The exam will be in class, this **Thursday, Nov 17**. It will be closed-book, closed-notes, but you will be allowed two “cheat sheets” of notes, front and back.

So far, we have studied a wide variety of data structures for a diverse set of applications. We have considered the material from both a theoretical and practical perspectives. We have illustrated various aspects of data-structure design and analysis, including worst-case and asymptotic analyses, randomized data structures.

**From the First Midterm:** The exam is cumulative in theory, but the emphasis will be on material from after the midterm. Nonetheless, some questions (perhaps 20%) may be drawn from before the first midterm. Here is a summary:

**Basic Data Structures:** Sequential and linked allocation, amortized analysis, multilists and sparse matrices.

**Amortized Analysis:** Rather than analyzing the running time of each individual operation, we instead consider the average running time for a sequence of operations.

**Trees:** Representations of rooted trees, binary trees and traversals, full and extended binary trees, threaded binary trees, complete binary trees (and array allocation).

**Disjoint Set Union/Find:** A tree-based data structure for maintaining a collection of disjoint sets, and supports the operations union and find. Very fast amortized running time.

**Priority Queues:** We studied the binary heap and the leftist heap.

**Ordered Dictionaries:** Support the operations of insert, delete, and find, and various ordered extensions of these operations (e.g., find-up, get-min, range queries).

**Binary Search Trees:** Standard (unbalanced) binary search trees. Good expected-case performance ($O(\log n)$) for random insertions.

**AVL Trees:** Height-balanced trees. All dictionary operations in $O(\log n)$ time (worst case).

**2-3 Trees:** Variable-width nodes. Also $O(\log n)$ worst-case time.

**Red-Black and AA Trees:** Binary encodings of 2-3 and 2-3-4 trees. Also $O(\log n)$ worst-case time.

**Treaps:** Randomized binary search tree. $O(\log n)$ expected-case time (over random choices).

**Skip lists:** Based on generalizing linked lists. Also $O(\log n)$ expected-case time (over random choices).

**Splay Trees:** A self-adjusting data structure. Good amortized performance for many operations.

**Quad- and kd-Trees:** Partition trees for geometric point data based on axis parallel cuts. We studied operations on kd-trees in detail.

**Insertion:** (Unbalanced) insertion leads to $O(\log n)$ height in expectation if insertion order is random.
**Deletion:** Similar to binary-tree deletion, but using a complex process for finding replacement nodes.

**Orthogonal range queries:** Counting queries can be answered in $O(\sqrt{n})$ time in $\mathbb{R}^2$ and generally $O(n^{1-1/d})$ in dimension $d$. Reporting queries can be answered as well, adding an additional term to account for the number of points reported (e.g., $O(\sqrt{n} + k)$).

**Nearest-Neighbor Queries:** Using a node’s cell to restrict which nodes need to be visited. (Not theoretically efficient, but practically efficient.)

**Range Trees:** Layered data structure based. Each layer constrains one additional condition on the keys. Can be used to answer many varieties of orthogonal range queries in $\mathbb{R}^d$. For counting, the query time is $O(\log^d n)$ and space is $O(n \log^{d-1} n)$. Can be adapted to a variety of queries through geometric transformations.

**Hashing:** Unordered dictionary, which works by scattering keys in a pseudo-random manner. Collision resolution is used to deal with instances of keys that hash to the same location. We studied linear probing, quadratic probing, and double hashing.