Data Structures and Algorithms in Python

Michael T. Goodrich
Department of Computer Science
University of California, Irvine

Roberto Tamassia
Department of Computer Science
Brown University

Michael H. Goldwasser
Department of Mathematics and Computer Science
Saint Louis University

Study Guide: Hints to Exercises

WILEY

Chapter 9 Priority Queues

Hints

Reinforcement

R-9.1) Each remove_min( ) operation takes $O(\log n)$ time.

R-9.2) Observe that the keys are guaranteed to satisfy the heap-order property. What other property does T need to satisfy in order to be a heap?

R-9.3) Use a simple list-based priority queue and a pencil with a good eraser.

R-9.4) There is a very good reason this exercise appears in this chapter.

R-9.5) Be prepared!

R-9.6) Sounds too good to be true.

R-9.7) Mimic the illustration style used in the book.

R-9.8) Mimic the illustration style used in the book.

R-9.9) Think about where insertion-sort has to put each added element and design your sequence so that insertion-sort has to put each next element as far as possible.

R-9.10) Where might the second smallest key be?

R-9.11) If the smallest is at the top of the heap...

R-9.12) Consider transforming the keys to invert their order.

R-9.13) Mimic the illustration style used in the book for insertion-sort and selection-sort.

R-9.14) Consider the heap-order property and the definition of the level number of a node in a tree.

R-9.15) Recall the definition of a complete binary tree.

R-9.16) The answers are "yes,no,yes." Now all you have to do is to give examples for the yeses and a reason for the no.

R-9.17) The preorder sequence starts out 0, 1, 3, 7, ...

R-9.18) Consider the last $n/2$ terms in this sum.

R-9.19) Try to construct a heap that has larger elements in left subtrees.

R-9.20) Try to construct a heap that has larger elements in right subtrees.

R-9.21) Mimic the illustration style used in the book.

R-9.22) Mimic the illustration style used in the book.

R-9.23) You need to be very careful about how you partition the keys between the subtrees rooted at the children of the root.

R-9.24) Structure the insertions so that each requires lots of down-heap bubbling.

R-9.25) Mimic the illustration style used in the book.

Creativity

C-9.26) Figure out a way to time stamp the entries in the priority queue.

C-9.27) Figure out a way to time stamp the entries in the priority queue.

C-9.28) Is it ever possible that a new element gets a key that is strictly smaller than a previously inserted element?

C-9.29) Beware: `pop(0)` is expensive for a Python list.

C-9.30) Use a loop.

C-9.31) Use a loop.

C-9.32) Do simple up-and-down searches in the tree to locate the last node each time.

C-9.33) Think about what changes need to be made when leaves are created or destroyed.

C-9.34) Consider the binary expansion of $n-1$, n, and $n+1$.

C-9.35) Note that the entries do not need to be reported in sorted order. Use binary recursion on the subtrees of the heap and think about where the keys smaller than k are stored in the heap T.

C-9.36) Think carefully about how location-aware entries can be implemented efficiently.

C-9.37) Use Calculus.

C-9.38) Study the combine step in the bottom-up heap construction algorithm.

C-9.39) Make sure you understand the proper semantics when the parameter is smaller then all heap elements.

C-9.40) Make sure you understand the proper semantics when the parameter is smaller then all heap elements.

C-9.41) Use a suitably constructed heap.

C-9.42) Start by using the bottom-up construction.

C-9.43) Process elements one at a time, always storing the largest k that you have seen.

C-9.44) Create a new nested key type that wraps the provided keys to invert comparisons.

C-9.45) Write a short function that computes the number of 1's in the binary expansion of an integer by using the bitwise "and" operation.

C-9.46) Rather than using elements as keys in the priority queue, use the result of the key function for each element.

C-9.47) Partition the array into a sorted part and an unsorted part and use swaps to move elements around.

C-9.48) Partition the array into a sorted part and an unsorted part and use swaps to move elements around.

C-9.49) Use the right portion of the array to store the heap.

C-9.50) You will need two priority queues.

C-9.51) Use adaptable priority queues.

C-9.52) You will need two data structures that somehow keep "links" between each other.

Projects

P-9.53) Use as large of inputs as you can for experimentation.

P-9.54) Experiment with for which values of k your new implementation outperforms the original.

P-9.55) Use a heap for the priority queue.

P-9.56) Study again the bottom-up heap construction algorithm.

P-9.57) Use a heap for the CPU job priority queue.

P-9.58) Decide early how you are going to implement the "pointers" for your location-aware entries.