**CS430 Lecture 19 Activities**

Opening Questions

1. Review the operations on a min (or max) binary heap and their runtimes

2. What if we needed a function to union two min binary heaps? How would you do it and what is the run time?

3. Why did we use an array for a binary heap? Does a heap need to be binary?

Mergeable (Min)Heaps

Consider the following operations on heaps. Our goal is to support all of these operations in no worse than Theta(log n) and not be constrained to use an array or a binary tree.

* Make-Heap: creates a new empty heap
* Insert: inserts a new element into a heap
* Minimum: returns the minimum element in a heap
* Extract-Min: returns the minimum element in a heap and removes it from the heap
* Union: creates a new heap consisting of the elements of two existing heaps

Two types of mergeable heaps are Binomial heaps and Fibonacci heaps. They also support these operations.

* Decrease-Key: changes the value of some element in a heap to a smaller value
* Delete: removes an element from a heap

Binomial Heaps (utilizing binomial trees)

|  |  |
| --- | --- |
| A binomial tree is an ordered tree defined recursively: | 1. How many nodes does Bk have, what is its height, and how many children does its root have?  2. Why are they called binomial trees? |

A binomial heap is a collection (linked list) of binomial trees in which each binomial tree is heap-ordered: each node is greater than or equal to its parent. Also, in a binomial heap at most one instance of Bi may occur for any i.

3. How many binomial trees are needed at most in the linked list of roots to make a binomial heap of n nodes?

4. See <https://www.cs.usfca.edu/~galles/visualization/BinomialQueue.html> to help describe how each operation is done, and its run time:

Make-Heap:

Minimum:

Union:

Insert:

Extract-Min:

Decrease-Key:

Delete:

Fibonacci Heaps

Fibonacci heaps which support heap operations that do not delete elements in constant amortized time. From a theoretical standpoint, Fibonacci heaps are especially desirable when the number of EXTRACT-MIN and DELETE operations is small relative to the number of other operations performed. This situation arises in many graph algorithms.

In essence, a Fibonacci heap is a “lazy” binomial heap in which the necessary housekeeping is delayed until the last possible moment: deletion.

|  |  |
| --- | --- |
| * Set of Heap ordered trees (each parent smaller than children) * Maintain pointer to minimum element (find-min takes O(1) time) * Set of marked nodes (if one of its children has been removed) * n – number of nodes in the heap * rank(x) – number of children of node x * rank(H) – max rank of any node in heap H * trees(H) – number of trees in heap H * marks(H) – number of marked nodes in H |  |

5. See <https://www.cs.usfca.edu/~galles/JavascriptVisual/FibonacciHeap.html> and <https://www.cs.usfca.edu/~galles/JavascriptVisual/FibonacciHeap.html> to help describe how each operation is done, and a rough estimate on its run time:

Make-Heap

Insert:

Minimum:

Union:

Extract-Min:

Decrease-Key:

Delete:

|  |  |
| --- | --- |
|  | Note that the times indicated for the Fibonacci heap are amortized times while the times for binary and binomial heaps are worst-case per-operation times. |