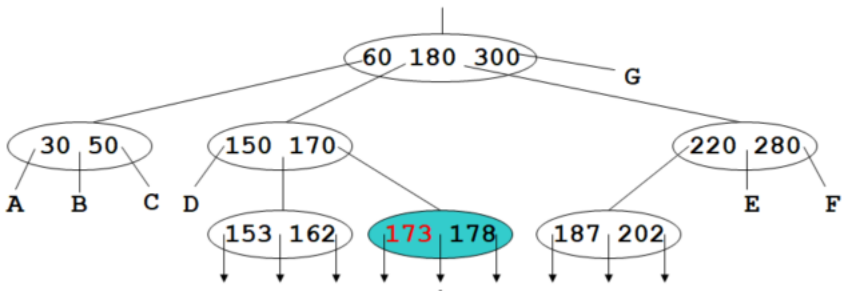
Department of Computer Science and Engineering
National Sun Yat-sen University
Data Structures - Final Exam., Jan. 4, 2021

1. Suppose that the inorder sequence of a binary tree is BCAEDGHFI, and the preorder sequence is ABCDEFGHI. Please draw the binary tree. (10%)

2. [3 2 8 5 7 1 4 6] is a permutation of [1 2 3 4 5 6 7 8]. Please give the two nontrivial permutation cycles in [3 2 8 5 7 1 4 6]. (6%)

3. (a) What is the main difference between B-trees and B^+-trees. (6%)

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



4. Starting with an empty *red-black tree*, suppose the key insertion sequence is 18, 10, 9, 25, 63, 90, 2. Draw the red-black tree after each insertion, and indicate the node colors (R for red, and B for black). (12%).

5. Let B_n denote the number of distinct binary trees with n nodes. Here, B_0=1, B_1=1, B_2=2. Please give the general recurrence formula to calculate B_n with some B_i, for $0 \le i \le n$-1. (10%)

6. There are n items, numbered as 1 through n, in a small store, where each item i has a profit p_i and a due day d_i, $1 \le i \le n$. The store can sell exactly one item each day. If item i is sold no later than day d_i, then the profit p_i can be earned. Here, the first day for the store is assumed to be day 1. Now, there are n=7 items with profits 3, 2, 8, 4, 1, 6, 9 and due days 2, 2, 4, 3, 5, 7, 4, respectively. Please give the maximum profit that can be earned by the store, and the item is sold in each day. (10%)

7. Explain each of the following terms. (16%)
   (a) linear probing
   (b) hash collision
   (c) complete binary tree
   (d) splay tree

8. Write a recursive C/C++ function to decide whether a given binary tree is an AVL tree or not. If it is an AVL tree, then return 1 as the answer; otherwise, return 0 as the answer. Note that the given tree must be a binary search tree. So you need not check the numbers storing in the tree. (12%)

```
class TreeNode {
    int data;      // the number stored in the node
    TreeNode *leftChild, *rightChild;
};
int AVL(….)
{
```

Please write the body of the function.

```
} // end of AVL( )
```

9. Write a recursive C++ function to perform *Quick Sort* with nondecreasing order. You can directly call swap(*x*,*y*) to exchange the values of *x* and *y*. (12%)

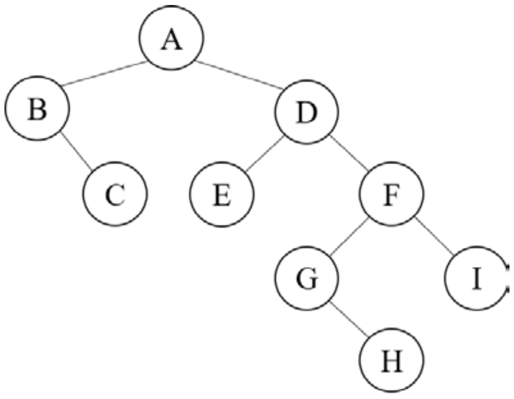
```
int a[100]; // global array for storing the elements to be sorted.
                  // The sorted elements are still stored in a [ ].
void swap(int &c, int &d)    // exchange the values of c and d
{ int temp;
  temp=c;    c=d;    d=temp;
}
void qsort(int left, int right) // a recursive function
// left, right: the left and right indexes of a[ ] to be sorted
{
```

Please write the body of the function.

```
} // end of qsort ( )
```

Answer:

1.



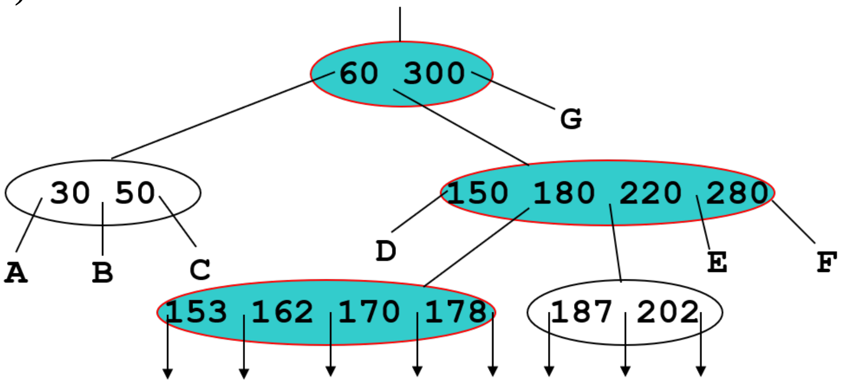
2.

(1, 3, 8, 6) and (4, 5, 7)

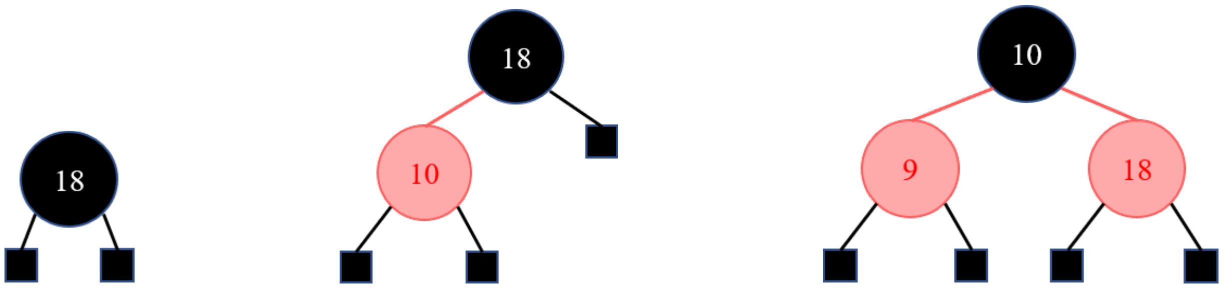
3.

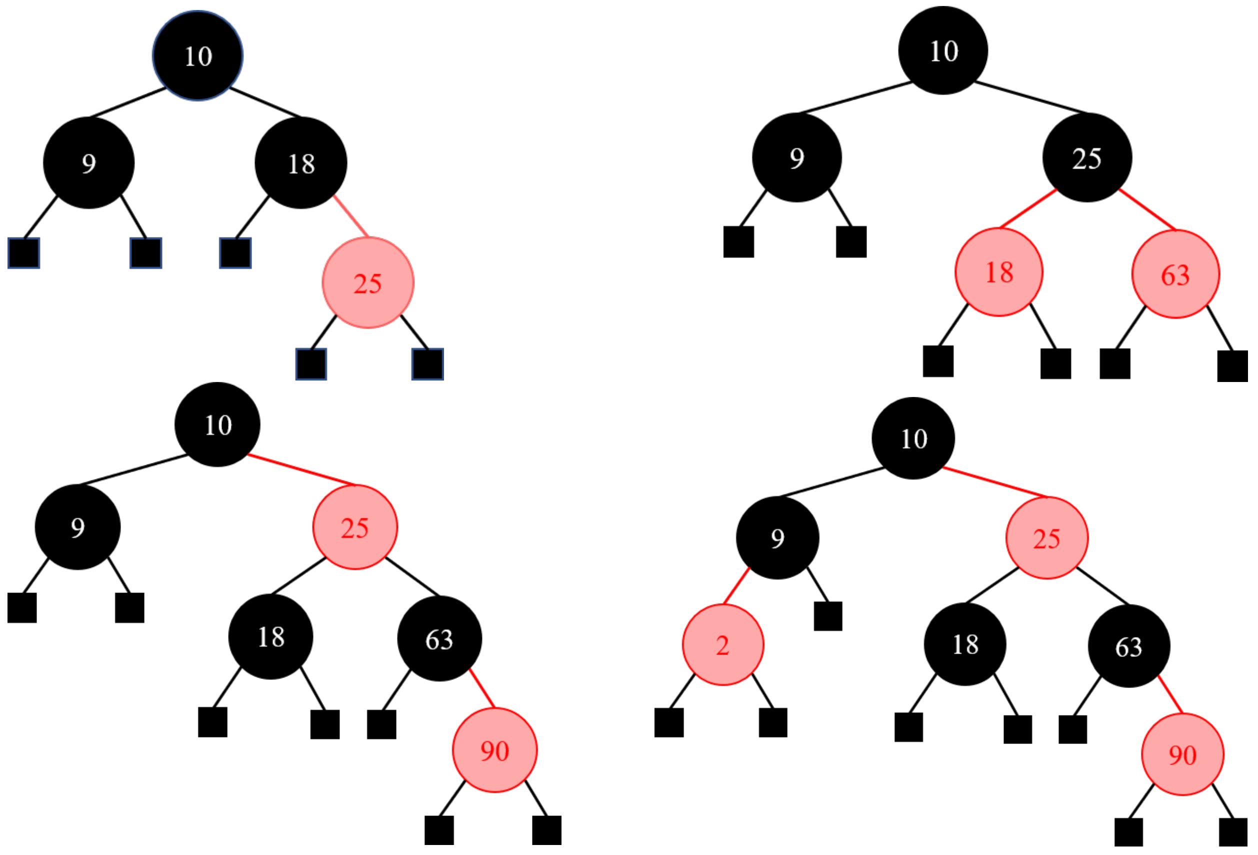
(a) B tree 的每一個 node(包含 internal node 與 leaf node)均有儲存資料，但是 B+tree 的資料僅儲存於 leaf node(internal node 沒有儲存資料)。此外，B+tree 的 leaf node 間以 linked list 串接在一起。

(b)



4.





5.

$$b_n = \sum_{i=0}^{n-1} b_i b_{n-i-1}, \quad n \ge 1, \text{and } b_0 = 1, b_1 = 1$$

6. Maximum profit =31

day	1	2	3	4	5	6	7
profit	3	4	8	9	1		6

7.

(a) linear probing:
經由 hash function 計算所得的 address，若發生碰撞(collision)，則嘗試下一個位置。若再度碰撞，則繼續嘗試下一個位置，以此類推，直到消除碰撞為止。

(b) hash collision:
經由 hash function 計算，兩筆資料欲放至相同位置，則產生碰撞(collision)。

(c) complete binary tree:
假設 tree 的深度為 d，則第 d-1 層以上均為全滿(均有 node)。最下面的一層(第 d 層)，若有空缺的 node，則必位於右方(左方無空缺的 node)。

(d) splay tree:
沒有存放 no balanced information 的 binary search tree。 進行搜尋時，會將被搜尋的 node 調整為 root，以方便往後可以較快搜尋到該 node。

8.

```
 // For each node of an AVL tree, the left subtree and the right subtree are
// both AVL trees, and the difference between the heights of the left subtree
// and the right subtree is no more than one.
  int AVL(TreeNode *root, int &height){
      // return 1 for AVL; return 0 for not AVL
      int leftHeight, rightHeight;    // heights of left subtree and right subtree
      if(root == NULL) {
           height=0;
           return 1;
      }
      int leftResult = AVL(root→leftChild, leftHeight);
      if(leftResult == 0) return 0;
              // left subtree is not AVL, need not calculate height
      int rightResult = AVL(root→rightChild, rightHeight);
      if(rightResult == 0) return 0;
              // right subtree is not AVL, need not calculate height
      int diff = leftHeight – rightHeight;
      if(diff >=2 || diff <=-2) return 0; // not AVL, need not calculate height
      height= max(leftHeight, rightHeight) + 1; // calculate height for AVL
      return 1;
 }
```

9.

```
 void qsort(int left, int right){
      if (left < ritght) {
           int i = left, j = right + 1, pivot = a[left];
           do{
                do i++; while (a[i] < pivot);
                do j--; while (a[j] > pivot);
                if (i<j) swap(a[i], a[j]);
           } while (i < j);
           swap(a[left], a[j]);

           qsort(left, j–1);
           qsort(j+1, right);
      }
 }
```