Department of Computer Science and Engineering
National Sun Yat-sen University
Data Structures - Final Exam., Jan. 15, 2018

1. Multiple choices (There may be zero or more correct answers. If there is no correct answer, you should write down "None".) (16%)

   (a) Which sorting method(s) is stable? (A) insertion sort (B) quick sort (C) heap sort (D) merge sort.

   (b) Suppose that the size of the data structure is *n*. Which statement(s) is correct? (A) An insertion in a linear sorted array needs O(log*n*) time. (B) The binary search can be performed in a linearly linked list if the list is sorted. (C) A deletion in a linearly linked sorted list can be done in O(1) time if the node to be deleted is known. (D) The binary search in a linear sorted array needs O(log*n*) time.

   (c) Which statement(s) is correct for an AVL tree? (A) An AVL tree is a binary search tree. (B) When a new element is inserted, if a rotation operation is required, then the tree height is not changed. (C) When a new element is inserted, the tree height is increased if and only if the original tree is a full binary tree. (D) The level difference of every two nodes is at most 1.

   (d) Which statement(s) is correct for a threaded binary? (A) The right pointer of a node *u* points to the preorder successor of *u* if the right child of *u* is null. (B) The left pointer of a node *u* points to the inorder predecessor of *u* if the left child of *u* is null. (C) If the right child of a node *u* is not null, then the left pointer of the right child points to *u*. (D) If the left child of a node *u* is not null, then the left pointer of the left child points to *u*.

2. (a) What are printed by the following C program? (4%)

   (b) Explain the relation between the printed content and variable a or b. (6%)

```
int a=37; int b=43;
printf("%d \n", ((a+1)|a)-a);    // | is bitwise OR
printf("%d \n", ((b+1)|b)-b);
```

3. Construct an optimal binary search tree with data elements a_1=1, a_2=2, a_3=3, a_4=4 and searching probability 0.1, 0.5, 0.25, 0.15, respectively. Note that the probability for unsuccessful search is 0. In your construction, you have to draw the optimal binary search subtree for every two consecutive elements a_i and a_{i+1}, then draw the optimal binary search subtree for every three consecutive elements a_i, $a_{i+1}$and a_{i+2}, and finally give the solution of the optimal binary search tree for the four elements. (12%)

4. Suppose that the division method (mod 13) is used in the hashing function. The linear probing method is applied when a collision occurs. Given input numbers: 17,

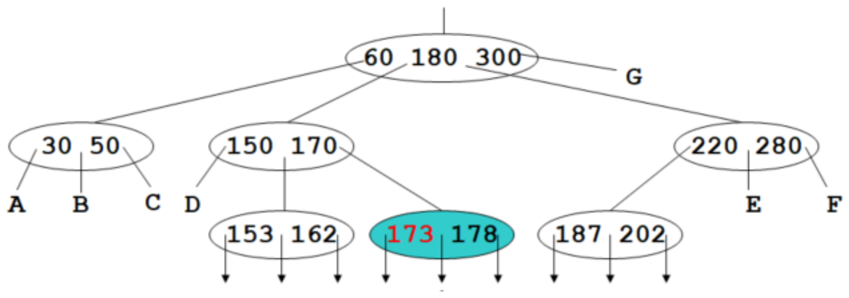
30, 18, 5, 4, 2, 15, please give the hash table after all numbers have been inserted into the table according to the input order. (10%)

5. A red-black tree is a binary search tree in which every node is colored either red or black.

   (a) There are three important properties for the colors of the nodes. Please describe the three properties for node definition. (6%)

   (b) Starting with an empty red-black tree, suppose the key insertion sequence is 20, 10, 5, 30, 40, 57, 3. Draw the red-black tree after each insertion, and indicate the node colors (R for red, and B for black). (6%).

6. Please draw the tree after 173 is deleted from the following B-tree of order 5. (6%)



7. Assume that an input file contains integers between p and q, with many numbers repeated several times. A distribution sort proceeds as follows. Declare an array a[ ] of size $q - p + 1$, and set $a[i - p]$ to the number of times that integer i appears in the file after you scan the input file. And then write the integers to the output file accordingly. Use the following input file to explain how this algorithm works: 9, 5, 7, 5, 8, 9, 10, 12, 5, 8. (10%)

8. Please write a *recursive* C++ function to print out the number stored in each node of a binary tree with the inorder traversal. (12%)

```
class TreeNode {
   int value;          // number stored in the node
   TreeNode *left, *right;    // left child and right child
};
void inorder(TreeNode *tree)
{
```

Please write the body of the function.

```
} // end of inorder( )
```

9. For each node in a binary tree, the *height balance* is defined as the height of the left subtree minus the height of the right subtree. Please write a *recursive* C++ function to calculate the height balance of each node, where the function returns the height of the subtree. (12%)

```
class TreeNode {
    int hb;          // value of height balance
    TreeNode *left, *right;    // left child and right child
};
int height_balance(TreeNode *tree)
// assign the value of the height balance to hb, and then return the tree height
// tree height of an empty tree is 0
{
```

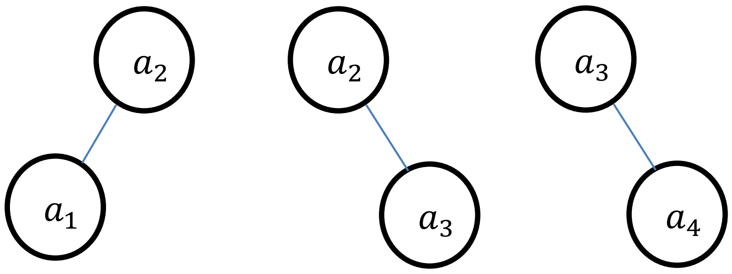
Please write the body of the function.

```
} // end of height_balance( )
```

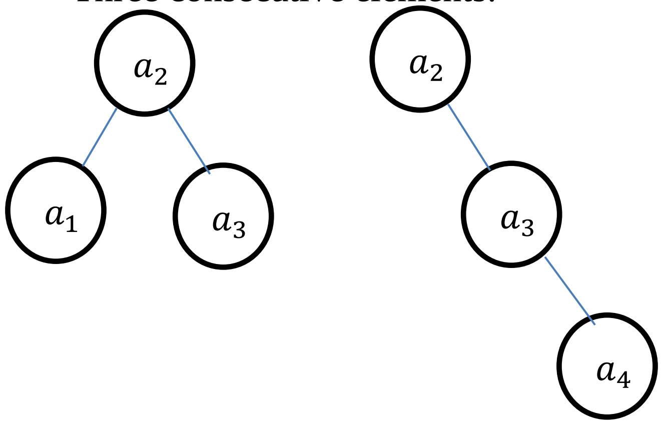
Answer:

1. AD, CD, AB, B
2. (a) 2, 4 (b) 所印出的值為 2 的次方，轉換成二進位來看，1 的位置就是 a 或 b 最右邊的 0 之位置。
3.

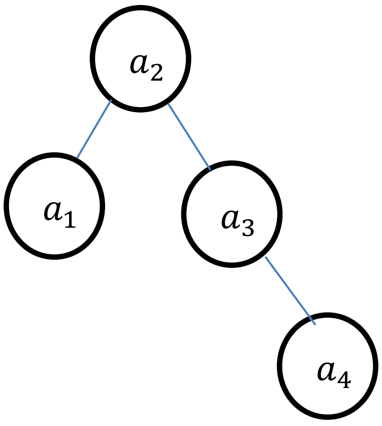
Two consecutive elements:



Three consecutive elements:



Final:



4. Hash table

0	1	2	3	4	5	6	7	8	9	10	11	12
		2	15	17	30	18	5	4				

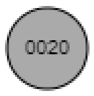
5. (a) 1.root 與 external node 都是黑色節點

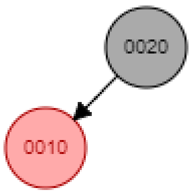
2. 從 root 到 external node 的路徑中，不會有兩個連續的紅節點

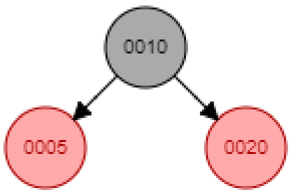
3. 所有 root 到 external node 的路徑中，黑色節點的數量都一樣

(b.)

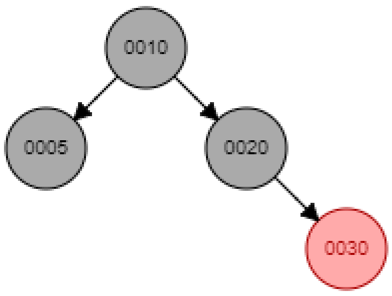
Insert 20: Insert 10: Insert 5:

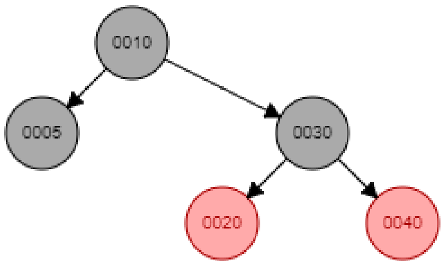




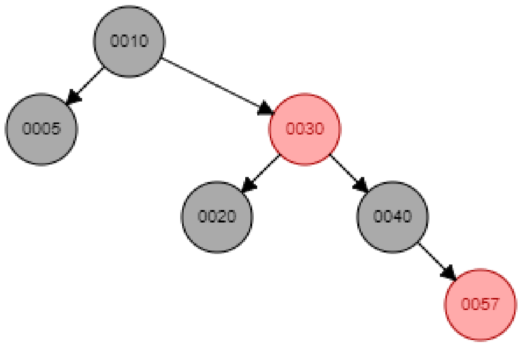


Insert 30: Insert 40:

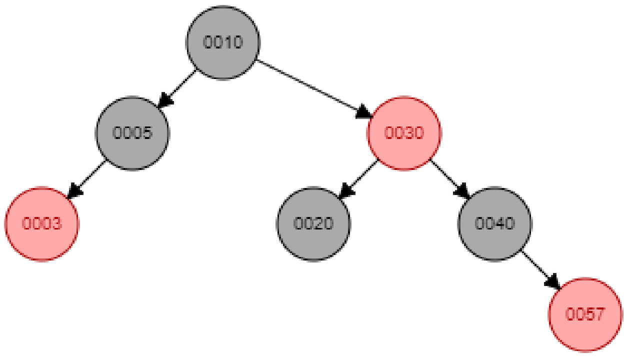




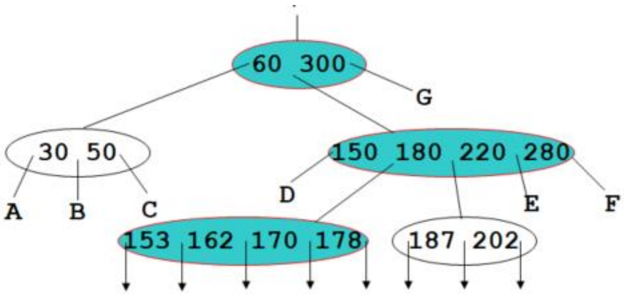
Insert 57:



Insert 3:



6. B tree



7. 首先利用 linear scan 得知 p=5 q=12 (欲排序的資料介於 5 與 12 之間)，因此宣告一個陣列 a[ ]，大小為 q-p+1=8，陣列中的元素初始值皆為 0，接著再次 linear scan，每讀進來一個整數 i，便把 a[i-p]的值加一，最後 a 陣列中的值如下：

a[0]=3 a[1]=0 a[2]=1 a[3]=2 a[4]=2 a[5]=1 a[6]=0 a[7]=1

陣列 a[ ]的每個元素 a[i]代表的是 i+p 數字出現的次數，只要 i 從 0 到 7，將數字 i+p 印出 a[i]次，所印出的序列便是排序好的。

最後的結果：5 5 5 7 8 8 9 9 10 12

8.

```
class TreeNode {
    int value;
    TreeNode *left, *right; // left child and right child
};
void inorder(TreeNode *tree)
{
    if(root==0)
        return ;      // Return if the binary tree is empty.
    inorder(tree->left);
    printf("%d\n", tree->value);
    inorder(tree->right);
} // end of inorder ( )
```

9.

```
class TreeNode {
    int hb;
    TreeNode *left, *right; // left child and right child
};
int height_balance(TreeNode *tree)
{
    int leftH,rightH;
    if(root==0)
        return(0);      // Return 0 if the binary tree is empty.
    leftH= height_balance(tree->left);
    rightH= height_balance(tree->right);
    tree->hb = leftH – rightH;
    if (leftH >= rightH)
        return(leftH+1);          // increase 1
    else
        return(rightH+1);
} // end of height_balance ( )
```