The Final exam is Tuesday May 1st from 8:00 - 9:50 AM in Wright 9. It will be closed-book and notes, except for **three 8” x 11” sheets of paper containing any notes that you want.** (Plus, the Python Summary Handout) About 75% of the test will cover the following topics (and maybe more) since the second mid-term test, and the remaining 25% will be comprehensive (mostly big-oh analysis and general questions about stacks, queues, priority queues/heaps, lists, and recursion).

**Chapter 6: Trees**

- **Coding question only over Ch.6**

  Terminology: node, edge, root, child, parent, siblings, leaf, interior node, branch, descendant, ancestor, path, path length, depth/level, height, subtree

  General and binary tree recursive definitions

  Tree shapes and their heights: full binary tree, balanced binary tree, complete binary tree

  Applications: parse tree, heaps, binary search trees, expression trees

  Traversals: inorder, preorder, postorder

  Binary search tree ADT: interface, implementation, big-oh of operations

  Balanced binary search trees: AVL tree ADT: interface, implementation, big-oh of operations

**File Structures - Lecture 24 handout:**

http://www.cs.unl.edu/~fienup/cs1520s18/lectures/lec24_questions.pdf

We talked about how the in memory data structures need to be adapted for slow disks.

From this discussion you should understand the general concepts of Magnetic disks:

- layout (surfaces, tracks/cylinders, sectors, R/W heads)
- access time components (seek time - moving the R/W heads over the correct track, rotational delay - disk spins to R/W head, data transfer time - reading/writing of sector as it spins under the R/W head)

Hash Table as a useful file structure

- **B+ trees as a useful file structure - see web resources:**

**Chapter 7: Graphs**

Terminology: vertex/vertices, edge, path, cycle, directed graph, undirected graph

Graph implementations: adjacency matrix and adjacency list

Graph traversals/searches: Depth-First Search (DFS) and Breadth-First Search (BFS)

General Idea of the following algorithms: topological sort, Dijkstra's algorithm (single-source, shortest path), Prim's algorithm (determines the minimum-spanning tree), TSP (Traveling-Saleperson Problem)

Approximation algorithm to solve TSP, general idea of backtracking and best-first search branch-and-bound.

You should understand the graph implementations and algorithms listed above. You should be able to trace the algorithms on a given graph.
BST

\[ \log_2 n \]

Add 50, 70, 90, 80, 30, 40, 25, 10, 5

Pread: 50, 30, 25, 10, 5, 40, 70, 90, 80

Postread: 5, 10, 25, 40, 30, 80, 90, 70, 50
AVL add 50, 70, 90, 80, 30, 40, 25, 10, 5

pivotal node →

\[ h' = \begin{cases} 
  x - 2 & \text{if } h + \text{LST} - h' + \text{RST} \neq 1 \text{ or } 2 \text{ or } 3 \\
  x - 1 & \text{otherwise} 
\end{cases} \]
9. A B+ Tree is a multi-way tree (typically in the order of 100s children per node) used primarily as a file-index structure to allow fast search (as well as insertions and deletions) for a target key on disk. Two types of pages (B+ tree "nodes") exist:

- Data pages - which always appear as leaves on the same level of a B+ tree (usually a doubly-linked list too)
- Index pages - the root and other interior nodes above the data page leaves. Index nodes contain some minimum and maximum number of keys and pointers based on the B+ tree's branching factor \( b \) and fill factor. A 50% fill factor would be the minimum for any B+ tree. All index pages must have \( \lceil b/2 \rceil \leq \# \text{ child} \leq b \), except the root which must have at least two children.

Consider an B+ tree example with \( b = 5 \).

![B+ tree example diagram]

a) How would you find 88?

b) The insert algorithm for a B+ tree is summarized by the below table. Where would you insert 50, 100, 105, 110, 180, 200, 210?

<table>
<thead>
<tr>
<th>Situation</th>
<th>insertion Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Page</td>
<td>Parent Index Page</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Stack implementation

\[
\begin{array}{cccc}
& & & \\
0 & 1 & 2 & 3 \\
\hline
\end{array}
\]

top

push, pop, peek

\[O(n)\quad O(n)\quad O(1)\]