Heap-Sort

COMPSCI 355
Fall 2016
Metrics for Sorting Algorithms

- worst-case asymptotic performance
- average-case asymptotic performance
- stability
- in-place
- behavior on partially sorted data
Agenda

- Heap Sort
- Merge Sort
- Quick Sort
- Radix Sort
- Theoretical lower bounds
- The selection problem

theoretically optimal
Heap-Sort

Heap-Sort algorithm demonstrates the sorting process using a binary heap data structure. The heap is a complete binary tree where each node is greater than or equal to its children. In the diagram, the numbers represent the heap structure after several steps of the sorting algorithm.
Heap-Sort

50  68  45  63  39  7  43  60  97
0   1   2   3   4   5   6   7   8

50

68       45

63  39  7  43

60
Heap-Sort

<table>
<thead>
<tr>
<th>50</th>
<th>68</th>
<th>45</th>
<th>63</th>
<th>39</th>
<th>7</th>
<th>43</th>
<th>60</th>
<th>97</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Reheapify with downheap bubbling.
Heap-Sort

<table>
<thead>
<tr>
<th>68</th>
<th>50</th>
<th>45</th>
<th>63</th>
<th>39</th>
<th>7</th>
<th>43</th>
<th>60</th>
<th>97</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Reheapify with downheap bubbling.
Heap-Sort

Reheapify with downheap bubbling.
Heap-Sort

HEAP PROPERTY RESTORED
Heap-Sort Complexity

- Initial heap construction:
- Each reheapification:
- Total time:
Heap-Sort Complexity

- Initial heap construction: $O(n \log n)$
- Each reheapification: $O(\log n)$
- Total time: $O(n \log n)$
- Stable?
- In place?
Bottom-Up Construction

- Idea: work from the bottom up, restoring the heap property at each node as you go.
Bottom-Up Construction

- Idea: work from the bottom up, restoring the heap property at each node as you go.

Each leaf is by itself a heap.
Bottom-Up Construction

- Idea: work from the bottom up, restoring the heap property at each node as you go.
Bottom-Up Construction

- Idea: work from the bottom up, restoring the heap property at each node as you go.
Bottom-Up Construction

- Idea: work from the bottom up, restoring the heap property at each node as you go.
**Bottom-Up Construction**

- Idea: work from the bottom up, restoring the heap property at each node as you go.
Bottom-Up Construction

- Idea: work from the bottom up, restoring the heap property at each node as you go.
Bottom-Up Construction

- Idea: work from the bottom up, restoring the heap property at each node as you go.
Bottom-Up Construction

• Idea: work from the bottom up, restoring the heap property at each node as you go.
Bottom-Up Construction

- Idea: work from the bottom up, restoring the heap property at each node as you go.

Now we have a heap!
Recursive Implementation

- First, construct two heaps recursively.
Recursive Implementation

- First, construct two heaps recursively.
Recursive Implementation

- Then perform downheap bubbling on root.
Recursive Implementation

• Then perform downheap bubbling on root.
Recursive Implementation

- Then perform downheap bubbling on root.
Recursive Implementation

- Reheapified.
- Complexity?
Bottom-Up Construction

- Let $P(x)$ denote the path that a node $x$ takes during downheap bubbling.
- Heap construction takes time on the order of

$$\sum_{x \in S} |P(x)|$$

where $S$ is the set of internal nodes.
For each internal node $x$, we define its *proxy path* $Q(x)$ to be the path to its in-order successor.

Where is this node's in-order successor?
Complexity

- For any node $x$, $|P(x)| \leq |Q(x)|$.
- The proxy paths are disjoint.
Complexity

• For any node \( x \), \( |P(x)| \leq |Q(x)| \).
• The proxy paths are disjoint.
Complexity

- For any node $x$, $|P(x)| \leq |Q(x)|$.
- The proxy paths are disjoint.
Complexity

• For any node $x$, $|P(x)| \leq |Q(x)|$.
• The proxy paths are disjoint.
Complexity

- For any node $x$, $|P(x)| \leq |Q(x)|$.
- The proxy paths are disjoint.
Complexity

• For any node \( x \), \( |P(x)| \leq |Q(x)| \).
• The proxy paths are disjoint.
Complexity

- For any node $x$, $|P(x)| \leq |Q(x)|$.
- The proxy paths are disjoint.
Complexity

- Heap construction takes time on the order of
  \[ \sum_{x \in S} |P(x)| \]
  where \( S \) is the set of internal nodes.

- But since the proxy paths are disjoint:
  \[ \sum_{x \in S} |P(x)| \leq \sum_{x \in S} |Q(x)| < |E| \]
Complexity

- Proportional to number of edges.
- How many edges are in heap of $n$ nodes?
Complexity

• Proportional to number of edges.
• How many edges are in heap of $n$ nodes?
• A tree of $n$ nodes has $n-1$ edges.
• Bottom-up construction: linear time.