Department of Computer Science and Engineering
National Sun Yat-sen University
Data Structures - Final Exam., Jan. 10, 2022

1. Let b_n denote the number of distinct *binary trees* with n nodes. Please present the recurrence formula for computing b_n. (10%)

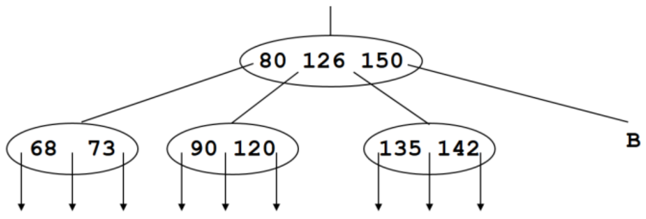
2. Explain the *radix sort* method with radix=10 by the input 13, 2, 16, 31, 8, 28, 4, 21, 6, 18. You should give the explanation of the radix sort method, and then present the list at the end of each pass. If you do not give the explanation, then you will get no point. (10%)

3. In the *hash* method, suppose that a rehash function $h_i(x)=(5i+x) \bmod 13$, $i \geq 0$, is applied sequentially when a new element is inserted into the hash table, and a hash collision occurs. In other words, In other words, $h_0(\,)$ is applied for the first time; $h_1(\,)$ is applied if a hash collision occurs (first collision); $h_2(\,)$ is applied if a hash collision occurs again (second collision); and so on. What is the content of the hash table after the elements 48, 22, 74, 11, 29, 32, 16, are inserted sequentially into an empty hash table of size 13, indexed as 0 through 12? (10%)

4. (a) Please give the definition of an *AVL* binary search tree. (5%)
   (b) Assume that the initial AVL tree is empty. Please draw the AVL tree after the numbers 11, 13, 8, 4 and 2 are inserted into the tree sequentially. (5%)
   (c) Please draw the AVL tree after the number 6 is inserted into the above AVL obtained in (b). (5%)

5. (a) For a B-tree of order m (m-way), how many children are there for each non-root node? How many children are there for the root node? (6%)
   (b) What is difference between a B-tree and B+-tree? (4%)

6. Please draw the tree after 120 is deleted from the following B-tree of order 5. (5%)



7. Explain each of the following terms. (16%)
   (a) threaded binary Tree
   (b) internal sorting
   (c) dynamic hashing
   (d) splay tree

8. Write a recursive C/C++ function to count the number of positive numbers (greater than zero) in a *binary tree*, where each node stores one integer number (not a binary search tree). (12%)

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

Please write the body of the function.

```
} // end of Positive( )
```

9. Please write a recursive C/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(...) or void merge_sort(...)
      //complete the parameters by yourself
      // merge_sort(...) is a recursive function.
{
```

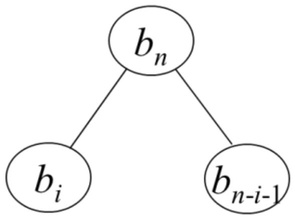
Please write the body of the function.

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

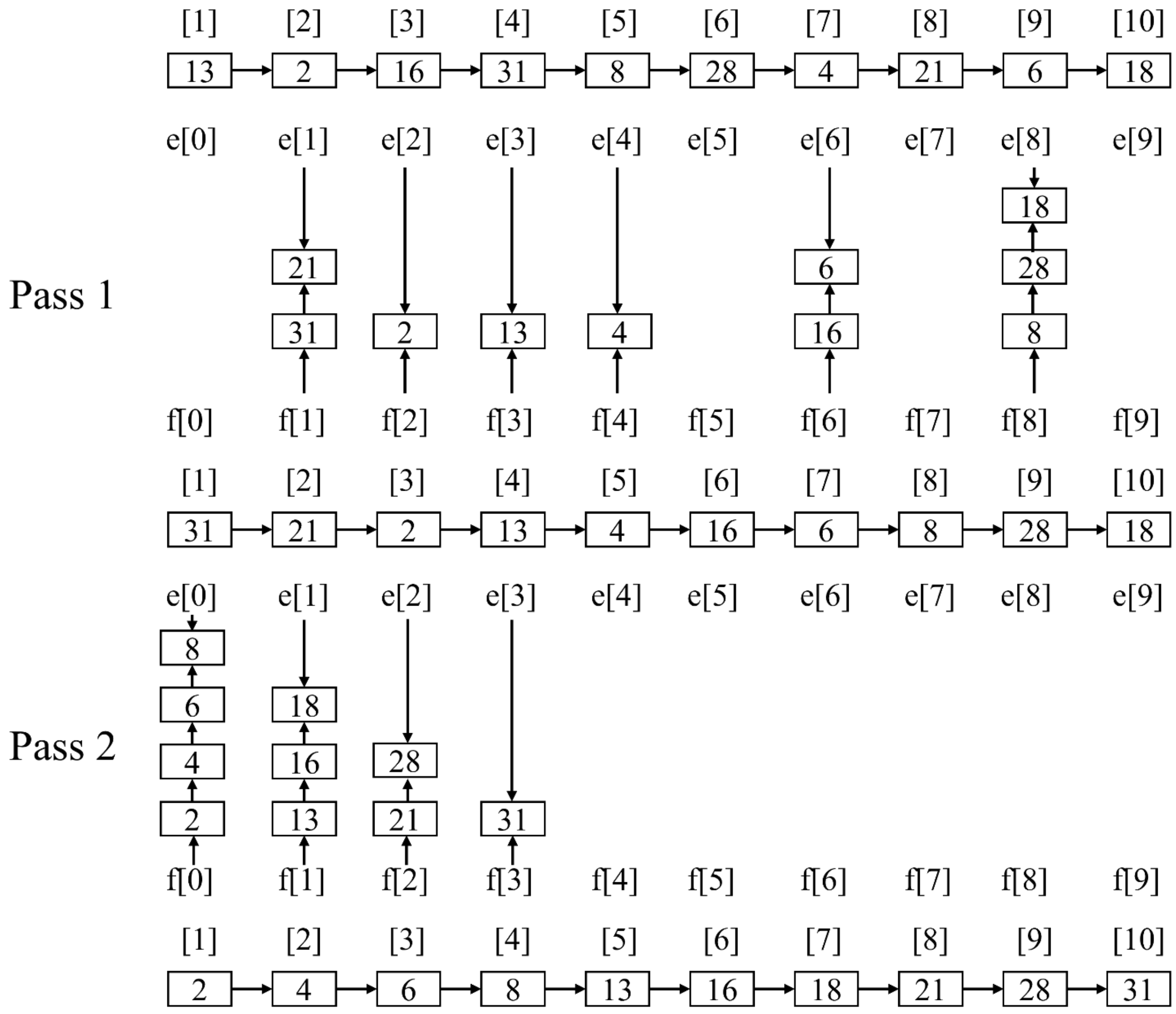
Answer:

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.



Radix sort: 每個資料不與其他資料比較，只看自己放在何處。從個位數開始處理若為 1 則放在 bucket 1、若為 2 則放在 bucket 2、…。接續十位數、百位數直至結束。

優點: 若以 array 處理，速度快

缺點: 若以 array 處理需要較多記憶體，若改以 linked list 則減少記憶體但增加時間

時間複雜度: $O((n+r)\log_r k)$，k 為 input 中最大的數，r 為基數(radix)，$\log_r k$為位數長度。

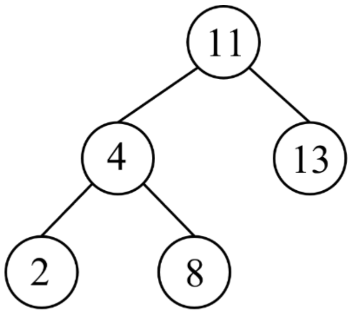
3. hash table

0	1	2	3	4	5	6	7	8	9	10	11	12
16	22		29			74		32	48		11	

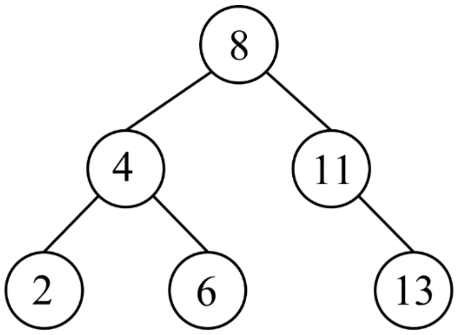
4. AVL tree

(a) 每一個 node，其 left subtree 與 right subtree 的高度差至多為 1。插入新資料或刪除資料時，若發生高度差超過 1 的情形，可使用 rotation 消除此情形。

(b)



(c)



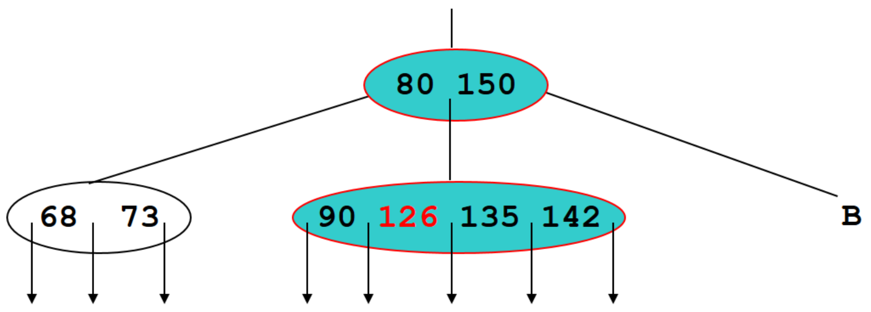
5.B-tree

(a) # of children of non-root node: $\left\lceil \frac{m}{2} \right\rceil$ ≤ # children of non-root ≤ m

# of children of root node: 2 ≤ # children of root ≤ m

(b) 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。

6. B-tree



7.
(a) 若 leftChild 為空，則指向其 inorder predecessor；若 rightChild 為空，則指向其 inorder successor。這樣的資料結構於進行 inorder traversal 時，可以不必使用 stack。
(b) 所有欲排序的資料，均存放於主記憶體，而無需使用輔助記憶體。
(c) Hash table 的大小，可以機動地調整而加大。當加入新資料時而發生 collision 時，可將 hash table 加大，但無需將已存在的元素全部重新 rehash(只需小範圍的 rehash)。
(d) 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.
```
class TreeNode {
    int data;       // the number stored in the node
    TreeNode *left, *right;
};
int Positive(TreeNode *root)
{
int leftP, rightP;
if( root == 0 )
        return 0;      // Return 0 if the binary tree is empty.
leftP = Positive(root->left );
rightP = Positive(root->right );
if ( root->data > 0)        // the root is positive
        return leftP + rightP +1;
else                // the root is not positive
        return leftP + rightP
} // end of Positive ( )
```

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)
          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
*/
```