Which of the following is false?

A. PQ is an ADT where the element with the highest priority is removed first. ("queue with priority")

B. Heap is a tree where every parent has a higher priority than its children.

C. PQ can be implemented with a Heap that is implemented with a Vector.

D. They are all correct

E. Whatever
Administrative Details

• Lab 8 Posted: Super Lexicon
  • Implement a Trie data structure
    • A tree of letters
  • Efficiently solve a problem using trees that are more interesting than a simple binary tree
Today’s Outline

• Heap
  • Basics
  • Heapify
  • Heapsort
Heap

• A heap is a tree “sorted top to bottom”:
  • any parent has a higher priority than it’s children
    • Heap invariant: value <= values of children
  • Recursive definition:
    • Root holds the highest priority value
    • Subtrees are also heaps

• Not Unique: Several valid heaps can be constructed for the same data set

because no ordering exists between siblings
Inserting into a PQ

- **Steps**
  - Add new value as a leaf
  - while (value < parent’s value)
    - swap with parent
  - Efficiency depends upon speed of
    - Finding a place to add new node
    - Finding parent
    - Tree height
Removing From a PQ

- Steps
  - Store the value of root
  - Delete the right most node among the nodes with the largest depth, put its value in the root
  - while (value > value of (at least) one child )
    - Swap with a child with the smallest value
  - Return the value stored in step 1
Implementing Heaps

ArrayTree Tradeoffs

- Why are ArrayTrees good?
  - Save space for links
  - No need for additional memory allocated/garbage collected
  - Works well for full or complete trees
    - No wasted space
    - Quick access to nodes (given the size of the tree)
- Why bad?
  - Could waste a lot of space for other trees
    - Tree of height of $n$ requires $2^{n+1}-1$ array slots even if only $O(n)$ elements

and Heaps can be kept complete
Implementing Heaps

- **VectorHeap**
  - Use array-based BT But use extensible Vector instead of array (makes adding elements easier)

- **Remember:**
  - Root of tree is location 0 of Vector
  - Children of node in location i are in locations 2i+1 (left) and 2i+2 (right)
  - Parent of node i is in location \([i-1]/2\)
Implementing Heaps

• Features
  - No gaps in array (tree is complete)
  - Heap Invariant becomes
    - data[i] <= data[2i+1]; data[i]<=data[2i+2] (or children might be null)
  - When elements are added and removed, do small amount of work to “heapify”
VectorHeap Summary

- Add/remove $O(\log n)$
- getFirst $O(1)$
Today’s Outline

- Heap
  - Basics
  - Heapify
  - Heapsort
Heapifying A Vector (or array)

- **Goal:** You are given a Vector V that is not a valid heap, and you want to make V a heap, i.e., “heapify” V
Heapifying A Vector (or array)

- Method I: Top-Down
  - Assume V[0...k] satisfies the heap property
  - Percolate up the item in location k+1
  - Then V[0..k+1] satisfies the heap property

- Time complexity
  - elements at depth d may be swapped d times

\[
O\left(\frac{n}{2}\log n\right) = O(n \log \log n)
\]
Heapifying A Vector (or array)

- Method II: Bottom-up
  - Assume V[k..n] satisfies the heap property
  - Push down the item in location k-1
  - Then V[k-1..n] satisfies heap property

- Time complexity
  - elements at depth d may be swapped h-d times

\[
\left( \frac{n}{2} \times 0 \right) + \left( \frac{n}{4} \times 1 \right) + \cdots + \left( 1 \times h \right)
\]

\[O(n)\]
Some Sums

All of these can be proven by induction.

\[ d=k \]
\[ d=0 \quad 2^d = 2^{k+1} \quad 1 \]

\[ d=k \]
\[ d=0 \quad r^d = (r^{k+1} \quad 1) / (r \quad 1) \]

\[ d=k \]
\[ d=1 \quad d \cdot 2^d = (k \quad 1) \cdot 2^{k+1} + 2 \]

\[ d=k \]
\[ d=1 \quad (k \quad d) \cdot 2^d = 2^{k+1} \quad k \quad 2 \]
Today’s Outline

• Heap
  • Basics
  • Heapify
• Heapsort
Heapsort

Steps:

- Make a *max-heap*: array[0…n]
  - array[0] is largest value
  - array[n] is “final” leaf
- Let k = n
- While k > 0:
  - “remove” the root of the max-heap stored in array[0…k] and store it at array[k]
  - Now array[0…k-1] stores a max-heap of size k-1, and array[k…n] is sorted
  - k = k - 1
Heapsort

- $O(n \log n)$ is guaranteed.
- Heapsort can be done \textit{in-place}
  - Great for resource-constrained environments
- But Heapsort is not \textit{stable}
Suppose we’d like to use multiple processors to build smaller heaps and then merge them together.

Rather than use Vector as underlying data structure, use BT.

Need a merge operation that merges two heaps together into one heap.

Details in book.