Heapsort (II)

Textbook, Chapter 7

CSCE310: Data Structures and Algorithms
www.cse.unl.edu/~choueiry/S01-310/

Berthe Y. Choueiry (Shu-we-ri)
Ferguson Hall, Room 104
choueiry@cse.unl.edu, Tel: (402)472-5444
**Heapsort**: Principle and algorithm

Input: Array $A[1 \ldots n]$, where $n = \text{length}[A]$

1. Build a heap out of the array
   the maximum element will be in $A[1]$, it should be last!!


3. Reset heap size to $(n - 1)$

4. Heapify $A[1 \ldots (n - 1)]$, repeat... down to a heap of size 2

\[\text{Heapsort}(A)\]
1. \(\text{Build-Heap}(A)\)
2. for $i \leftarrow \text{length}[A]$ downto 2
4. \(\text{heap-size}[A] \leftarrow \text{heap-size}[A] - 1\)
5. \(\text{Heapiify}(A, 1)\)
Heapsort: Example $A = \langle 4, 1, 3, 2, 16, 9, 10, 14, 8, 7 \rangle$
Heapsort: Example $A = \langle 4, 1, 3, 2, 16, 9, 10, 14, 8, 7 \rangle$
**Heapsort:** Complexity

For an array $A$ of size $n$

- **Build-Heap** is called once, its complexity is $O(n)$
- **Heapify** is called $n - 1$ times, its complexity is $O(lg n)$
- Complexity of **Heapsort** is $O(n + (n - 1) lg n) = O(n lg n)$

**Heapsort:** Utility

- **Heapsort** is an excellent algorithm, however **Quicksort** is better in practice

- Heap data structure has enormous utility e.g., efficient priority queue
Priority queue as a data structure

• Data structure for maintaining a set $S$ of elements, each with a value called key

• Supports operations:
  \( \text{Insert}(S, x) \) (i.e., \( S \leftarrow S \cup \{x\} \))
  \( \text{Maximum}(S) \) returns element of \( S \) with largest key
  \( \text{Extract-Max}(S) \) removes and returns the element of \( S \) with largest key
**Priority queue: Applications**

1. Job scheduling on a shared resource (e.g., computer)
   Priority queue keeps track of jobs and their relative importance.
   Insert allows us to add new jobs at any time
   Extract-Max selects highest priority jobs when current job is finished/interrupted

2. Event-driven simulator
   Queue of events to be simulated, each has occurrence time →
   key value
   Occurrence of event causes other events to be simulated in future → Insert Extract-Min and Minimum instead of
   Extract-Max and Maximum

Implementation → a heap
**Heap-Extract-Max**($A$)

If heap-size[$A$] < 1
    then error “heap underflow”

max ← $A[1]$


heap-size[$A$] ← heap-size[$A$] − 1

Heapify($A, 1$)

return $max$

→ Returns $A[1]$
→ Places the last element of the heap in $A[1]$
→ Decrements size of heap
→ Heapifies the array
→ Running time is in $O(lg \ n)$ (constant effort and 1 heapify)
**Heap-Insert**\((A, key)\)

\[
\text{heap-size}[A] \leftarrow \text{heap-size}[A] + 1
\]

\[
i \leftarrow \text{heap-size}[A]
\]

while \(i > 1\) and \(A[\text{Parent}(i)] < key\)
  do 
    \[
    A[i] \leftarrow A[\text{Parent}(i)]
    \]
    \[
    i \leftarrow \text{Parent}(i)
    \]
  \[
  A[i] \leftarrow key
  \]

→ First expands the heap by adding 1 new leaf
→ Then traverses path from new leaf to root to find
  a proper place for the new element
**Heap-Insert** \((A, key)\): Example

\(key = 15\)

Running time is \(O(\lg n)\), since path from new leaf to root has length \(\lg n\)
Heap as a priority queue

For any set $S$ of $n$ elements:

- $\text{Heap-Maximum}(A)$ is in $\Theta(1)$
- $\text{Heap-Extract-Max}(A)$ is in $O(lg n)$
- $\text{Heap-Insert}(A, key)$ is in $O(lg n)$
**Bubble-sort**

- Go through the list in order, swapping two elements if their keys are out of order
- Repeat until no swaps are performed. The list is sorted
- \( n \) passes suffice
- Similar to Insertion-sort, but has lots of swaps

**Bubble-Sort** (\( A \))
For \( i \rightarrow n - 1 \) downto 1
  For \( j \rightarrow 1 \) to \( i \)
<table>
<thead>
<tr>
<th>Name</th>
<th>Upper bound</th>
<th>Lower bound</th>
<th>Tight bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection-sort</td>
<td>$O(n^2)$</td>
<td>$\Omega(n)$</td>
<td></td>
</tr>
<tr>
<td>Insertion-sort</td>
<td>$O(n^2)$</td>
<td>$\Omega(n)$</td>
<td></td>
</tr>
<tr>
<td>Bubble-sort</td>
<td>$O(n^2)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Merge-sort</td>
<td></td>
<td></td>
<td>$\Theta(n \lg n)$</td>
</tr>
<tr>
<td>Heap-sort</td>
<td>$O(n \lg n)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quick-sort</td>
<td>$O(n^2)$</td>
<td>$\Omega(n \lg n)$</td>
<td></td>
</tr>
</tbody>
</table>