CS 240: Data Structures and Data Management
Module 2 Study Guide

Taylor J. Smith — Spring 2017

Key Concepts

- A **priority queue** is a collection of items with associated priorities.
- Operations for priority queues:
  - Insert: inserts item
  - DeleteMax/DeleteMin: deletes item with max/min priority
- A **max-heap** is a tree with two properties: **structural property** (all levels, except possibly the last, are filled and the last level is left-justified) and **heap-order property** (for all keys i, priority of the parent of i $\geq$ priority of i).
- A **min-heap** is the same as a max-heap structurally, but the heap-order property is reversed ($\leq$).
- Building heaps:
  - Start with empty heap, insert items one at a time – $\Theta(n \log(n))$
  - Use bubble-downs – $\Theta(n)$
- Operations for heaps:
  - Insert: add item to leftmost empty spot, bubble-up if needed – $O(\log(n))$
  - DeleteMax/DeleteMin: swap root and rightmost leaf, remove, bubble-down if needed – $O(\log(n))$
- Sorting using priority queues/heaps (PQ-sort/heapsort) – $O(n \log(n))$

Suggested Readings

- **Sedgewick:** 5.5 (Mathematical Properties of Trees), 9.1 (Elementary Implementations), 9.2 (Heap Data Structure), 9.3 (Algorithms on Heaps), 9.4 (Heapsort)
- **CLRS:** Chapter 6 (Heapsort)
- **Goodrich/Tamassia:** 2.4 (Priority Queues and Heaps)

Practice Questions

Sedgewick

9.1. A letter means *insert* and an asterisk means *remove the maximum* in the sequence

P R I O * R * * I * T * Y * * * Q U E * * * U * E.

Give the sequence of values returned by the *remove the maximum* operations.

9.3. Explain how to use a priority queue ADT to implement a stack ADT.

9.4. Explain how to use a priority queue ADT to implement a queue ADT.

9.17. Is an array that is sorted in descending order a heap?

9.33. For $n = 32$, give an arrangement of keys that makes heapsort use as many comparisons as possible.

9.34. For $n = 32$, give an arrangement of keys that makes heapsort use as few comparisons as possible.

CLRS

6.1-1. What are the minimum and maximum numbers of elements in a heap of height h?

6.1-4. Where in a max-heap might the smallest element reside, assuming that all elements are distinct?

6.1-6. Is the sequence $\langle 23, 17, 14, 6, 13, 10, 1, 5, 7, 12\rangle$ a max-heap?

6.2-5. The code for MAX-HEAPIFY is quite efficient in terms of constant factors, except possibly for the recursive call in line 10, which might cause some compilers to produce inefficient code. Write an efficient MAX-HEAPIFY that uses an iterative control construct (a loop) instead of recursion.

```
MAX-HEAPIFY(A, i, n)
 l = LEFT(i)
 r = RIGHT(i)
 if l ≤ n and A[l] > A[i]
     largest = l
 else largest = i
 if r ≤ n and A[r] > A[largest]
     largest = r
 if largest ≠ i
     exchange A[i] with A[largest]
     MAX-HEAPIFY(A, largest, n)
```

6.2-6. Show that the worst-case running time of MAX-HEAPIFY on a heap of size n is $\Omega(\log(n))$. (*Hint:* For a heap with n nodes, give node values that cause MAX-HEAPIFY to be called recursively at every node on a path from the root down to a leaf.)

6.4-1. Using Figure 6.4 as a model, illustrate the operation of heapsort on the array $A = \langle 5, 13, 2, 25, 7, 17, 20, 8, 4\rangle$.

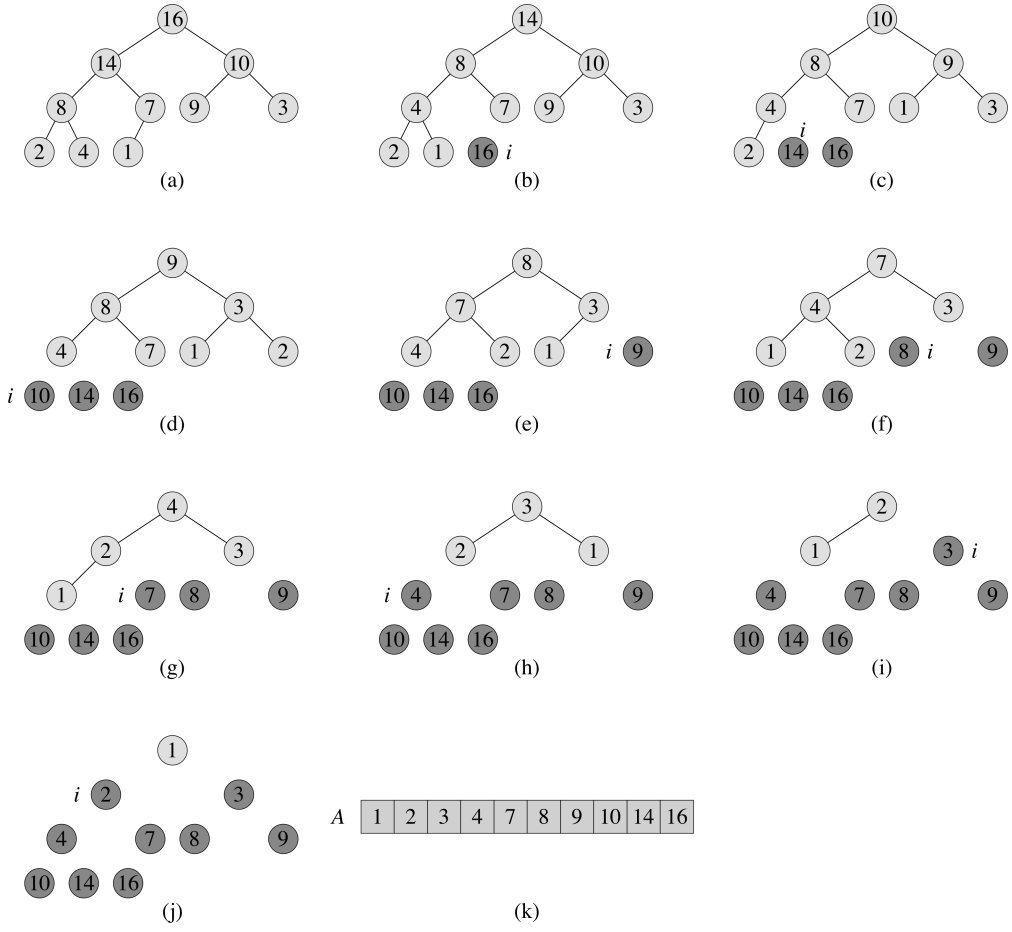

Figure 6.4: An example of the operation of heapsort after the max-heap is initially built.

6.4-3. What is the running time of heapsort on an array A of length n that is already sorted in increasing order? What about decreasing order?

6.4-4. Show that the worst-case running time of heapsort is $\Omega(n \log(n))$.

6-2. A *d-ary heap* is like a binary heap, but (with one possible exception) non-leaf nodes have d children instead of 2 children.

(a) How would you represent a d-ary heap in an array?

(b) What is the height of a d-ary heap of n elements in terms of n and d?

(c) Give an efficient implementation of DELETEMAX in a d-ary max-heap. Analyze its running time in terms of d and n.

(d) Give an efficient implementation of INSERT in a d-ary max-heap. Analyze its running time in terms of d and n.

(e) Give an efficient implementation of HEAP-INCREASE-KEY, which first sets $A[i] \leftarrow \max(A[i], key)$ and then updates the d-ary max-heap structure appropriately. Analyze its running time in terms of d and n.

```
HEAP-INCREASE-KEY(A, i, key)
if key < A[i]
    error "new key is smaller than current key"
A[i] = key
while i > 1 and A[PARENT(i)] < A[i]
    exchange A[i] with A[PARENT(i)]
    i = PARENT(i)
```

Goodrich/Tamassia

R-2.14. Is there a heap T storing seven distinct elements such that a preorder traversal of T yields the elements of T in sorted order? How about an inorder traversal? How about a postorder traversal?