Faster Sorting Methods (Ch7-9)

Philip Chan

1 Mergesort (Sec 8.1-3; skip 8.2)
   - Keep splitting the array into two equal halves until 1 or two items are left (handout and Program 8.3, p354)
   - two sorted halves are merged (handout and Program 8.1, p349)
   - Worst-case analysis: \(O(N \log N)\) comparisons
   - optimal for sorting in number of comparisons
   - Average-case analysis: \(O(N \log N)\) comparisons
   - need \(O(N)\) extra space
   - Stable alg—as long as items in the left half have priority over the ones with the same key in the right half during merging

2 Quicksort (Sec 7.1-2)
   - Partition the items into two groups: smaller or bigger than the partitioning element (pivot) (Program 7.2, p319)
     - lo (l), hi (r)
     - loswap (i), hiswap (j)
     - pick pivot: \(a[hi]\) or \(a[r]\) (pivot could be any key)
     - repeat
       - scan from left to right to find a key larger than the pivot (loswap)
       - scan from right to left to find a key smaller than the pivot (hiswap)
       - swap \(a[\text{loswap}]\) and \(a[\text{hiswap}]\)
     - until loswap and hiswap cross
       - swap \(a[\text{loswap}]\) and \(a[\text{hi}]\)
   - Recursively quicksort on the two partitions (Program 7.1, p317)
   - Worst-case analysis: \(O(N^2)\) comparisons
   - Average-case analysis: \(O(N \log N)\) comparisons
   - Unstable alg—in the partitioning step...
   - Quite popular because it doesn’t need extra space and the average case is \(O(N \log N)\)

   We will come back after we discuss trees.
3 Sorting with BST’s (sometimes known as tree sort)

- build the BST by $N$ insertions: worst-case = $O(N^2)$ comparisons, average-case (random input) = $O(N \log N)$ comparisons
- inorder traversal of the BST: worst-case or average-case = $N$ nodes to visit

4 Priority Queues and Sorting (Sec 9.1)

- a queue arranged by priority of the items (normal queue has time of arrival as priority)
- operations: insert, removeMax
- how do we sort with priority queues?
- simple implementations (Table 9.1, p.379)

5 Heaps (Sec 9.2)

- heap-ordered tree: the key at a node is larger than the keys at its children
- heap: a complete binary tree that is heap-ordered
- complete tree: no empty internal nodes and all nodes at the bottom are on the left
- array representation of a complete binary tree: node is the index of a node with root as index 1 [root is index 0]
  - $parent = \text{node}/2$
  - $leftChild = 2 \times \text{node}$
  - $rightChild = 2 \times \text{node} + 1$
- path from root to all leaves is at most $\log N$, where $N$ is the number of tree nodes

6 Heap Algorithms (Sec 9.3)

- promotion: if a child is larger than its parent, the child is promoted to be the parent, and the parent is demoted to be the child
- insert (Program 9.5, p.386):
  - put the item at the end of the array
  - Bottom-up Heapify (Figure 9.3; Program 9.3, p.384)
  - if parent is smaller, swap the current node with the parent
  - Why not insert at the root and push smaller keys down? Will the tree be complete?
- removeMax (Program 9.5, p.386):
  - get the key at the root
  - replace key at the root with the last array element
  - Top-down Heapify (Figure 9.4; Program 9.5, p.385)
  - if the larger child is larger than the current node, swap
  - Why not just promote the larger child to the empty parent? Will the tree be complete?
- Worst-case insert: $O(\log N)$ comparisons (Property 9.2)
- Worst-case removeMax: $O(\log N)$ comparisons (Property 9.2)
- Sorting with a priority queue (Program 9.6, p.388): $N$ insert and $N$ removeMax: $O(N \log N)$
7 Heapsort (Sec 9.4)

- put all the items in the array before heapifying
- Use Top-down Heapify (fixDown) on each node starting from one level above the leaves, and go to each node in reverse array order
- while the heap has more than one key
  - swap the root with the last array element (the heap is one node smaller) [Selection Sort?]
  - top-down heapify at the root
- worst-case:
  - initial heap construction: \( O(N) \) comparisons
    - For example, a heap of 127 keys
    - 32 heaps of size 3, 16 heaps of size 7, 8 heaps of size 15...
    - Number of promotions = \( 32 \times 1 + 16 \times 2 + 8 \times 3 + 4 \times 4 + 2 \times 5 + 1 \times 6 = 120 \)
    - Let \( N = 2^n - 1 \)
    - Number of promotions: \( \sum_{l \leq k < n} k2^{n-k-1} = 2^n - n - 1 \) (Property 9.4, p.391)
    - extra credit for HW5 (10 points): derive Property 9.4 and use proof by induction to verify it
    - \( 2^n - n - 1 < N \)
    - each promotion needs 2 comparisons
  - removeMax: \( O(N \lg N) \), \( N - 1 \) max’s, each \( O(\lg N) \) comparisons
    - total: \( O(N \lg N) \) comparisons
- Mergesort needs extra space
- Quicksort and sorting with a BST have a worst case of \( O(N^2) \) if the keys are sorted
- Heapsort doesn’t need extra space nor has a worst case of \( O(N^2) \)
- Quicksort is usually faster for random keys
- skip p.393 and the rest of the chapter