Final Review
Final Exam...

• Part A: Basic knowledge of data structures and C++
  – 20% of final score
  – Multiple choice

• Part B: Application, Comparison and Implementation of the data structures
  – 20% of final score
  – Apply supported operations (like find and insert) to data structures we have covered like: BST, AVL, RBTs, Multiway trie, Ternary Trie, B trees, skip lists. Also essential concepts in Huffman codes

• Part 3: Simulating algorithms and run time analysis
  – 15% of final score
  – Graph algorithms: BFS, DFS, Dijkstra, Prims’, Kruskals’. Also union find

• Part 4: C++ and programming assignments
  – 15% of final score
  – Short answer
b) Which of the following are legal 2,3 trees (B tree of order 3)? For a tree that is not a valid 2,3 tree, state a reason why.

A.  
```
22  
30  
8   64
```

B.  
```
22 52  
5 9  
24 53 54
```

C.  
```
22  
5 9 1 7
```

D.  
```
12 50 79
```

\( m=4 \)  
```
4 7 
30 64 69 84 102
```

B-tree at order 4
Insert the value 42 into the following BTree.
Ternary Tries

(a) Consider the following ternary search tree. Nodes with double circles have their end bits set to true. Circle all of the words from the list on the right that are in the tree and write in any words that are missing. At the end you should have a complete list of all words found in the tree, and only those words.

(b) Does the height of this tree depend on the order in which the keys have been inserted?

(c) Briefly explain why you would prefer to use a ternary search tree rather than a binary search tree to implement `getValidWords()` in PA4.
Write the sequence of vertices visited when running DFS on the following graph. Assume the link to the vertex with the minimum edge weight is chosen when multiple choices are available

A, C, D, B, E

Write the sequence of vertices visited when running BFS on the following graph. Assume the link to the vertex with the minimum edge weight is chosen when multiple choices are available

A C B E D
Which of the following is/are balanced trees?

And thus can become AVL trees by adding the balance factors

A. A&B&C

D. A&C

E. A&B&C

Annotate the trees with balance factors
Insert 50. Then insert 66. Draw the resulting AVL tree.
Red Black trees

How can we make the above tree a valid red-black tree

Insert 50. Then insert 66. Draw the resulting red-black tree.
# Data structure Comparison

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Insert</th>
<th></th>
<th>Find</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg</td>
<td>Worst</td>
<td>Avg</td>
<td>Worst</td>
</tr>
<tr>
<td>Sorted array</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
</tr>
<tr>
<td>Sorted Linked list</td>
<td>$O(1)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>Queue</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>Skip list</td>
<td>$O(1)$</td>
<td>$O(N)$</td>
<td>$O(\log N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>BST</td>
<td>$O(\log N)$</td>
<td>$O(N)$</td>
<td>$O(\log N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>AVL/RBT</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
</tr>
<tr>
<td>Min-heap</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>Hash table</td>
<td>$O(1)$</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>B-trees</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(\log N)$</td>
</tr>
</tbody>
</table>

⚠️ $M$ and $N$ denote specific complexities.
Data structure Comparison

Which of the following pairs of data structures, which of the pair is the better choice for:

• Inserting a list of sorted elements (worst case):
  A. AVL tree
  B. Binary search tree
  C. They are about equal

• Ease of implementation (assume it’s not built in):
  A. Skip list
  B. Red-Black tree (RBT)
  C. about equal

• In-order traversal of elements:
  A. Hashtable
  B. Binary search tree
  C. They are about equal
Data structure Comparison

Which of the following pairs of data structures, which of the pair is the better choice for.

• Smallest average-case (Big-O) time to find an element:
  A. Hashtable
  B. AVL tree
  C. They are about equal

• Fastest actual time to find an element from secondary storage (NOT big-O)
  A. RBT
  B. AVL tree
  C. B-trees
  D. They are all about equal

• Requires less space:
  A. Multi-way trie
  B. Ternary tree
  C. They are about equal
Some comic relief...

```
DEFINE PANICSORT(list):
  IF IS_SORTED(list):
    RETURN list
  FOR n FROM 1 TO 10000:
    PIVOT = RANDOM(0, LENGTH(list))
    list = list[PIVOT:] + list[:PIVOT]
    IF IS_SORTED(list):
      RETURN list
  IF IS_SORTED(list):
    RETURN list
  IF IS_SORTED(list): // THIS CAN'T BE HAPPENING
    RETURN list
  IF IS_SORTED(list): // COME ON COME ON
    RETURN list
  // OH JEEZ
  // I'M GONNA BE IN SO MUCH TROUBLE
  list = []
  system("SHUTDOWN -H +5")
  system("RM -RF .")
  system("RM -RF ~/*")
  system("RM -RF /")
  system("RD /S /Q C:\*") // PORTABILITY
  RETURN [1, 2, 3, 4, 5]
```

http://xkcd.com/1185/
Some comic relief...

```
DEFINE JOBINTERVIEWQUICKSORT(list):
  OK SO YOU CHOOSE A PIVOT
  THEN DIVIDE THE LIST IN HALF
  FOR EACH HALF:
    CHECK TO SEE IF IT'S SORTED
    NO, WAIT, IT DOESN'T MATTER
    COMPARE EACH ELEMENT TO THE PIVOT
    THE BIGGER ONES GO IN A NEW LIST
    THE EQUAL ONES GO INTO, UH
    THE SECOND LIST FROM BEFORE
    HANG ON, LET ME NAME THE LISTS
    THIS IS LIST A
    THE NEW ONE IS LIST B
    PUT THE BIG ONES INTO LIST B
    NOW TAKE THE SECOND LIST
    CALL IT LIST, UH, A2
    WHICH ONE WAS THE PIVOT IN?
    SCRATCH ALL THAT
    IT JUST RECURSIVELY CALLS ITSELF
    UNTIL BOTH LISTS ARE EMPTY
    RIGHT?
    NOT EMPTY, BUT YOU KNOW WHAT I MEAN
    AM I ALLOWED TO USE THE STANDARD LIBRARIES?
```
Good luck with the final!