H[j+1] then
                j ← j+1;
        if v ≥ H[j] then
            heap ← true;
        else
            H[k] ← H[j];
            k ← j;
        H[k] ← v;
```

Heap Building: Bottom up

- Assume that $n = 2^k - 1$ so that a heap's tree is full.
  - So,$h = log_2(n)$ and $log_2(n+1) - 1 = k - 1$
- In the worst case, each key on level i of the tree will travel to leaf level h
- Moving to the next level requires two comparisons: one to find a larger child and one to determine if an exchange is required
- The total number of key comparisons involving a key exchange on level i will be $2(h-i)
- So,

$$C_{worst}(n) = \sum_{i=0}^{h-1} 2(h-i)2^i = 2(n - log_2(n+1))$$

- Thus, we can create a heap in this case with less than $2n$ comparisons

Heap Building Top Down

- Successive insertion of a new key into previously constructed heap
- First, attach a new node with key K immediately after the last leaf of the existing leaf
- Sift K to the correct spot in the leaf
  - Compare K with its parent's key, if the latter is greater, then stop as you have a heap
  - Swap these two keys and compare K with its new parent
  - Keep going until you stop or you reach the root
- This process is $O(log(n))$

Heap Building: Deleting the maximum from a heap

- Exchange the root's key with that last key K of the heap
- Decrease the heap's size by 1
- Heapify the smaller tree by sifting K down the tree exactly as we did it in the bottom-up construction
  - Verify the parental dominance for K
- And this is a $O(log(n))$ operation as well

3 Heapsort

Heapsort: procedure

- *Heap construction*: Construct a heap for a given array.
- *Maximum deletion*: Apple the root-deletion operation $n-1$ times to the remaining heap

Heapsort: Connection to Selection Sort

Algorithm 2: Selection Sort

Input: An array A
Output: A sorted array Sort
for $i \leftarrow$ *[1..n]* **do**
  sort[i] $\leftarrow$ the smallest element in A;
  Delete the smallest element in A;
return sort;

Heapsort: Connection to Selection Sort

- We saw that selection sort takes $O(n^2)$ time
- But note how heapsort is just selection sort but with a heap rather than an array
- So, by doing a change in data structure, we ended up with an $O(n * log(n))$ sort