COMP 3760
Algorithm Analysis and Design

Lesson 11: Heaps and Heapsort

Rob Neilson

rneilson@bcit.ca

Today's Agenda

- Heaps
  - insertion (review)
  - heapify
  - RemoveMax
- HeapSort

Algorithm Heap.Insert(item)

From last class …

- use this algorithm to insert a new key into an existing Heap
- can be used to create a new heap in O(nlogn) time (by putting it in a loop)

```
Insert(H[1...n], item)
   insert item into last empty location
   while item > parent AND item is not root
       swap item with parent
```

- efficiency class: O($\log_2 n$)
  - in the worst case we have to swap with every parent
  - a node has a maximum $\lfloor O(\log_2 n) \rfloor$ parents

Algorithm TrickleDown(A[0...n], root)

- we use this algorithm to fix a heap at any root
- assumes the heap is stored as an array
- complete algorithm details in textbook
- Note: this algo is sometimes called “sift-down”.

- a high level description of the algorithm is:

```
TrickleDown(H[1...n], root)
   While the root has child(ren)...
      Point to roots left child
      If right child value is greater than left child ...
          ... point to the right child instead
      If the value in root is less than in child...
          ... swap the root and the child
      Make root point to its child
```

Use Heapify to build a heap

To build a heap from an arbitrary array, do this:

```
Heapify(A[1...n])
  For each parent node in the array (from bottom-right to top)
      Set root to be the next unprocessed parent
      TrickleDown(H[1..n], root)
```

Note: The textbook calls this algo “HeapBottomUp”

Efficiency (see textbook for analysis details):

- TrickleDown O(logn)
- Heapify O(nlogn)

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14
value	9	8	13	12	14	2	20	18	17	14	16	24	3	6

Algorithm Heap.RemoveMax(H)

- need to remove the max item
- this means we need a new max item as the root
- O(logn)

```
RemoveMax(H[1...n])
   move the last item to root position in heap
   call TrickleDown(H[], root)
```

HeapSort

How can we use a Heap to sort an array?

1. transform the array into a heap (use Heapify)
2. call RemoveMax to get all array elements in sorted order

```
HeapSort(A[1...n])
   heapify(A)
   while A.size is not 0
      temp ← RemoveMax(A)
      A[A.size+1] ← temp
```

- note that this algorithm is “in place” (it doesn’t require any additional memory)
- efficiency is n x O(logn) $\in$ O(nlogn)

Accessing Arbitrary Elements

- we have fast access to max in a maxHeap, or min in a minHeap
- what if we want to get the minimum item in a maxHeap?
  - we know that the smallest item will be one of the leaves … so we will have to search n/2 items, for example:

```
MaxHeap.RemoveMin(A)
   min = A[0]            // start with a big min
   for i = n/2 to n
      if A[i] < min
         min ← A[i]
   return min
```

  - note that if you wanted to remove this item, you would have to move the last item A[size] to position i, and check that the parental dominance rule still holds

- to get an arbitrary item we need to search all n elements on the heap

The End