6.006 Introduction to Algorithms
Spring 2008

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

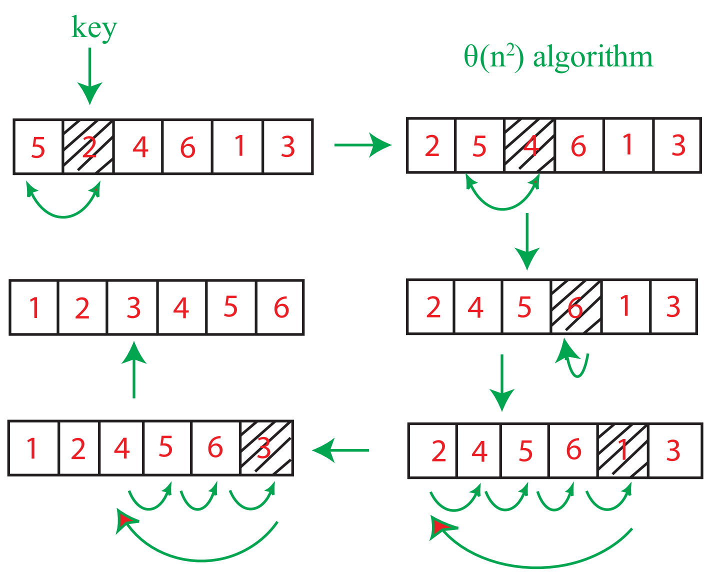

Figure 1: Insertion Sort Example

Merge Sort

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

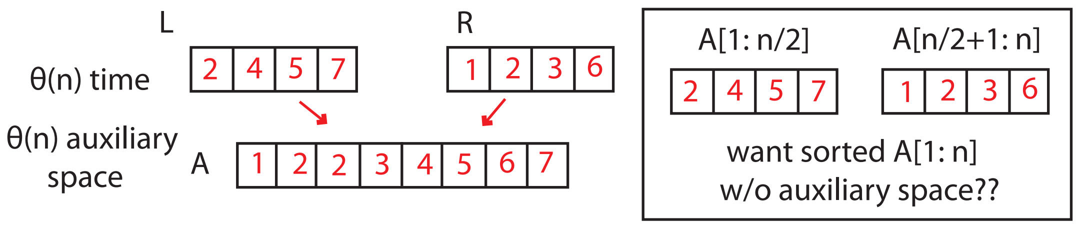

Figure 2: Merge Sort Example

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 $O(\lg n)$?

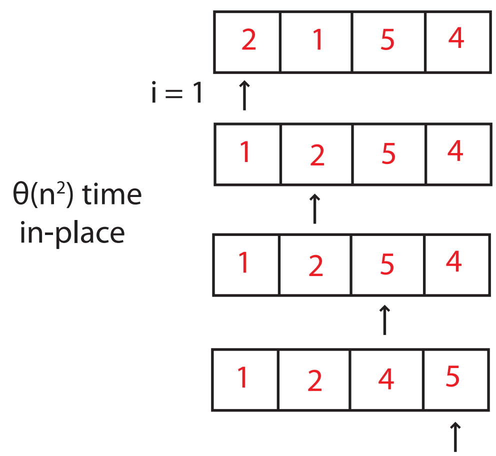

Figure 3: Selection Sort Example

Heaps (Not garbage collected storage)

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

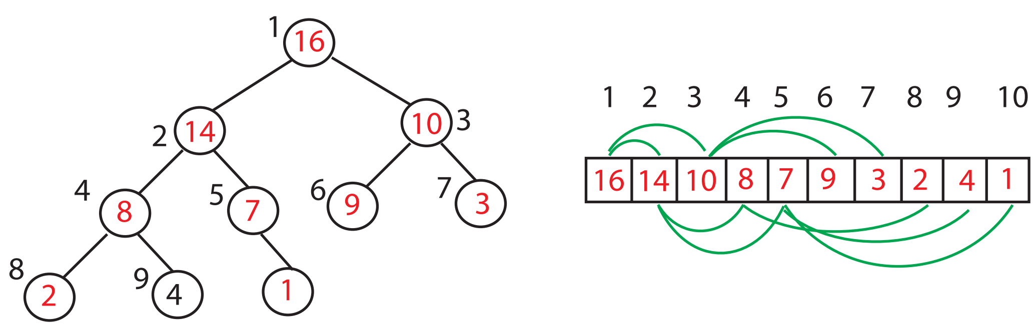

Figure 4: Binary Heap

Data Structure

root $A[i]$
Node with index i
PARENT(i) $= \lfloor \frac{i}{2} \rfloor$
LEFT(i) $= 2i$
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]: $\leq$ length[A]

Max-Heaps and Min-Heaps

Max-Heap Property: For every node i other than the root A[PARENT(i)] $\geq A[i]$
Height of a binary heap $O(\lg n)$

MAX_HEAPIFY: $O(\lg n)$ maintains max-heap property

BUILD_MAX_HEAP: $O(n)$ produces max-heap from unordered input array

HEAP_SORT: $O(n \lg n)$

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

Max_Heapify(A,i)

```
l ← left(i)
r ← right(i)
if l ≤ heap-size(A) and A[l] > A[i]
   then largest ← l
   else largest ← i
if r ≤ heap-size(A) and A[r] > largest
   then largest ← r
if largest ≠ i
   then exchange A[i] and A[largest]
        MAX_HEAPIFY(A, largest)
```

This assumes that the trees rooted at left(i) and 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

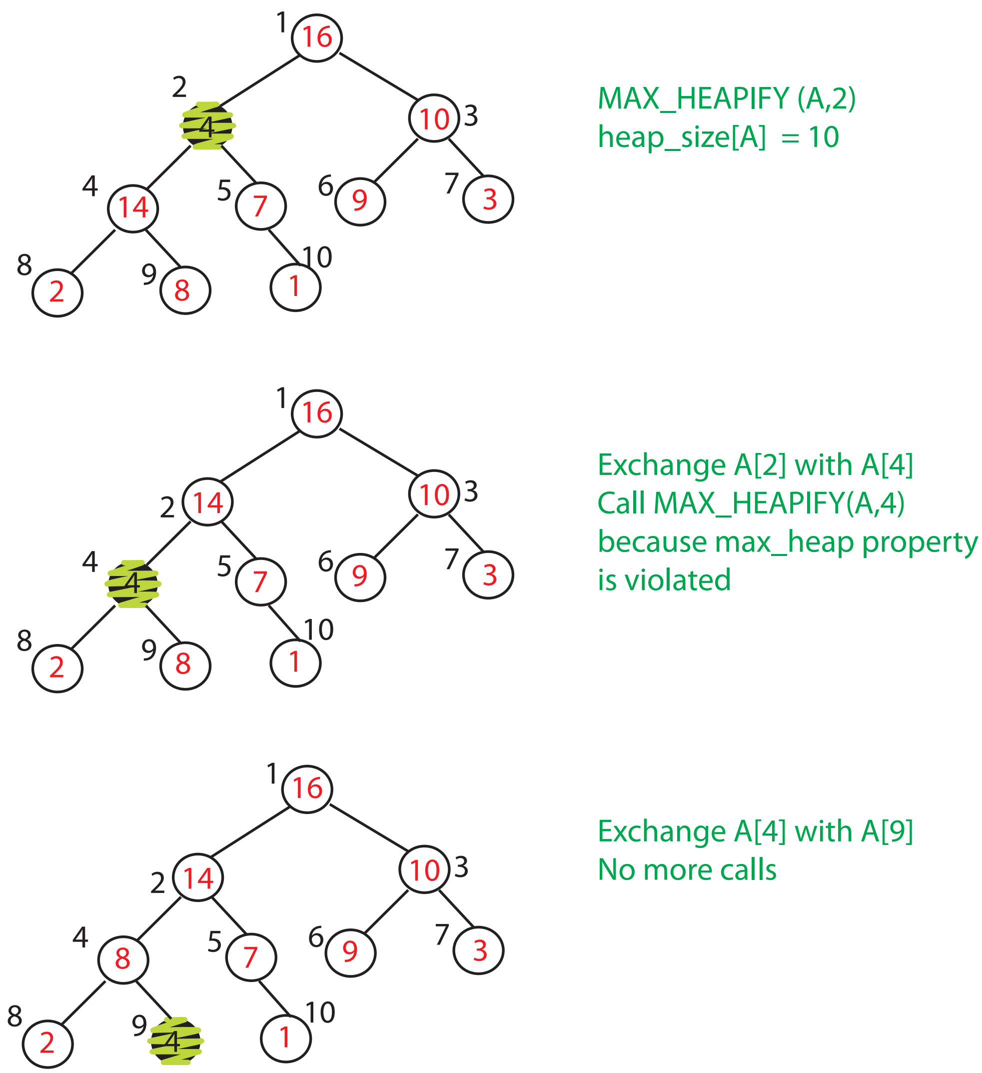

Figure 5: MAX HEAPIFY Example