Computer Science I
Final Examination - Nov 28th, 2011

Max Marks: 50 **Time: 3 Hrs**

Answer ANY 4 out of questions 1-6 (*Algorithms*) and ANY 2 from questions 7-9 (*Programming*)

---

1. Consider a circularly sorted array A of size n. An array is circularly sorted if there is a k, $1 \le k \le n$ such that $A[k+1] \dots A[n]\ A[1] \dots A[k]$ (the array indices start from 1 and if $k = n$, the sequence will be $A[1] \dots A[k]$) is the sorted order of the elements in A. Notice that this is a sorted list with a circular shift. Show (with the pseudocode for your algorithm) how you can search for an element in the array in time $O(\log n)$. **Marks: 9**

2.
   a. Hadamard Matrices $H_1, H_2, \dots, H_n$ are defined as follows:
      - H_0 is the $1X1$ matrix $[1]$
      - For $k > 0$, H_k is the $2^k X 2^k$ matrix

$$\begin{bmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} \end{bmatrix}$$

      Show that if v is a column vector of length $n = 2^k$, then the matrix-vector product $H_k v$ can be calculated using $O(n \log n)$ operations. Assume that all the numbers involved are small enough that basic arithmetic operations like addition and multiplication take unit time. **Marks: 5**

   b. Assume that nXn matrices can be squared in $S(n) = O(n^c)$ time, for a constant c. Find out how long it would take to compute $AB+BA$ using squaring. Now take any two nXn matrices X and Y. Show that they too can be multiplied in $O(n^c)$ time by looking at $AB+BA$ for two $2nX2n$ matrices $A = \begin{bmatrix} X & 0 \\ 0 & 0 \end{bmatrix}$, $B = \begin{bmatrix} 0 & Y \\ 0 & 0 \end{bmatrix}$. Incidentally this shows squaring a matrix is no easier than multiplying any two arbitrary matrices. **Marks: 4**

3. Prove the following two properties of the Huffman encoding scheme. **Marks: 4+3+2**
   a. If some symbol appears with frequency more than $2/5$ then there will be a symbol with code of length exactly 1.
   b. If the frequency of every symbol is strictly less than $1/3$ then no symbol as code of length 1.
   c. Suppose the symbols a, b, c, d, e occur with frequencies $1/2$, $1/4$, $1/8$, $1/16$ and $1/16$ respectively. What is the Huffman encoding of these symbols? If this encoding is applied to a file consisting of $1{,}000{,}000$ characters with the given frequencies, what is the length of the encoded file in bits?

4. ***Nuts and bolts***. You have a mixed pile of n nuts and n bolts and need to quickly find the corresponding pairs of nuts and bolts. Each nut matches exactly one bolt, and each bolt matches exactly one nut. By fitting a nut and bolt together, you can see which is bigger, but it is not possible to directly compare two nuts or two bolts. Give an efficient method for solving the problem. What is the complexity of your algorithm? Can you show that it is optimal for the nuts and bolts problem? ***Hint***: Look at its relationship with a familiar sorting algorithm. **Marks: 9**

5. A d-ary heap is a heap where each non-leaf element has d children. **Marks: 3+4+2**
    a. How would you implement a d-ary heap in an array?
    b. Analyze the complexity of the following operations on a d-ary heap – Extract Min/Max, Heap_Increase_Key(A,i,k) that sets the element A[i] of the heap (represented as an array A) to max(A[i], k) and readjusts the heap to restore the heap property. The complexity will be a function of n and d.
    c. Is the array [23,17,14,6,13,10,1,5,7,12] a binary heap? Justify your answer.

6. Suppose you are choosing between the following three algorithms: **Marks: 9**
    a. Algorithm A solves problems by dividing them into five sub-problems of half the size, recursively solving each sub-problem, and then combining the solutions in linear time.
    b. Algorithm B solves problems of size n by recursively solving two sub-problems of size $n-1$ and then combining the solutions in constant time.
    c. Algorithm C solves problems of size n by dividing them into nine sub-problems of size $n/3$, recursively solving each sub-problem, and then combining the solutions in $O(n^2)$ time.

    What are the running times of each of these algorithms (in big-O notation), and which would you choose?

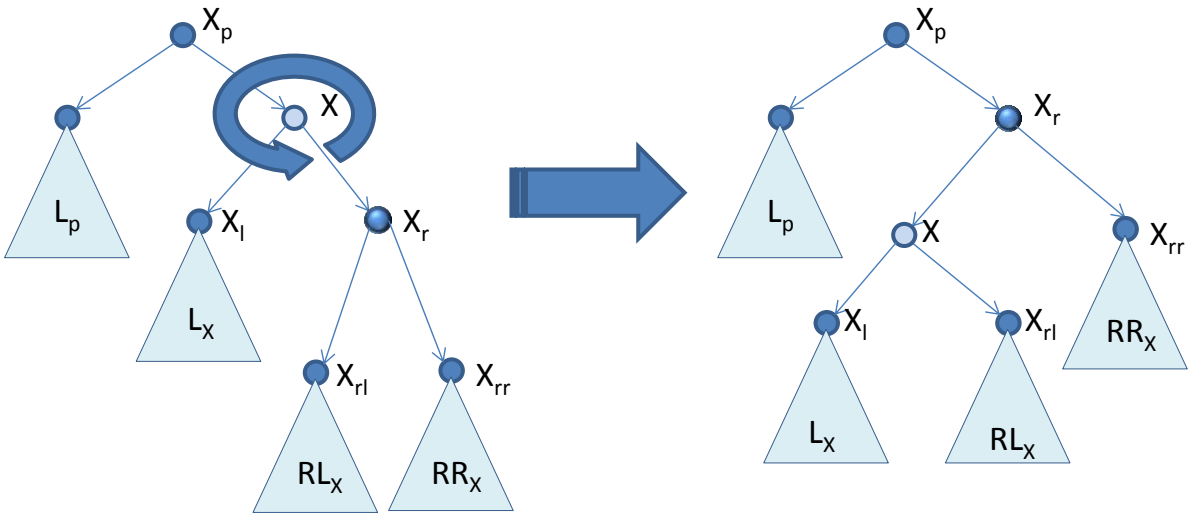
7. Assume an explicit implementation of a binary tree using pointers where each node in the binary tree is an instance of a TreeNode class as defined below:

```
template <typename T>
class TreeNode {
public:
        // pointers to parent node, left child and right child
        TreeNode *up, *left, *right;
        T data;
}
```

    A key property of a binary tree that needs to be maintained to ensure that the operations (such as search) on the tree are efficient is that every root->leaf path in the tree is at most $O(\log n)$ where n is the total number of nodes in tree. In other words, since the worst case complexity of such operations on the tree will often be of the same order as the length of the longest root->leaf path in the tree, we don't want the tree to be lopsided where a few

paths are very long (say something like **O(n)**). An important primitive that is often applied on the tree to keep it balanced is a *rotation* at a node of the tree as illustrated in the picture below (the picture illustrates a rotation at the node **X**). Write the code for such a rotation given a pointer to the node about which the rotation needs to be done. It could be a method for instance in a **BinaryTree** class with following signature returning the pointer to the (new) node at the rotated position:

TreeNode* BinaryTree<T>::rotateAt(TreeNode *node);



Marks: **7**

8. Sparse matrices are used in a number of numerical methods (e.g., finite element analysis). A sparse matrix is one which has the great majority of its elements set to zero. In practice, sparse matrices of sizes up to several hundreds (even thousands) of rows and columns are not uncommon. On a machine which uses a 64-bit representation for reals, storing a 500 X 500 matrix as an array would require 2 megabytes of storage. A more economic representation would record only nonzero elements together with their positions in the matrix. We have defined a **SparseMatrix** class as shown below which uses linked-lists (one for each row) to keep only the nonzero elements. Assume all the matrix entries are integers.

```
class SparseMatrix {
public:
        // creates 'size' linked lists – all of which are empty (only 0's)
        SparseMatrix(int n)
                { size = n; rows = new LinkedList<SparseElem>[n]; }

        LinkedList<SparseElem> *rows; // Array of rows as linked lists

        int size; // remembers the number of rows
```

```
    // adds two sparse matrices – assume they have the same sizes
    SparseMatrix operator+(SparseMatrix& m1, SparseMatrix& m2);
    int operator[](int i, int j) const; // gets the [i,j]-th element
}
```

Define the class **SparseElement** that keeps the information necessary to define an element of the sparse matrix. Write the code for the two operators **+** and **[]** of **SparseMatrix** to return the sum of two sparse matrices and to return the **[i,j]**-th element of the matrix, respectively. Assume that the **LinkedList<SparseElem>** class supports the following methods:

```
SparseElem currentItem(void); // returns the current data item

// Inserts elem next to the current item – inserted elem becomes current
// Also if the list is currently empty it will make the new elem the head
void insert(SparseElem elem);
void next(void); // current moves to the next element of the list
bool endOfList(void); // returns true if the current element is the last in the list
void reset(void); // resets the current element to be the head of the list
```

Marks: **7**

9. Create a class to represent the time of day. Call your class **Time** and give it the following attributes and methods.
   a. The data should be held in three private variables representing the hour, minute, and second with the corresponding **getHour(void)**, **getMinute(void)** and **getSeconds(void)** methods.
   b. The default (no-argument) constructor should create a value equal to midnight and a three-argument constructor should create the time specified (hours from **0** to **23**).
   c. Define addition of a Time object and a (long) integer. If **T** is type **Time** and **n** is type long, then **T+n** and **n+T** should be the time **n** seconds after **T**. (Of course, **n** might be negative.)
   d. Define subtraction, but only in the form **T-n** but **not n-T**.
   e. Define **++** and **--**; these should increase (decrease) **T** by one second, respectively.
   f. Define **ampm(void)** and **military(void)** methods to control how the time is printed. These methods should affect how all **Time** objects are printed. Also provide a **is_ampm(void)** method that returns true if the current output style is to use AM/PM and false if the current style is military (24 hour).
   g. Define **<<** for printing Time objects to the screen. The style of the output should either be **5:03:24 pm** or **17:03:24** as specified by the user with the methods **ampm()** and **military()**, respectively. Note the zero in front of the **3** but not in front of the **5**. Midnight should be reported either as **12:00:00 am** or **0:00:00** and noon as **12:00:00** noon.

Marks: 7