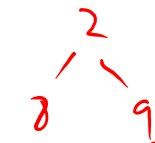
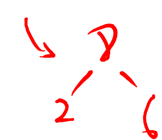
[TAP:RLOMA] PQ vs Heap



- 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.

    min-heap

    max-heap



  - C. PQ can be implemented with a Heap that is implemented with a Vector.
  - D. They are all correct
  - E. Whatever

Some of them are wrong

$\neg(\forall x \;\; x \text{ is correct})$

$\exists x \; \neg(x \text{ is correct})$

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

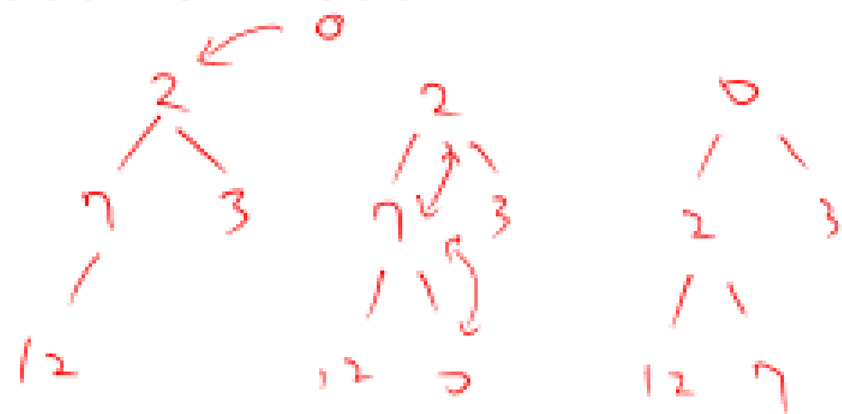
because no ordering exists between siblings

7

min-Heap

Inserting into a ~~PQ~~

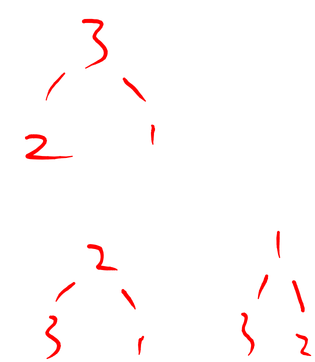
- Steps
  - Add new value as a "final" leaf
  - while (value < parent's value) "bubble up" "percolate up"
    - swap with parent
- Efficiency depends upon speed of
  - Finding a place to add new node
  - Finding parent
  - Tree height



Removing From a ~~PQ~~ min Heap

- Steps
  - Store the value of root
  - Delete the (right most node among the nodes with the largest depth), put its value in the root

    = "find" leaf
  - 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 and heaps can be kept complete
    - 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

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 $\lfloor (i-1)/2 \rfloor$

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" = percolate up

    push down

VectorHeap Summary

- Add/remove O(log n)
- getFirst O(1)

Today's Outline

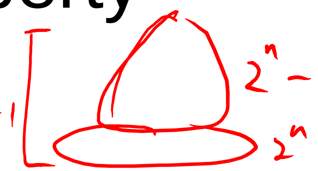
- Heap
  - Basics
  - Heapify
  - Heapsort

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

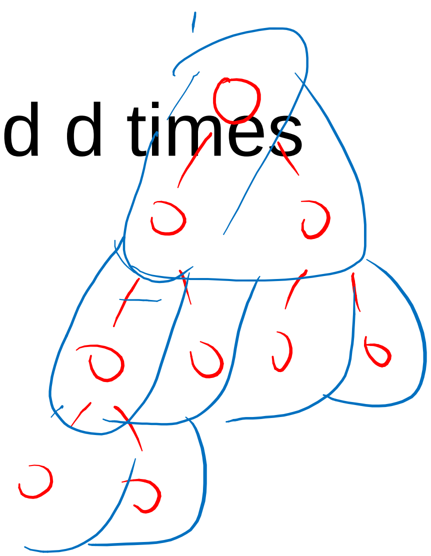
Heapifying A Vector (or array)

- Method I: Top-Down $k = 1 \cdots n$
  - 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 n = # of nodes

  h = height of the tree
  - elements at depth d may be swapped d times



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

$h = \log n$

$$O\left(\frac{n}{2} \log n\right) = \underline{O(n \log n)}$$

Heapifying A Vector (or array)

- Method II: Bottom-up V[4..8]
  - 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

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

$$\boxed{O(n)}$$

Some Sums

All of these can be proven by induction.

$$\sum_{d=0}^{d=k} 2^d = 2^{k+1} - 1$$

$$\sum_{d=0}^{d=k} r^d = (r^{k+1} - 1)/(r - 1)$$

$$\sum_{d=1}^{d=k} d * 2^d = (k - 1) * 2^{k+1} + 2$$

$$\sum_{d=1}^{d=k} (k - d) * 2^d = 2^{k+1} - k - 2$$

Today's Outline

- Heap
  - Basics
  - Heapify
  - Heapsort

Heapsort

given an unsorted array

- 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

$O(n)$

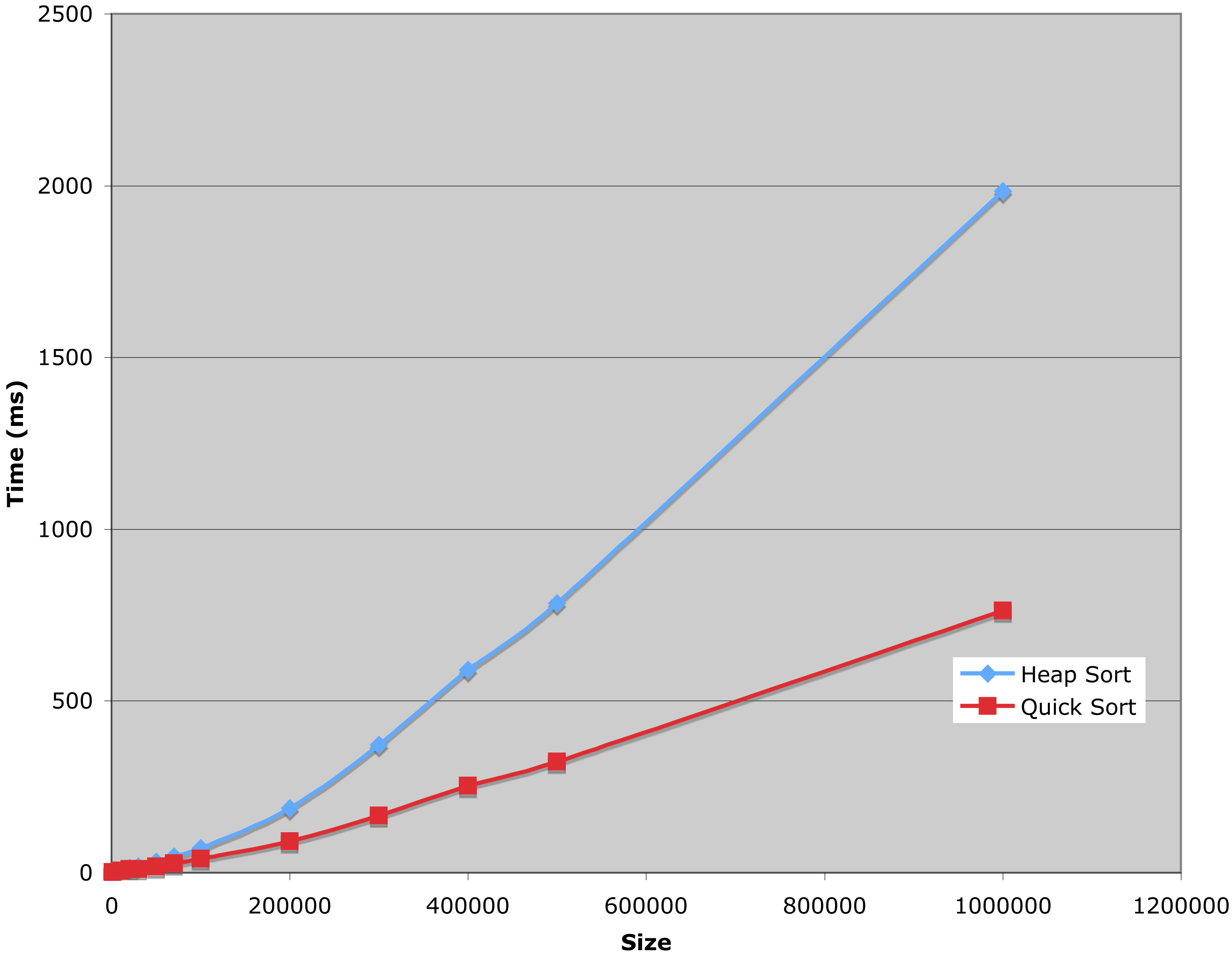
Selection sort

$n \times O(\log n)$

$= O(n \log n)$

$O(n \log n)$

Heapsort vs Quicksort



Heapsort

- $O(n \log n)$ is guaranteed.

- Heapsort can be done *in-place*
  - Great for resource-constrained environments
- But Heapsort is not *stable*

Skew Heap

- 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