Heap Sort

CSE 373 - Data Structures
May 10, 2002

Readings and References

- Reading
  > Sections 7.5, *Data Structures and Algorithm Analysis in C*, Weiss

- Other References

Binary Search Trees for Sorting?

- Shell sort with Hibbard's increments got us to $O(N^{1.5})$
- Can we beat $O(N^{1.5})$ using a BST to sort $N$ elements?
  > Insert each element into an initially empty BST
  > Do an In-Order traversal to get sorted output
- Running time:
  > $N$ Inserts at $O(\log N)$ apiece = $O(N \log N)$
  > plus $O(N)$ for In-Order traversal
  > $O(N \log N)$ total which is $o(N^{1.5})$

Binary Search Tree sort issue

- Extra Space
  > Need to allocate space for tree nodes and pointers
  > $O(N)$ extra space, not *in place* sorting
- What if the tree is complete, and we use an array representation – can we sort in place?
  > Recall your favorite data structure with the initials B. H.
Binary Heaps

- A binary heap is a binary tree that is:
  - Complete: the tree is completely filled except possibly the bottom level, which is filled from left to right
  - Satisfies the heap order property
    - every node is less than or equal to its children
    - or every node is greater than or equal to its children
  - The root node is always the smallest node
    - or the largest, depending on the heap order

Heap order property

- A heap provides limited ordering information
- Each path is sorted, but the subtrees are not sorted relative to each other
  - A binary heap is NOT a binary search tree

Structure property

- A binary heap is a complete tree
  - All nodes are in use except for possibly the right end of the bottom row
- Array implementation is compact because we know how many children there are and we know that they are all present
  - no pointers are needed, we can directly calculate subscript offsets to the nodes of the tree

Heap Sort using an array

- Root node = $A[0]$
- Keep track of current size $N$ (number of nodes)
Using Binary Heaps for Sorting

- Build a **max-heap**
- Do N **DeleteMax** operations and store each Max element as it comes out of the heap
- Data comes out in largest to smallest order
- Where can we put the elements as they are removed from the heap?

1 Removal = 1 Addition

- Every time we do a DeleteMin, the heap gets smaller by one node, and we have one more node to store
  - Store the data at the end of the heap array
  - Not "in the heap" but it is in the heap array

Heap Sort is In-place

- After all the DeleteMins, the heap is gone but the array is full and is in sorted order
- Note that this heap implementation uses index 0 for data and has no sentinel value

Heapsort: Analysis

- Running time
  - time to build max-heap is \(O(N)\)
  - time for N DeleteMax operations is \(N O(\log N)\)
  - total time is \(O(N \log N)\)
- Can also show that running time is \(\Omega(N \log N)\) for some inputs,
  - so worst case is \(\Theta(N \log N)\)
  - Average case running time is also \(O(N \log N)\)