Heapsort

COMP 215 Lecture 13

Heapsort

- Since a heap is a left complete binary tree, it can be stored in an array.
  - Entry 1 is the root.
  - $2i$ is the left child of the node at index i.
  - $2i + 1$ is the right child of the node at index i.
- Heapsort:
  - Build a valid heap starting with unordered keys: $\Theta(n)$.
  - Remove the root (largest) item n times, copying the result to an array. time?
  - It turns out that heapsort is in-place. (??)

Analysis of Key Removal

- We know that each the remove operation takes lg(n) comparisons where n is the size of the heap.
- It does not immediately follow that n such operations must require n lg(n) comparisons.
- Recall that when n is a power of 2, there is exactly one node at level d in the tree.
  - So, *no* nodes will be sifted down through d levels.
  - 2^{d-1} nodes may be sifted down through d-1 levels.
  - 2^{d-2} nodes may be sifted down through d-2 levels.
  - And so on...

Analysis of Key Removal

- Each node sifted through requires two comparison
- Leading us to this summation:

$$2\sum_{j=1}^{d-1} j\,2^{j} \;=\; 2\,n\,lg\,n - 4\mathrm{n} + 4$$

- Since the number of comparison required to create the heap was 2(*n*-1), the worst case number of comparisons for heapsort when *n* is a power of 2 is:

$$2(n-1) + 2\,n\,lg\,n - 4\mathrm{n} + 4 \;=\; 2(n\,lg\,n - n + 1) \;\in\; \Theta(n\,lg\,n)$$

Space Usage

- The same array can be used to both maintain the heap, and store the output.
- In place.