Exam 3 Review

- Data structures covered:
  - Hashing and Extensible hashing
  - Priority queues and binary heaps
  - Skip lists
  - B-Tree
  - Disjoint sets
- For each of these data structures
  - Basic idea of data structure and operations
  - Be able to work out small example problems
  - Prove related theorems
  - Advantages and limitations
  - Asymptotic time performance
  - Comparison
- Review questions are available on the web.

Hashing

- Hash table, table size.
- Hashing functions
  - Properties making a good hashing function
  - Examples of division and multiplication hashing functions
- Collision management
  - Separate chaining
  - Open addressing (different probing techniques, clustering)
- Worst case time performance: O(1) for find/insert/delete if λ is small and hashing function is good

Extensible Hashing

- Why need extensible hashing
  - Useful/advantageous only when hash table size is too large to store in memory (external storage accesses required)
- Basics for extensible hashing
  - Hash keys to long integers (binary): implicitly very large table size
  - Leaf: stores actual records (in disk), all records share the same leading d_L digits.
  - Directory:
    - Every entry has D digits
    - Each entry points to one leaf, with d_L <= D

Extensible Hashing

- Operations
  - Find, Remove (lazy remove)
  - Insert
    - Only insert to nonempty leaf,
    - Split if leaf full, extend directory
    - Duplicates (collisions)
- Compare with regular hash table (especially with separate chaining).

PQ and Heap

- Definition of binary heap (CBT with al partial order)
- Heap operations (implemented with array)
  - findMin, deleteMin, insert
  - percolateUp (for insertion), percolateDown (for deletion)
  - Heap construction, Heap sort
- Time performance of all operations
- Leftist tree and leftist heap
  - Why we need this?
  - Definition
  - Meld operations and applications

Skip Lists

- What is a skip list
  - Nodes with different size (different # of forward references or skip pointers)
  - Node size distribution according to the associated probability *p*
    - Nodes with different size do not have to follow a rigid pattern
    - What is the expected # of nodes with exactly *i* pointers?
    - How to determine the size of the head node ($\log_{1/p} N$)
- Why need skip lists
  - Expected time performance O(lg N) for find/insert/remove
  - Probabilistically determining node size facilitate insert/remove operations
  - Advantages over sorted arrays, sorted list, BST, balanced BST

- Skip list operations
  - find
  - insert (how to determine the size of the new node)
  - arrange pointers in insert and remove operations (backLook node in findInsertPoint)
- Performance
  - Expected time performance O(lg N) for find/insert/remove (very small prob. of poor performance when N is large)
  - Expected # of pointers per node: $1/(1 - p)$

B-Trees

- What is a B-tree
  - Special M-way search tree (what is a M-way tree)
  - Interior and exterior nodes
  - M and L (half full principle), especial requirement for root
- Why need B-tree
  - Useful/advantageous only when external storage accesses required
  - Why so?
  - Height $O(\log_M N)$, so are performances for find/insert/remove
- B-tree operations
  - search
  - insert (only insert to nonempty leaf, split, split propagation)
  - Remove (borrow, merge, merge propagation)
  - B-tree design (determining M and L based on the size of key, data element, and disk block)

Disjoint Sets

- Equivalence relation and equivalence class (definitions and examples)
- Disjoint sets and up-tree representation
  - representative of each set
  - direction of pointers
- Union-find operations
  - basic union and find operation
  - path compression (for find) and union by weight heuristics
  - time performance when the two heuristics are used:
    O(m lg* n) for m operations (what does lg* n mean)
    O(1) amortized time for each operation