Name: ______________________

CSE 326 DATA STRUCTURES
Midterm Exam
Summer 2007

Do not begin work until everyone has received their exam. Please put your name on all pages. The test is 11 pages long, and worth a total of 100 points.

If you have scratch work, please be sure to **box off your final answer from the scratch work**, or I may grade the wrong thing.

Question	Points	Score
1	15	
2	15	
3	20	
4	25	
5	25	
Total:	100	

1. (15 points) Identifications
   (a) (6 points) Identify the types of the following constraints, and identify the ADT to which they belong:
      i. (2 points) Every path from root to leaf must be monotonically increasing.
      ii. (2 points) The left and right children of every node of this tree must have heights differing by at most 1.
      iii. (2 points) A forest of trees, where each tree B_k consists of a root node to which is attached one tree each of size $B_0, B_1, \ldots, B_{k-1}$.
   (b) (5 points) Identify the most appropriate data structure, from among the ones we've discussed in class so far, for the following scenarios, and give a *brief* (5–10 words) reason why:
      i. Fast random access for a multi-GB database.
      ii. Guaranteed fast access, insertion and deletion to a small in-memory tree.
      iii. Efficient scheduling of jobs by length.
   (c) (4 points) Define and distinguish an Abstract Data Type (ADT) from a concrete data structure.

2. (15 points) Asymptotics

(a) (5 points) Place the following functions of n in increasing order of growth. Circle together any functions that have the same order of growth.

$$\log(n^2),\quad \frac{1}{n^2},\quad \log\log n,\quad \binom{n}{3},\quad 3^n,\quad 2^n,\quad \frac{n!}{2n},\quad \log_5 n,\quad n^2,\quad n^n,\quad 15,\quad n^2\log n$$

(b) (5 points) Analyze the two versions of *FindInSortedArray* below. Write down the recurrences and solve them to get big-O runtimes. Which version is faster?

```
bool FindInSortedArray1(int source[],
                        int minIndex, int maxIndex,
                        int target) {
  if minIndex > maxIndex
    return false
  if source[minIndex] == target
    return true
  return FindInSortedArray1(source, minIndex + 1, maxIndex, target)
}

bool FindInSortedArray2(int source[],
                        int minIndex, int maxIndex,
                        int target) {
  if source[(minIndex + maxIndex)/2] == target
    return true
  if minIndex >= maxIndex
    return false
  if source[(minIndex + maxIndex)/2] < target
    return FindInSortedArray2(source,
                              (minIndex + maxIndex) / 2, maxIndex, target)
  else
    return FindInSortedArray2(source,
                              minIndex, (minIndex + maxIndex) / 2, target)
}
```

For convenience, you may define n as the original length of the array for use in your analysis.

(c) (5 points) Write down the complete recurrence relation $T(n)$ for the running time of the following function. Be sure to include a base case. *You do not have to solve this relation, just write it down. You may assume n is always positive.*

```
int mystery(int n) {
  int answer;
  if (n > 0) {
    answer = (mystery(n-3) + 3*mystery(n/4)) / 5;
    return answer;
  } else {
    return 1;
  }
}
```

(a)

(b)

(c)

3. (20 points) Efficiency

   Give a big-O bound on the *worst-case* running time for each of the following in terms of n. No explanation is required, but explanations may help for partial credit. Assume that all keys are distinct.

   (a) (3 points) Insertion in an AVL tree of size n.

   (b) (3 points) Insertion into a binary heap of size n.

   (c) (3 points) Insertion into a splay tree of size n.

   (d) (3 points) Floyd's `buildHeap` for a heap of size n.

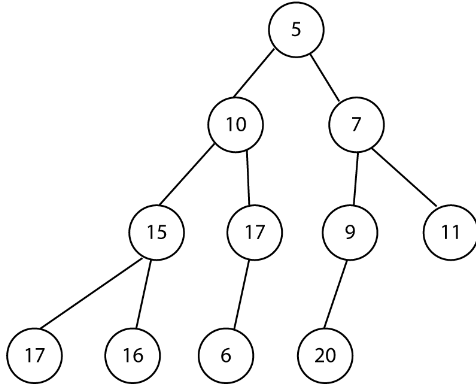
   (e) (3 points) FindMin in a max-heap of size n.

   (f) (2 points) FindMin in a min-heap of size n.

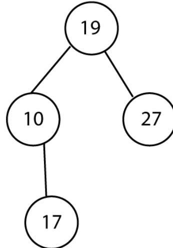
   (g) (3 points) Deletion from a binomial queue of size n.

4. (25 points) Technical operations

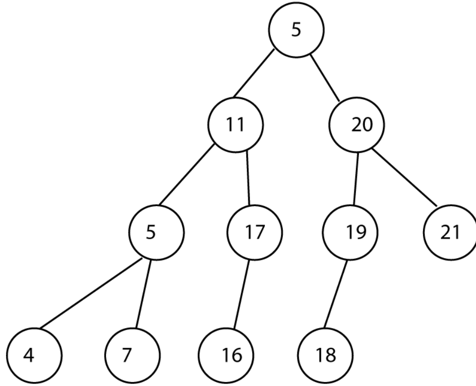
   (a) (5 points) Identify two problems with the following binary heap. One problem cannot be fixed with heap operations; the other can. Explain what to do, and draw a new tree to show how to fix that problem.



   (b) (5 points) Insert 16, 15, and 14 in that order into the following AVL tree.



(c) (5 points) Delete 11 and 19 from the following Binary Search Tree.



(d) (5 points) Draw the up-trees from the given up-array. Find 6, using path compression. Write the resulting up-array.

i	1	2	3	4	5	6	7	8	9	10	11	12
$p[i]$	2	5	1	2	0	1	3	7	6	11	0	8

(e) (5 points) Give the *pre-order* traversal of the result of part 3b.

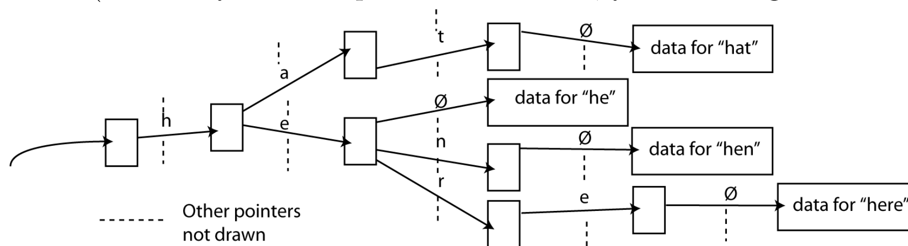
5. (25 points) Design

In literary and linguistic studies, a *concordance* is a special kind of index, which lists all the words present in a particular text, along with their counts, their locations in the text, definitions, and other information. Professor Libby R. Arian has asked you to write a program to input and store such a concordance. She needs to be able to efficiently:

1. `DEFINE` a *term* with a particular *definition*. She only needs one definition per term, though.
2. `INDEX` a *term* at a particular *page*. Obviously, a term can be used many times in a particular text, but if it occurs multiple times on a single page, the page number should only be reported once (see next operation), but counted each time.
3. `FIND` all instances of a *term* within a *start* and *end* page. This could print out multiple results, so she wants them printed in order, please.
4. `COUNT` all instances of a *term*. This should include every instance of the term, even if it occurs multiple times on the same page.

Since you're a smart Computer Scientist, you realize you only need to handle two primitive operations: `INSERT`, and `LOOKUP`. From there, you can easily define all the operations above. However, you realize that for something this large, you want to save as much storage space as possible, so you come up with the following plan:

- Your primary storage will be a 27-way tree. The first 26 branches correspond to each letter of the alphabet; the 0th branch indicates a word has stopped. The idea is to encode each word as a *path* down from the root to a leaf, rather than store the word explicitly within each leaf. For instance, the following tree holds the words "hen", "here", "he" and "hat" (note: only non-null pointers are drawn; you can imagine the rest are there):



- `INSERT`ing a word into this structure means reading each character, one at a time, and following a path down the appropriate branches. If a path ends too soon, create new nodes as needed. When the word finishes, create a leaf node at the 0th branch of the last letter, and store the word's information there. Only the leaves of this tree correspond to real words.
- `LOOKUP` can return `null` if the word is not present.

Therefore, your job is to define the following two functions:

```
Node INSERT(Node root, char[] term)
Node LOOKUP(Node root, char[] term)
```

and to implement

```
void INDEX(Node root, char[] term, int page)
void FIND(Node root, char[] term, int startPage, int endPage)
```

using them. Remember, Professor Arian wants these to be as efficient as possible. As a hint, here are `DEFINE` and `COUNT`:

```
void DEFINE(Node root, char[] term, char[] definition) {
  Node termNode := LOOKUP(root, term);
  if (termNode = NULL) {
    termNode := INSERT(root, term);

  termNode.definition = definition;
}

int COUNT(Node root, char[] term) {
  Node termNode := LOOKUP(root, term);
  if (termNode = NULL)
    return 0;
  return termNode.count;
}
```

You may use any of the data structures and operations we defined in class, and do not have to write them from scratch. (For instance, you can assume `InsertIntoSplayTree` or `StackPop` exist, if you want to use them.) If you're not sure how to define these operations precisely using pseudocode, describe them as precisely as you can in English. Explain what data structure you need to store the page numbers, what you intend each function to do, etc.

No function should take more than 20 lines or so.

(a) (7 points) `INSERT`:

(b) (5 points) `LOOKUP`:

(c) (7 points) `INDEX`:

(d) (6 points) `FIND`: