Dept. of Computer Science and Engineering, undergraduate
National Sun Yat-sen University
Data Structures - Final Exam., Jan. 9, 2012

1. Multiple choices (There may be zero or more correct answers. If there is no correct answer, you should write down "None".) (20%)
   Answer: (a) B (b) AD (c) BC (d) ABC (e) AD

   (a) Which statement(s) is correct for a *binary tree*? (A) A binary tree is a special case of an *ordered tree*. (B) An *ordered tree* can be implemented by a binary tree. (C) An *almost complete binary tree* is a *strictly binary tree*. (D) A *strictly binary tree* is an *almost complete binary tree*.

   (b) Which statement(s) is correct for an *AVL tree*? (A) An insertion in an AVL tree requires at most two rotations. (B) A deletion in an AVL tree requires at most two rotations. (C) A *left rotation* performed on a *complete binary tree* will make the height difference of the two subtree of the root be 1. (D) If no rotation is required in an insertion, then the tree height may increase at most 1.

   (c) Which sorting algorithm(s) is a *stable* sorting method? (A) Heap sort (B) Insertion sort (C) Merge sort (D) Shell sort.

   (d) Which sorting algorithm(s) needs about $\frac{n(n-1)}{2}$ comparisons in the worst case, where n is the size of the input data? (A) Bubble sort (B) Insertion sort (C) Quicksort (D) Radix sort.

   (e) Which statement(s) is correct for the *hashing* method? (A) Hashing is a fast searching method. (B) If the elements in the hashing table are extracted linearly, then they are ordered. (C) An element deletion can be accomplished by first finding its position with the hashing function and then removing the element directly. (D) Linear probing is one of the methods for resolving hash collisions.

2. (a) Please explain *Huffman algorithm* with the message "ABACCDA" as an example. Draw the *Huffman tree* for this example. (12%)

   (b) What is a *strictly binary tree*? Is a Huffman tree strictly binary? Why? (4%)

3. The following two methods can increase the performance of the sequential search. Please explain these two methods and give examples to illustrate the methods.

   (a) move-to-front method. (5%)

   (b) transposition method. (5%)

4. Given a permutation $p_1 p_2 \cdots p_n$ of $123 \cdots n$, a *homing operation* (going home at once), denoted as h_j, is to put one number p_i into its home position j and to shift the numbers between positions i and j, where the home position of p_i is j if the value of p_i is j. Note that here, the leftmost position is position 1 and the rightmost position is position n. For example, after h_2 is applied, 81374256 will become 82137456, and after h_7 is applied, 81374256 will become 81342576.

   (a) How do you sort 81374256 into 12345678 with the minimum number of homing operations? (5%)

   (b) To sort 23451 into 12345 requires only one homing operation (h_1). But, you are asked to use the maximum number of homing operations to sort 23451 into 12345. How do you finish this job? (10%)

Answer: (a) 3 operations: h_8, h_7, h_2 or h_8, h_2, h_7
(b) 15 operations: $h_3, h_2, h_4, h_2, h_3, h_2, h_5, h_2, h_3, h_2, h_4, h_2, h_3, h_2, h_1$

5. Givens two strings $A =$ "$a_1 a_2 \cdots a_m$" and $B =$ "$b_1 b_2 \cdots b_n$", the following longest common subsequence (LCS) algorithm can be used to evaluate the similarity between A and B:
   $L_{i,j} = L_{i-1,j-1} + 1$ if $a_i = b_j$, for $1 \le i \le m$ and $1 \le j \le n$,
   $L_{i,j} = Max\{L_{i-1,j}, L_{i,j-1}\}$ if $a_i \ne b_j$, for $1 \le i \le m$ and $1 \le j \le n$,
   $L_{0,0} = L_{0,j} = L_{i,0} = 0$, for $1 \le i \le m$ and $1 \le j \le n$.

   (a) Suppose that $A =$ "*abcad*" and $B =$ "*bcbd*", write down the value of the L matrix. (5%)

   (b) Write a *nonrecursive* C program to calculate the L matrix. (10%)

   Answer: (a)
   0 0 0 0 0
   0 0 0 0 0
   0 1 1 1 1
   0 1 2 2 2
   0 1 2 2 2
   0 1 2 2 3

6. Write a *recursive* C function to determine if a binary tree is complete. The C function returns the depth of the tree if it is complete, and -2 if it is not complete, where the depth of a tree consisting of a single node is 0. (12%)

```
struct nodetype {
    int info;
    struct nodetype *left;
    struct nodetype *right;
}
int complete(struct nodetype *tree)
/*  *tree: root */
```

7. In a *binary search tree*, the left subtree of every node stores the numbers less than or equal to the number in the node. The right subtree stores the larger numbers. Write a *recursive* C function to insert a new number x into the binary search tree, if x is not in the tree. Do nothing if x is already in the tree. (12%)

```
struct nodetype {
    int info;
    struct nodetype *left;
    struct nodetype *right;
}
struct nodetype *insert(int x, struct nodetype *tree) or
void insert(int x, struct nodetype *tree)
/*  *tree: root */
```

   You can call the following function directly. In other words, you need not write the program of this function.
   *struct nodetype *getnode( )*: Allocate the storage of a tree node.