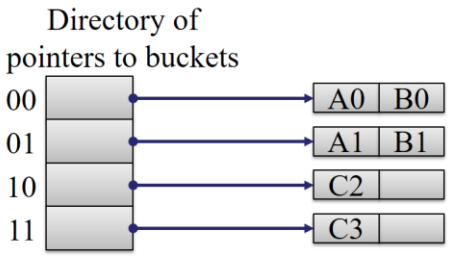
# Department of Computer Science and Engineering
# National Sun Yat-sen University
# Data Structures - Final Exam., Dec. 18, 2023

1. Let $b_n$ denote the number of distinct binary trees with $n$ nodes, where $b_0$=1, and $b_1$=1. Please present the recurrence formula for computing $b_n$. (10%)

2. Suppose that an input file contains positive 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%)

3. Suppose that there are 10 elements, each $k$ with a hashing function $h(k)$ provided in the left table. The dynamic hash table with a directory is used to insert these elements, where there are two slots in each bucket. Assume that some elements have been inserted into the hash table, as shown in the right figure.

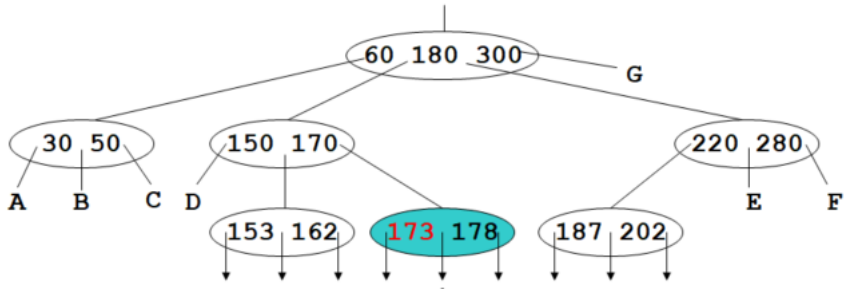
| $k$ | $h(k)$ | $k$ | $h(k)$ |
|---|---|---|---|
| A0 | 100 000 | B5 | 101 101 |
| A1 | 100 001 | C1 | 110 001 |
| B0 | 101 000 | C2 | 110 010 |
| B1 | 101 001 | C3 | 110 011 |
| B4 | 101 100 | C5 | 110 101 |




(a) Please draw the hash table after the next element C5 is inserted. (5%)
(b) Please draw the hash table after the next elements C5 and C1 are inserted. (5%)

4. (a) Assume that an initial red-black tree is empty. Please draw the tree after the numbers 25, 15, 10, 50, 70 are inserted into the tree sequentially, and indicate the node colors (R for red, and B for black). (5%)
   (b) Please draw the tree after the numbers 90, 6 are inserted into the above red-black tree obtained in (a), and indicate the node colors. (5%)

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




6. [3 2 8 5 7 1 4 6] is a permutation of [1 2 3 4 5 6 7 8], meaning that position 3 has to move to position 1, and so on. Please give the two nontrivial permutation cycles in

[3 2 8 5 7 1 4 6]. (6%)

7. Explain each of the following terms. (24%)
   (a) optimal binary search tree
   (b) stable sorting
   (c) hash collision
   (d) inorder successor
   (e) B+ tree
   (f) external path length

8. An AVL tree is a height-balanced tree, where the height difference of the left and right subtrees in each node is at most one. Write a recursive C/C++ function to decide whether a binary tree, pointed by a root, is height-balanced. (12%)

```
class TreeNode {
   int data;      // the number stored in the node
   TreeNode *leftChild, *rightChild;
};
int HBal(TreeNode *root, ...) or void HBal(TreeNode *root, ...)
//complete the parameters by yourself
// HBal is a recursive function
{
```

| Please write the body of the function. |
|---|

```
} // end of HBal( )
```

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.
{
```

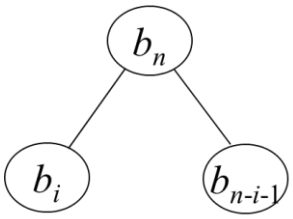
| Please write the body of the function. |
|---|

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

Answer:

1.

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




2. distribution sort

p 為資料之最小值 5，q 為資料之最大值 12。

a[i-p]存放 i 出現的次數。因此，a[5-5]=a[0]存放 5 出現之次數

a[1] 存放 6 出現之次數，…，a[7] 存放 12 出現之次數。

a[ ]初始化時，全部均為 0。

掃描每個資料時，在該資料對應的位置增加 1。

資料內容為 9, 5, 7, 5, 8, 9, 10, 12, 5, 8

Step 1: 輸入資料為 9，a[i-p] = a[9-5] = a[4]，將 a[4] 增加 1

Step 2: 輸入資料為 5，a[i-p] = a[5-5] = a[0]，將 a[0] 增加 1

…

Step 10:輸入資料為 8，a[i-p] = a[8-5] = a[3]，將 a[3] 增加 1

經過掃描後，a 陣列的結果如下

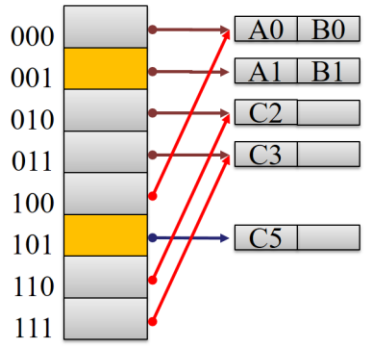
| i | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|
| i-p | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| a[i-p] | 3 | 0 | 1 | 2 | 2 | 1 | 0 | 1 |

a[i-p]代表出現之次數。然後，依序將 a[i-p]的資料輸出。a[5-5]=a[0]=3 代表，5 出現 3 次，以此類推。

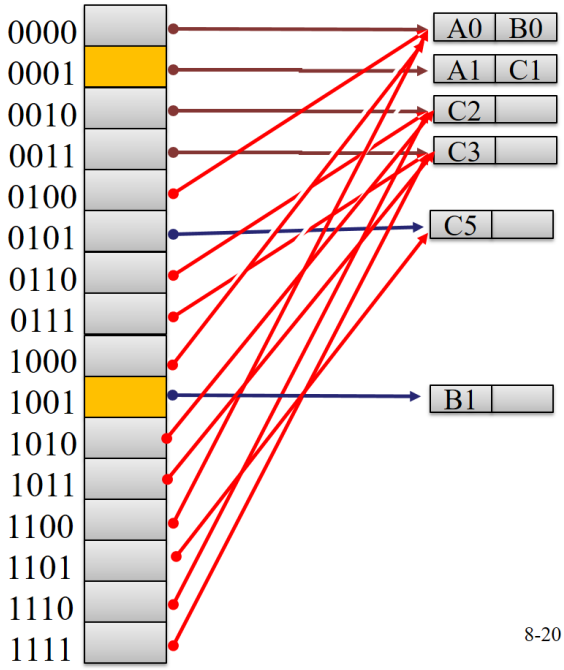
因此 output = 5, 5, 5, 7, 8, 8, 9, 9, 10, 12

3. hash table

(a)



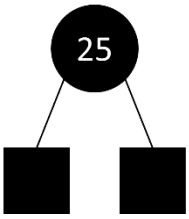

(b)



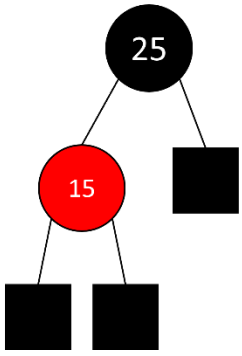

4. Red-black tree

(a)

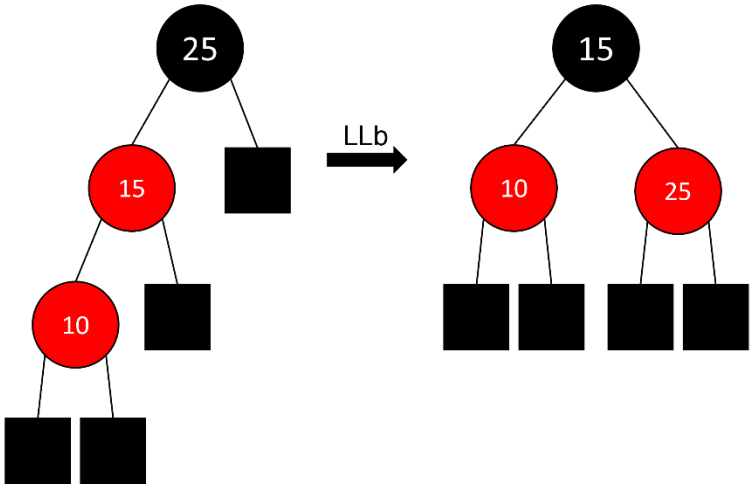
1. Insert 25



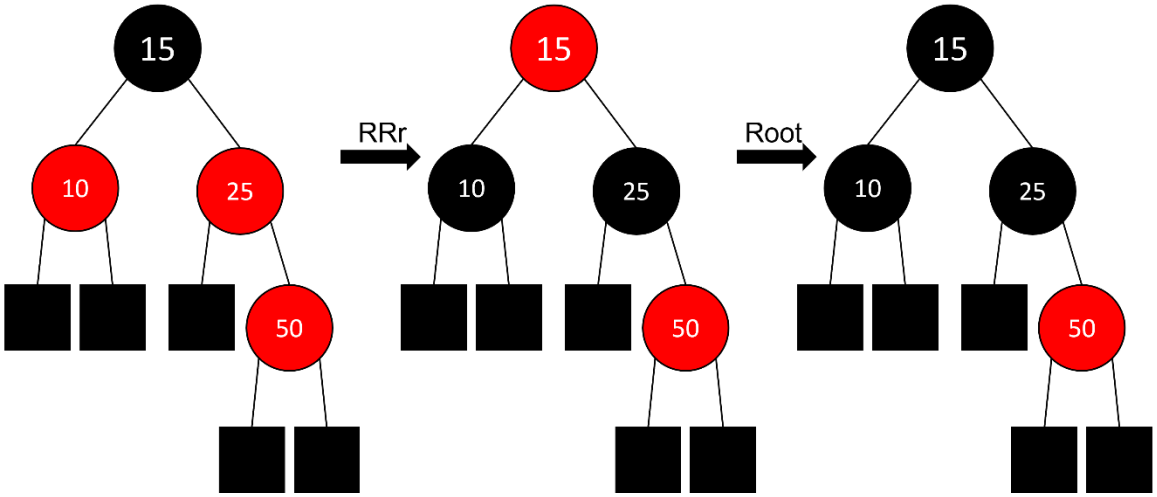

2. Insert 15



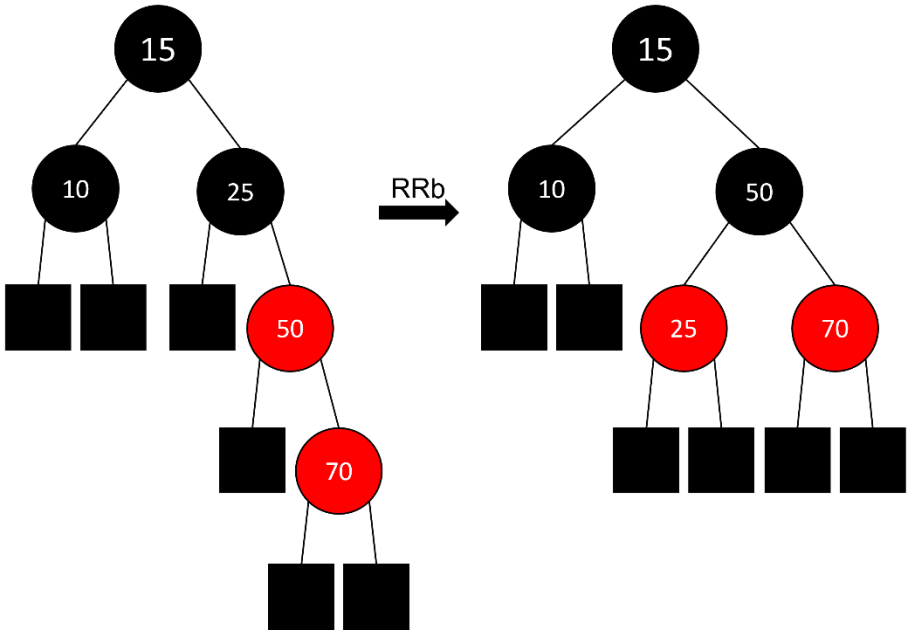
# 3. Insert 10




# 4. Insert 50

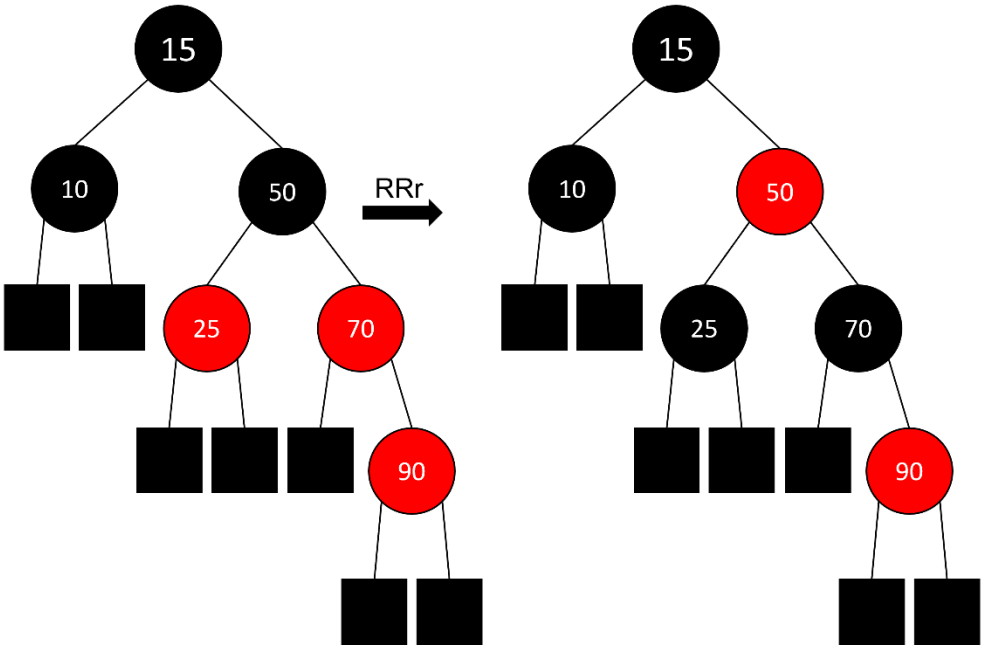


5. Insert 70

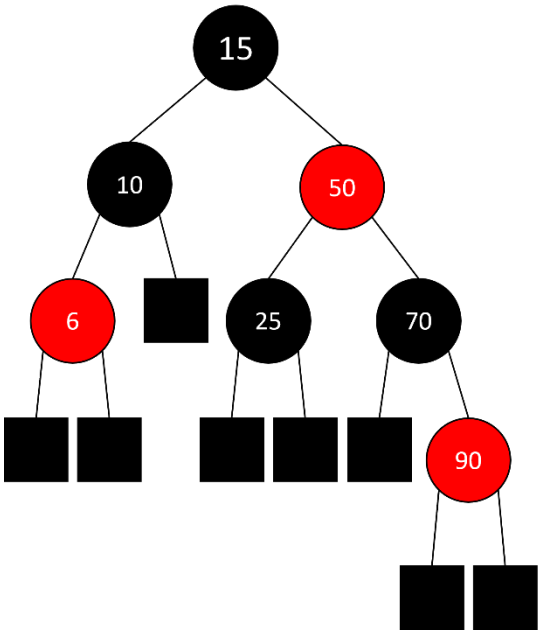



(b)

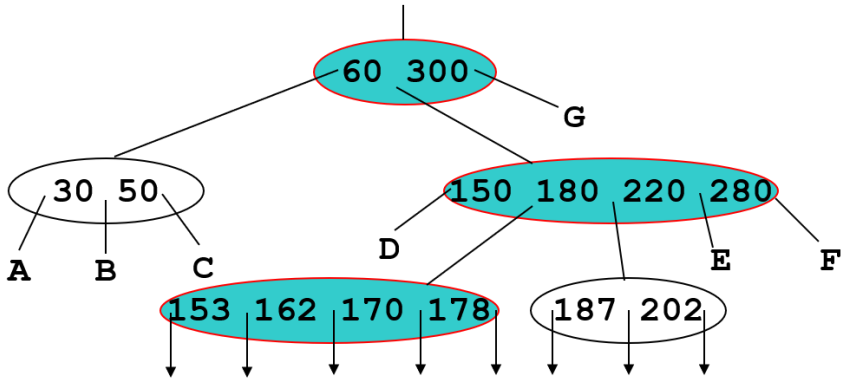
6. Insert 90



7. Insert 6




5. B tree




6. permutation cycle
(1, 3, 8, 6)  (4, 5, 7)

7.

(a) optimal binary search tree: a binary search tree that minimizes the total cost.

(b) stable sorting: the records with the same key have the same relative order as they have before sorting.

(c) hash collision: 經由 hash function 計算，兩筆資料欲放至相同位置，而該位置已無空位，則產生碰撞(collision)

(d) inorder successor: the node that would be visited after node p when traversing the tree in inorder.

(e) B+ tree: B tree的變形，internal node放指標和分界點，而external node放所有

的資料，並用double linked list串起來

(f) external path length: sum of the distances of all external nodes from the root.

8.

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

9.

another solution: https://par.cse.nsysu.edu.tw/~cbyang/course/ds/merge_sort.txt

```
int merge_sort(int a[], int left, int right){      // sorting between left and right
    if(left < right){
        int mid = (left + right)/2;      // or    left + (right – left)/2;
        merge_sort(a, left, mid);
        merge_sort(a, mid+1, right);
        int c[right – left + 1];          // or    int *c = new int[right-left+1];
        twoway(a+left, a+mid+1, c, mid-left+1, right-mid);
        for(int i=0 ; i < right – left + 1 ; ++i)
                    // This step is needed; otherwise, array a[ ] remains unsorted,
                    // since the sorted result is stored in c[ ]
            a[i + left] = c[i];
    }
    return 1;      // end of conquer
```

```
}
---------------------------------or-----------------------------
int merge_sort(int a[], int n){    // n is the length of a[ ]
      if(n==1) return 1;      // stop dividing
      merge_sort(a, n/2);
      merge_sort(a + n/2, n – n/2);
      int c[n];                  // or    int *c = new int[n];
      twoway(a, a + n/2, c, n/2, n – n/2);
      for(int i=0 ; i < n ; ++i)
                        // This step is needed; otherwise, array a[ ] remains unsorted,
                        // since the sorted result is stored in c[ ]
            a[i] = c[i];
      return 1;                  // end of conquer
}


/* Wrong answer, 以下格式的答案會陷入無限循環 */
int merge_sort(int *a, int left, int right, int n){      // n is the length of a[ ]
      int mid = n/2;
      int *tmp = new int[n];      // or    (int*)malloc(sizeof(int)*n));
      if(left < right){
            merge_sort(a, left, mid, mid+1);
            merge_sort(a, mid+1, right, n – mid – 1);
            twoway(a+left, a+right, tmp, mid, n – mid);
      }else{
            return 1;    // stop dividing
      }
      return 1;           // end of conquer
}
/*
Suppose the input is given as follows. Then you will get a wrong answer.
int a[10] = {…};
merge_sort(a, 0, 9, 10);
→ mid = 10/2 = 5 → merge_sort(a, 0, 5, 6);
→ mid = 6/2 = 3 → merge_sort(a, 0, 3, 4);
→ mid = 4/2 = 2 → merge_sort(a, 0, 2, 3);
→ mid = 3/2 = 1 → merge_sort(a, 0, 1, 2);
→ mid = 2/2 = 1 → infinite loop
*/
```