Dept. of Computer Science and Engineering, undergraduate
National Sun Yat-sen University
Data Structures - Final Exam., Jan. 13, 2014

1. Multiple choices (There may be zero or more correct answers. If there is no correct answer, you should write down "None".) (28%)
   Answer: (a) ABC (b) BD (c) BCD (d) BC (e) A (f) ACD (g)BCD

   (a) Build a *binary search tree* for the input sequence 9, 4, 8, 7, 10, 15, 14, 3, 2. It is assumed that the tree root is on level 1. (A) There are four levels in the tree. (B) 15 is on level 3. (C) 3 and 8 have the same father. (D) The left subtree of 10 has 2 nodes.

   (b) Which statement(s) is correct for a *right in-threaded binary tree* ? (A) The operations for inserting a new node as the left child and the right child are the same. (B) The right child of a node points to its inorder successor if the right child is empty. (C) The right child of a node points to its postorder successor if the right child is empty. (D) It can be traversed in inorder without a stack.

   (c) Suppose that the input sequence for the *bubble sort* is 2, 4, 1, 6, 5, 3, and the output sequence after sorting is nondecreasing. Which of the following is(are) impossible to appear in the sorting process. (A) 2, 1, 4, 5, 3, 6. (B) 1, 2, 4, 6, 5, 3. (C) 2, 1, 4, 6, 3, 5. (D) 1, 2, 4, 3, 6, 5.

   (d) Which statement(s) is correct? (A) The heap used in *heapsort* is a binary search tree. (B) The heap used in *heapsort* is an almost complete binary tree. (C) The heap used in *heapsort* can be implemented by an array. (D) In a *min heap*, the values stored in the nodes on upper levels are always less than or equal to those stored in the nodes on lower levels.

   (e) Which statement(s) is correct for a *B-tree* ? (A) All leaves are on the same level. (B) The number of elements stored in the root of a *B-tree* of order n is between $\lfloor \frac{n-1}{2} \rfloor$ and $n-1$. (C) A newly inserted element must be stored in a leaf node. (D) The height increases by one only when all nodes are full.

   (f) Which statement(s) is correct for a *general list* ? (A) An element in a general list may be a list. (B) The head of a general list cannot be an empty list. (C) If the *reference count* method is used to manage the storage of lists, an extra count field for each node is required. (D) In the reference count method, only when the count of a node becomes zero, it can be restored to the system storage.

   (g) Which statement(s) is correct for the *hashing* method? (A) Hashing is a fast sorting method. (B) If one element is inserted into the hash table without collision, then it can be searched successfully with only one comparison in the future. (C) It is difficult to remove an element from the hash table. (D) Linear probing is one of the methods for resolving hash collisions.

2. What are printed by each of the following C programs? (Total 16%: (a)4%, (b)4%, (c)8%)

(a)
```
void fun(int a[ ], int b[ , int c[ ], int d[ ])
{ printf("%d %d %d %d \n", a[3],b[3],c[3],d[3]); }
void main( )
{ int e[ ]={10,11,12,13,14,15,16,17,18,19,20};
fun(e,e+3,&e[2],&e[2]+3); }
```
answer: 13 16 15 18

(b)
```
union {
char m;
unsigned char n; }u;
u.n=183; printf("%d \n",u.m);
```
answer: -73

(c)
```
d[ ]={10,11,12,13,14,15,16,17};
int max(int i, int j){
   int n; int m=(i+j)/2;
   if (j==i) return d[i];
   else {
      if (max(i,m) > max(m+1,j)) n=max(i,m);
      else n=max(m+1,j);
      printf("%d %d %d \n", i,j,n); return n; }
}
void main( )
{ max(0,7); }
```

answer:
```
0 1 11
2 3 13
2 3 13
0 3 13
4 5 15
6 7 17
6 7 17
```

4 7 17

4 5 15

6 7 17

6 7 17

4 7 17

0 7 17

3. *Huffman algorithm* is a variable-length encoding method, in which a Huffman tree is constructed. Suppose seven symbols A, B, C, D, E, F, G with frequencies 8, 15, 6, 1, 15, 3, 7, respectively, are given. You have to obey the following rules for constructing the Huffman tree:

   (a) The label of each leaf node is a single symbol. The label of each internal node is the concatenation of the labels of its left child and right child.

   (b) When two nodes are merged, always set the node of label with less lexical order as the left child, and the other as the right child.

   (c) If more than two nodes have the same least frequency, then you have to merge the two that have the least and the second least label in lexical order.

   Please draw the full Huffman tree. (10%)

   answer:

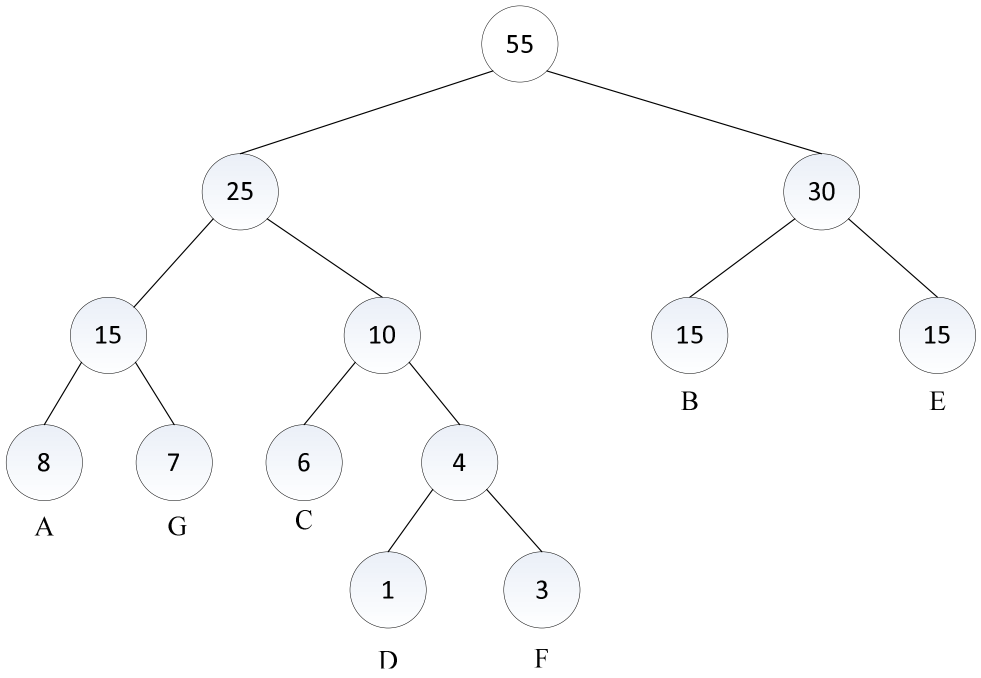

Figure 1: Full Huffman tree.

4. In the *buddy system*, a block of memory size 2^i is called an i-block, and the i-list consists of the starting addresses of free i-blocks. When one i-block of i-list is to be allocated, the one with the highest memory address is allocated first if more than one block is contained in i-list. Suppose that the total memory size in our computer system is 1024 bytes, whose starting address is 0.

   (a) If a 5-block starts at address p, what is the starting address of its buddy? (4%)

   (b) Suppose 9-list={0}, 8-list={512} and 7-list={768}. Which block (called block B_1) has been allocated? Please give its size and its starting address. (4%)

   (c) Under the allocation situation of the above subproblem (b), if we request four more blocks B_2, B_3, B_4 and B_5, with sizes 50, 50, 50 and 50, what is the content of each i-list, $6 \le i \le 9$? (4%)

   (d) Under the allocation situation of the above subproblem (c), after blocks B_1, B_3, B_4 and B_5 have been freed, what is the content of each i-list, $6 \le i \le 9$? (4%)

   Answer:
   (a) If p is a multiple of 64, then its buddy is p+32; otherwise, its buddy is p-32.
   (b) 7-block at 896, size 128 bytes
   (c) 6-list=empty, 7-list={512}, 8-list=empty, 9-list={0}
   (d) 6-list={768}, 7-list={896}, 8-list={512}, 9-list={0}

5. What are the *internal search* and *external search*. (6%)

6. Write a recursive C function to determine the *depth* of a binary tree. Note that the depth of a binary tree consisting of a single node is 0. (12%)

```
struct nodetype {
    int info;
    struct nodetype *left;
    struct nodetype *right;
}
int depth(struct nodetype *tree)
```

7. Write a *recursive* C function to perform the *merge sort*. To implement your merge sort, you can call the following *2-way merge* function as a basic function, which merges two sorted arrays into a single one. In other words, you need not write the 2-way merge function. (12%)

```
void twoway(int a[ ], int b[ ], int c[ ], int na, int nb)
```

```
/* a[ ] and b[ ] are input sorted arrays */
/* c[ ] is the output sorted array after a[ ] and b[ ] are merged */
/* na and nb are the lengths of a[ ] and b[ ], respectively */
```