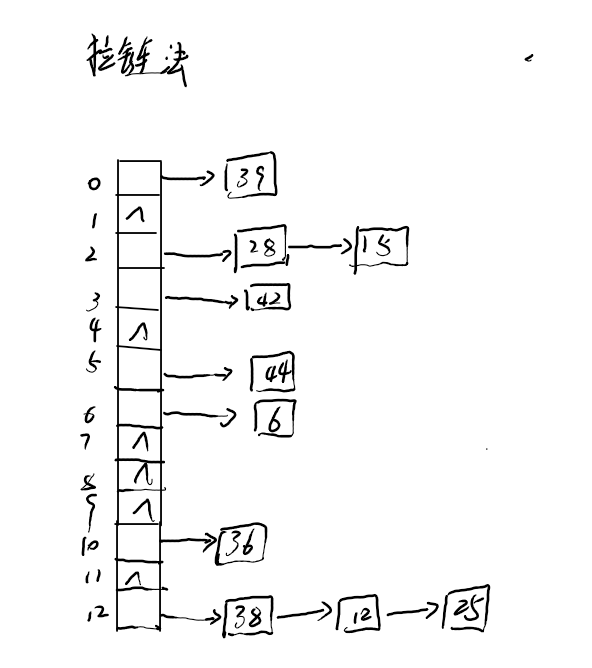
1.B 2.C 3.C 4.D 5.D 6.D 7.C 8.A 9.A

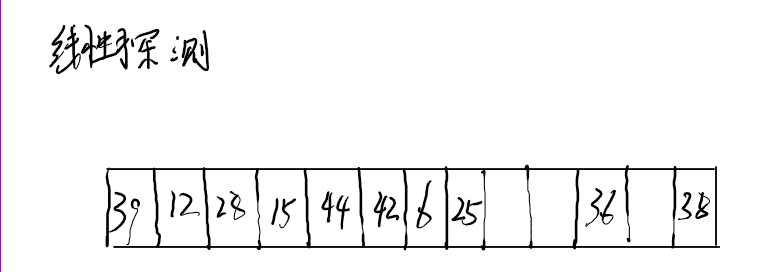
填空题

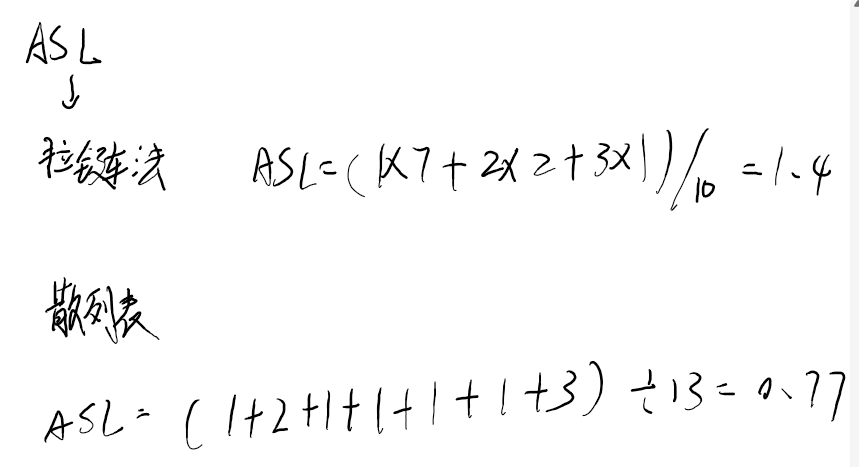
1. (n+1)/2 ,1 2.哈希表 3.顺序存储结构 4.15 5.log2n+1 6.O(n) O(log2n) O(n)

综合题

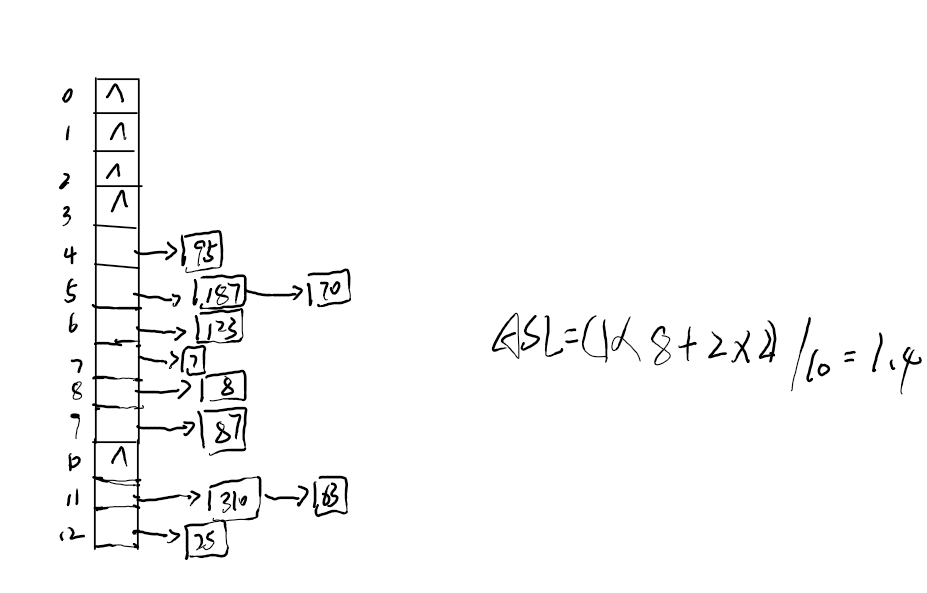
1.





、

2.



3.

void DiverSearch(int\* *Arr*,int *Low*,int *High*,int *value*)

{

    int Mid=(*Low*+*High*)/2;

    if(*Arr*[Mid]>*value*)

    {

*High*=Mid;

        DiverSearch(*Arr*,*Low*,*High*,*value*);

    }

    else if(*Arr*[Mid]<*value*)

    {

*Low*=Mid;

        DiverSearch(*Arr*,*Low*,*High*,*value*);

    }

    else

    {

        std::cout<<"Found it in "<<Mid<<std::endl;

    }

}

4.

bool isBST(TreeNode\* *root*) {

    if (*root* == nullptr) {

        return true;

    }

    Stack s;

    TreeNode \*current = *root*;

    TreeNode \*prev = nullptr; *// 用于记录前一个访问的结点*

    while (current != nullptr || !s.isEmpty()) {

*// 遍历到最左结点*

        while (current != nullptr) {

            s.push(current);

            current = current->left;

        }

*// 访问结点*

        current = s.pop();

*// 检查当前结点的值是否大于前一个结点的值*

        if (prev != nullptr && current->val <= prev->val) {

            return false;

        }

        prev = current;

*// 转向右子树*

        current = current->right;

    }

    return true;

}

5.

void outputNotLessThanX(TreeNode\* root, int x) {

if (root == nullptr) {

return;

}

TreeNode\* stack[100];

int top = -1;

TreeNode\* current = root;

while (current != nullptr || top != -1) {

while (current != nullptr) {

stack[++top] = current;

current = current->right;

}

current = stack[top--];

if (current->val < x) {

break;

}

// 输出当前结点的值

std::cout << current->val << " ";

// 转向左子树

current = current->left;

}

}

6. #include <vector>

using namespace std;

// 二叉树结点的定义

struct TreeNode {

int val;

TreeNode \*left;

TreeNode \*right;

TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

};

// 中序遍历，将结点值存入数组

void inorderTraversal(TreeNode\* root, vector<int>& nodes) {

if (root == nullptr) return;

inorderTraversal(root->left, nodes);

nodes.push\_back(root->val);

inorderTraversal(root->right, nodes);

}

// 合并两个有序数组

vector<int> mergeSortedArrays(const vector<int>& arr1, const vector<int>& arr2) {

vector<int> merged;

int i = 0, j = 0;

while (i < arr1.size() && j < arr2.size()) {

if (arr1[i] < arr2[j]) {

merged.push\_back(arr1[i++]);

} else {

merged.push\_back(arr2[j++]);

}

}

while (i < arr1.size()) merged.push\_back(arr1[i++]);

while (j < arr2.size()) merged.push\_back(arr2[j++]);

return merged;

}

// 根据有序数组构建平衡BST

TreeNode\* sortedArrayToBST(const vector<int>& nums, int start, int end) {

if (start > end) return nullptr;

int mid = start + (end - start) / 2;

TreeNode\* root = new TreeNode(nums[mid]);

root->left = sortedArrayToBST(nums, start, mid - 1);

root->right = sortedArrayToBST(nums, mid + 1, end);

return root;

}

// 合并两棵BST

TreeNode\* mergeBSTs(TreeNode\* root1, TreeNode\* root2) {

vector<int> nodes1, nodes2;

inorderTraversal(root1, nodes1);

inorderTraversal(root2, nodes2);

vector<int> merged = mergeSortedArrays(nodes1, nodes2);

return sortedArrayToBST(merged, 0, merged.size() - 1);

}

综合题

