Exam 2 Important Information

- Exam 2 will be held on Monday, April 7, 2003, from 2:00 to 3:30 in DCC308.
- Coverage is Chapters 4 through 7.
- The exam is closed-book and closed-notes.

Exam 2 Guidelines

- To prepare for the exam you should review the class notes, labs and projects, and you should rework the homework problems. Included below are a few extra problems to help you prepare for the exam.

- The exam will require that you simulate some algorithms by hand, including AVL trees, splay trees, binary heaps, and leftist heaps. We will also provide you with the pointer manipulation code to do single rotations from the left, single rotations from the right, and double rotations. You need to remember the criteria under which you should apply them.

- The exam will also require that you write several different functions to manipulate the data structures we have discussed.

- You must have an idea of how each of the sorting algorithms we discuss works. Detailed questions about algorithm design or performance will include code for you to work from.

- Finally, you need to understand the computational advantages and disadvantages of the data structures we have discussed. Think about which ones are faster or slower for which operations?
Practice Questions

1. Consider the following binary search tree.

```
          24
           /
          /  
         8    35
        /    / 
       4    18  28
    /    /  
   12    21
```

Assuming it is an AVL tree, show the state of the tree after inserting 10 and then again after deleting 28.
Do the same assuming it is a splay tree.

2. Given an empty AVL tree of integers, show the structure of the tree after each of the values 5, 2, 4, 6, 7, 1, 8 is inserted and then show the change to the resulting tree when 4 is deleted. Indicate where rotations are done to rebalance the tree.

3. Given a binary search tree containing $n$ floating point values and having a height that is $O(\log n)$ (i.e. it is balanced), write an algorithm to count the number of nodes storing values greater than $x_0$ and less than $x_1$. What is the running time of your algorithm? (More credit will be given for an efficient algorithm.)

You may assume the following declaration for tree nodes:

```cpp
class TreeNode {
public:
    float x;
    TreeNode * left;
    TreeNode * right;
};
```

Start from the following prototype and assume the function is initially passed a pointer to the root:

```cpp
int Count( TreeNode * T, float x0, float x1 )
```

4. Consider a binary heap, implemented, of course, as an array. Write a function to delete the element stored at location $i$ of the heap. Assume that functions Percolate_Down and Percolate_Up exist, with prototypes

```cpp
void Percolate_Down( ElementType heap[], int size, int i);
void Percolate_Up( ElementType heap[], int size, int i);
```
where \texttt{heap} is the array of elements, and \texttt{size} is the current number of elements. Assume that \texttt{i} is correctly given to you, i.e., assume \(1 \leq i \leq \texttt{size}\).

Start from the following prototype

\[
\text{void Delete( ElementType heap[], int \& size, int i);}
\]

5. Consider a priority queue implemented as a heap. The heap is implemented using a vector, and it contains \(n\) values stored in subscript locations \(1\) through \(n\). Assume it is a “max heap”, so that the largest value is stored in subscript location \(1\).

(a) What are the possible subscript locations for the second largest value in the heap (assuming all values are distinct)?

(b) What is the range of possible subscript locations for the smallest value in the heap (again assuming all values are distinct)?

(c) What is the worst-case time complexity required to find the minimum value in the heap?

6. Suppose you are given a pointer, \texttt{Root}, to the root of a leftist heap of floating point values. Suppose you are also given a pointer, \texttt{P}, to a particular node in the leftist heap. Assume each \texttt{Leftist} node has the structure

\[
\begin{align*}
\text{class Leftist} &\{ \\
&\text{public:} \\
&\quad \text{float Value;} \\
&\quad \text{Leftist *Parent, *Left, *Right;} \\
&\quad \text{int Np1; // the null path length} \\
&\};
\end{align*}
\]

You may assume the existence of a merge function with the following prototype:

\[
\text{Leftist* Merge( Leftist* t1_root, Leftist* t2_root )}
\]

which merges two leftist heaps, updates the null path lengths as necessary, and returns a pointer to the root of the resulting leftist tree.

Write a function to remove the node pointed to by \texttt{P}. The function should be as efficient as possible. Here is the prototype:

\[
\text{void LeftistRemove( Leftist* \& Root, Leftist* \& P )}
\]
7. Here is a slightly different version of the Quick Select algorithm, which is much closer to the original version of Quick Sort (as opposed to the Median-of-Three version). At the end of the function (including all recursive calls), the desired value will be in the $k$th location of the array.

```cpp
Q_Select_One ( int A[ ], const int k, const int Left, const int Right )
{
    if( Left < Right ) {
        int Pivot = A[ Left ];
        unsigned int i = Left, j = Right + 1;
        for( ; ; ) {
            while( A[ ++i ] < Pivot && i<Right );
            while( A[ --j ] > Pivot );
            if( i < j )
                Swap( A[ i ], A[ j ] );
            else
                break;
        }
        if( k < j ) Q_Select_One( A, k, Left, j-1 );
        else if( k > j ) Q_Select_One( A, k, j+1, Right );
    }
}
```

(a) Assume the original call is made to this function with

```
Q_Select_One( A, 3, 0, 8)
```

with the following contents of $A$

```
0 1 2 3 4 5 6 7 8
14 24 8 16 32 71 26 25 12
```

Show the contents of $A$ near the end of the code, just before entering the conditional to check if a recursive call should be made. What recursive call, if any, is made?

(b) What is the worst-case number of comparisons in Q_Select_One as a function of both $n$ and $k$, where $n$ is the value of Right-Left+1 in the first call? When does this occur?

8. Solve the Quick Sort recursive function

$$T(n) = T(k) + T(n - k - 1) + n$$

to yield a non-recursive form when $k = 1$. Give an “O” estimate of this form. Assume that $T(0) = T(1) = 0$ and $T(2) = 2$. You may assume that $n$ is even.