Department of Computer Science and Engineering
National Sun Yat-sen University
Data Structures - Final Exam., Jan. 12, 2015

1. Multiple choices (There may be zero or more correct answers. If there is no correct answer, you should write down "None".) (24%)

   (a) Which statement(s) is correct for a binary search tree? (A) If a new element is equal to the middle of the elements already in the tree, then the new element is inserted into the root. (B) If we traverse the tree nodes with the inorder sequence, then the sequence is sorted. (C) If a node is to be deleted, it must be a leaf node. (D) The elements smaller than the element in the root are stored in the left subtree.

   (b) Which sorting algorithm(s) is a stable sorting method? (A) Heap sort (B) Quick sort (C) Merge sort (D) Radix sort.

   (c) Which sorting algorithm(s) has time complexity O($n\log n$) in the worst case? (A) Heap sort (B) Quick sort (C) Merge sort (D) Insertion sort.

   (d) Which statement(s) is correct for the hashing method? (A) Hashing is a fast sorting method. (B) If the linear probing method is used and element x is smaller than element y, then the position index for storing x must be smaller than that for y in the hash table. (C) If the linear probing method is used, then an element in the hash table can be deleted directly. (D) Linear probing is one of the methods for resolving hash collisions.

   (e) Which statement(s) is correct for an AVL tree? (A) A new node is always inserted as a leaf node. (B) LL rebalancing rotation is symmetric to RR rebalancing rotation. (C) In the LR rebalancing rotation, one of the tree nodes is moved up 2 levels. (D) An AVL tree is also a binary search tree.

   (f) Which statement(s) is correct for a B-tree of order m? (A) All leaves are on the same level. (B) The number of children of the root is between $\lceil m/2 \rceil$ and m. (C) The number of children of a node other than the root node and external nodes is between $\lceil m/2 \rceil$ and m. (D) If a new key is inserted into a node and the node is full, then the node will be split into two nodes.

2. What are printed by each of the following C programs? (12%)

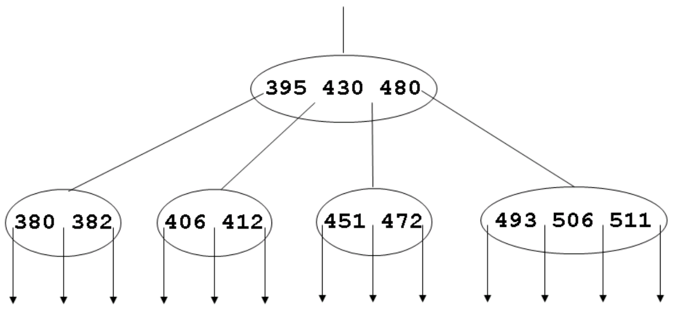
   (a)
```
void f(int a[ ], int b[ ], int *c, int d)
{    for (int i=0; i<=3; i++)
     {      a[i]+=20;    b[i]+=30;    *c+=100;      d=d+1000; }
}
```

```
int main( )
{    int e[ ]={10,11,12,13,14,15,16,17,18,19,20};
     f(e,e+3,&e[1],e[2]);
     printf("%d %d %d %d %d %d\n", e[0],e[1],e[2],e[3],e[4],e[5]); }
```

(b)
```
union {
   char m;
   unsigned char n;
}u;
u.n=186;     printf("%d \n",u.m);
u.m= 'A';    printf("%d \n",u.n);
```

3. The inorder sequence of a binary tree is BCAEDGHFI, and its preorder sequence is ABCDEFGHI. Please draw the tree. (6%).

4. Please draw the tree after 507 and 520 are inserted into the following B-tree of order 5. (6%)



5. Suppose that we have eight elements whose hashing function is given as follows:

k	*h*(*k*)
A0	100 000
A1	100 001
B4	101 100
B5	101 101
C1	110 001
C2	110 010
C3	110 011
C5	110 101

The directoryless dynamic hash table is used to insert the above elements, where each bucket has two slots. Initially, suppose that some elements have been inserted into the table as follows:

00	B4, A0
01	A1,B5
10	C2, -
11	C3, -

(a) Please show the hash table after the next element C5 is inserted. (5%)

(b) Please show the hash table after the next elements C5 and C1 are inserted. (5%)

6. Suppose 4,3,2,6,7,5 are inserted into an empty AVL in that order. Please draw the three trees after (a) 4,3,2; (b) 4,3,2,6,7; (c) 4,3,2,6,7,5 are inserted. (9%)

7. A red-black tree is a binary search tree in which every node is colored either red or black. There are three important properties for the colors of the nodes. Please describe the three properties. (9%)

8. Write a recursive C++ function to calculate the height of a binary tree. Note that the height of an empty tree is zero, and the height of a binary tree with a single node is one. (12%).

```
class TreeNode {
   int data;
   TreeNode *leftChild, *rightChild;;
};
int height( TreeNode *root)
// Return the height of a binary tree pointed by “root”.
// Return 0 if the binary tree is empty.
{
```

Please write the body of height ( ).

```
} // end of height ( )
```

9. Write a recursive C++ function to perform the recursive 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 body of 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 */
//You can call twoway(…) directly.

int merge_sort(….)    //complete the parameters by yourself
// merge_sort(….) is a recursive function.
{
```

Please write the body of the function.

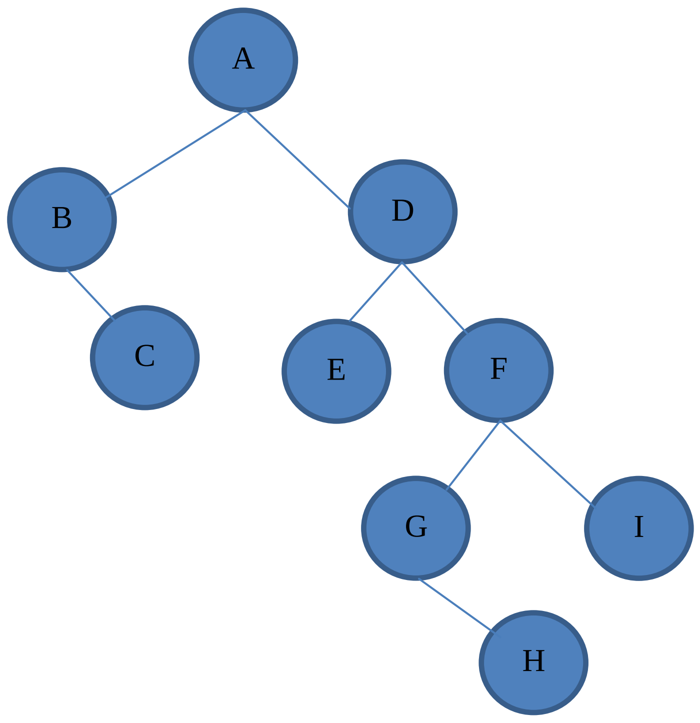
```
} // end of merge_sort ( )
```

Answer:

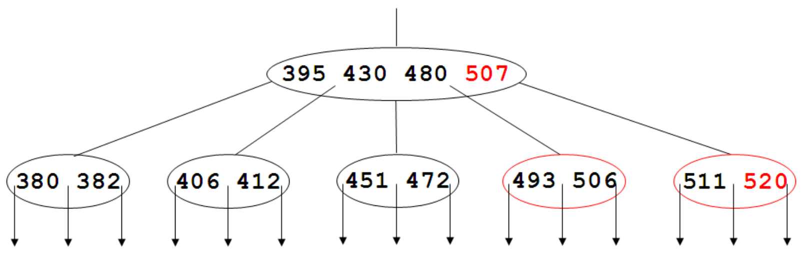
1. BD, CD, AC, D, ABCD, ACD
2. (a) 30 431 32 63 44 45

   (b) -70

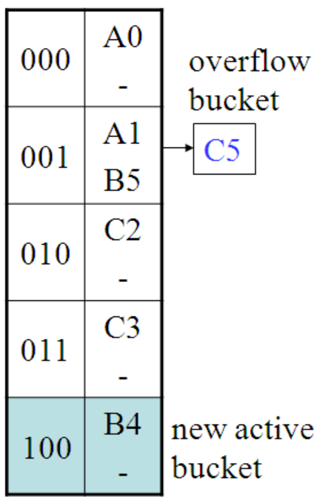
   65
3.



4.



5(a)



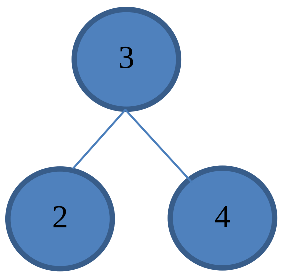
5(b)

000	A0 -
001	A1 C1
010	C2 -
011	C3 -
100	B4 -
101	B5 C5

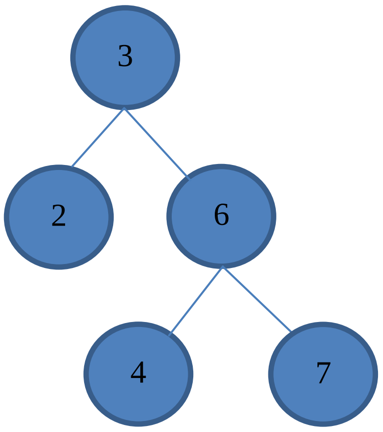
new active bucket

6.

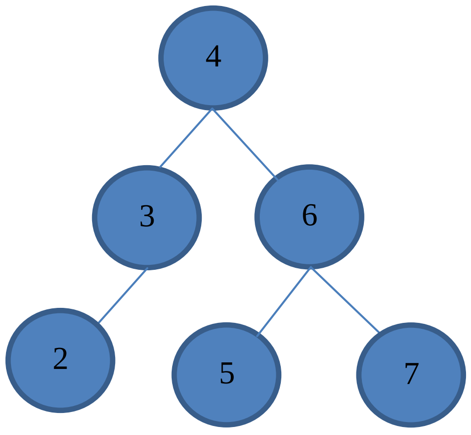
(a)



(b)



(c)



7.
RB1: The root and all external nodes are black.
RB2: No root-to-external-node path has two consecutive red nodes.
RB3: All root-to-external-node paths have the same number of black nodes.

8.

```
class TreeNode {
    int data;
    TreeNode *leftChild, *rightChild;;
};
int height( TreeNode *root)
{
    int leftH,rightH;
    if(root==0)
        return(0);     // Return 0 if the binary tree is empty.
    leftH=height(tree->left);
    rightH=height(tree->right);
    if (leftH>=rightH)
        return(leftH+1);              // left subtree is higher
    else
        return(rightH+1);              // right subtree is higher
} // end of height ( )
```

9.

```
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 */

int merge_sort(int unsorted[ ], int sorted[ ], int n)
// unsorted[ ]: input array,     sorted[ ]: output array,     n: number of elements
{
    if(n==1) {     // only one element
        sorted[0]=unsorted[0];
        return 0;       // return nothing
    }
    merge_sort(unsorted, a, n/2);    // sort the left half, sorted elements stored in a[ ]
    merge_sort(unsorted+n/2, b, n-n/2); // sort the right half, stored in b[ ]
    twoway(a, b, sorted, n/2, n-n/2);    // merge the left half and the right half
    return 0;       // return nothing
} // end of merge_sort ( )
```