Lecture 8: Sorting I: Heaps

Lecture Overview

- Review: Insertion Sort and Merge Sort
- Selection Sort
- Heaps

Readings

CLRS 2.1, 2.2, 2.3, 6.1, 6.2, 6.3 and 6.4

Sorting Review

Insertion Sort

Merge Sort

Divide $n$-element array into two subarrays of $n/2$ elements each. Recursively sort sub-arrays using mergesort. Merge two sorted subarrays.
In-Place Sorting

Numbers re-arranged in the array $A$ with at most a constant number of them sorted outside the array at any time.

Insertion Sort: stores key outside array $\Theta(n^2)$ in-place

Merge Sort: Need $O(n)$ auxiliary space $\Theta(n \lg n)$ during merging

*Question:* Can we have $\Theta(n \lg n)$ in-place sorting?

Selection Sort

0.  $i = 1$

1.  Find minimum value in list beginning with $i$

2.  Swap it with the value in $i^{th}$ position

3.  $i = i + 1$, stop if $i = n$

Iterate steps 0-3 $n$ times. Step 1 takes $O(n)$ time. Can we improve to $O(\lg n)$?
Heaps (Not garbage collected storage)

A heap is an array object that is viewed as a nearly complete binary tree.

Data Structure

- root $A[i]$
- Node with index $i$

- $\text{PARENT}(i) = \lfloor \frac{i}{2} \rfloor$
- $\text{LEFT}(i) = 2i$
- $\text{RIGHT}(i) = 2i + 1$

Note: NO POINTERS!
length[A]: number of elements in the array

heap-size[A]: number of elements in the heap stored within array A

heap-size[A]: \leq \text{length}[A]

Max-Heaps and Min-Heaps

Max-Heap Property: For every node $i$ other than the root $A[\text{PARENT}(i)] \geq A[i]$

Height of a binary heap $O(\lg n)$

\text{MAX\_HEAPIFY}: O(\lg n) maintains max-heap property

\text{BUILD\_MAX\_HEAP}: O(n) produces max-heap from unordered input array

\text{HEAP\_SORT}: O(n \lg n)

Heap operations insert, extract\_max etc $O(\lg n)$.

Max\_Heapify(A,i)

\begin{align*}
l & \leftarrow \text{left}(i) \\
r & \leftarrow \text{right}(i) \\
& \text{if } l \leq \text{heap-size}(A) \text{ and } A[l] > A[i] \\
& \quad \text{then } \text{largest} \leftarrow l \\
& \quad \text{else } \text{largest} \leftarrow i \\
& \text{if } r \leq \text{heap-size}(A) \text{ and } A[r] > \text{largest} \\
& \quad \text{then } \text{largest} \leftarrow r \\
& \text{if } \text{largest} \neq i \\
& \quad \text{then exchange } A[i] \text{ and } A[\text{largest}] \\
& \quad \text{MAX\_HEAPIFY}(A, \text{largest})
\end{align*}

This assumes that the trees rooted at $\text{left}(i)$ and $\text{Right}(i)$ are max-heaps. $A[i]$ may be smaller than children violating max-heap property. Let the $A[i]$ value “float down” so subtree rooted at index $i$ becomes a max-heap.
Example

MAX_HEAPIFY (A,2)
heap_size[A] = 10

Call MAX_HEAPIFY(A,4)
because max_heap property
is violated

No more calls

Figure 5: MAX HEAPIFY Example