考试常用排序算法

案例基于洛谷P1177：

将读入的 N 个数从小到大排序后输出。

输入格式

第一行为一个正整数 N。

第二行包含 N 个空格隔开的正整数 a_i，为你需要进行排序的数。

输出格式

将给定的 N 个数从小到大输出，数之间空格隔开，行末换行且无空格。

样例 #1

样例输入 #1

```
5
4 2 4 5 1
```

样例输出 #1

```
1 2 4 4 5
```

提示

对于 20% 的数据，有 $1 \leq N \leq 10^3$；

对于 100% 的数据，有 $1 \leq N \leq 10^5$，$1 \leq a_i \leq 10^9$。

使用不同排序方式解决问题

1. 冒泡排序$O(n^2)$

基本思想是通过不断交换相邻元素的位置来逐步将元素排序。它得名于排序过程中较大的元素像气泡一样“冒”到数组的顶端。

基本步骤：

1. **从头到尾遍历整个列表**，比较相邻的两个元素，如果前一个元素大于后一个元素，就交换它们的位置。
2. **重复上述过程**，每遍历一次列表，最大（或最小）元素就会被“冒泡”到数组的最后位置。
3. 继续对剩余未排序的部分重复以上步骤，直到整个数组排序完成。

代码：

```
#include <bits/stdc++.h>
```

```cpp
using namespace std;

#define maxN 100001
int arr[maxN], n;

//冒泡排序（传入左右指针）
void bubblesort(int *start, int *end);

//冒泡排序（传入数组指针与数组长度）
void bubblesort(int *arr, int len);

int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    bubblesort(arr, arr + n - 1);
    //bubblesort(arr, n);
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}

void bubblesort(int *arr, int len) {
    for (int i = len - 1; i >= 0; i--) {
        for (int j = 0; j < i; j++) {
            if (arr[i] < arr[j]) {
                swap(arr[i], arr[j]);
            }
        }
    }
}

void bubblesort(int *start, int *end) {
    for (int *i = end; i >= start; i--) {
        for (int *j = start; j <= i; j++) {
            if (*i < *j) {
                swap(*i, *j);
            }
        }
    }
}
```

2. 选择排序$O(n^2)$

基本思想是将数组分为已排序部分和未排序部分，找到未排序部分的最小值，放到已排序数组尾端；

基本步骤：

1. **遍历整个数组**，找出未排序部分中最小的元素。
2. 将该最小元素与未排序部分的第一个元素交换。
3. 重复上述过程，直到整个数组排序完成。

```cpp
#include <bits/stdc++.h>
using namespace std;

#define maxN 100001
int arr[maxN], n;

//选择排序
void selectionsort(int *start, int *end);

void selectionsort(int *arr, int len);

int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    selectionsort(arr, arr + n - 1);
    //selectionsort(arr, n);
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}

void selectionsort(int *arr, int len) {
    for (int i = 0; i < len; i++) {
        int min = i;
        for (int j = i + 1; j < len; j++) {
            if (arr[min] > arr[j]) {
                min = j;
            }
        }
        swap(arr[i], arr[min]);
    }
}

void selectionsort(int *start, int *end) {
    for (int *i = start; i <= end; i++) {
        int *min = i;
        for (int *j = i + 1; j <= end; j++) {
            if (*j < *i) {
                min = j;
            }
        }
        swap(*i, *min);
    }
}
```

3. 插入排序$O(n^2)$

```cpp
#include <bits/stdc++.h>
using namespace std;

#define maxN 100001
```

```cpp
int arr[maxN], n;

//选择排序
void insertSort(int *arr, int len);
void insertSort(int *start, int *end);

int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    insertSort(arr, arr + n - 1);
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}


void insertSort(int *arr, int len) {
    for (int i = 1; i < len; ++i) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            --j;
        }
        arr[j + 1] = key;
    }
}

void insertSort(int *start, int *end) {
    for (int *i = start + 1; i < end; i++) {
        int key = *i;
        int *j = i - 1;

        while (j >= start && *j > key) {
            *(j + 1) = *j;
            j--;
        }
        *(j + 1) = key;
    }
}
```

4. 快速排序$O(nlogn)$

快速排序是一种高效的排序算法，采用 **分治法**（Divide and Conquer）策略。它的基本思想是通过一个**基准元素**将数组分成两部分，其中一部分包含比基准元素小的元素，另一部分包含比基准元素大的元素，然后递归地对这两部分继续进行排序。

基本步骤：

1. **选择一个基准元素**：通常选取数组的第一个元素、最后一个元素或随机选择一个元素作为基准。

2. **划分操作**：将数组中的元素按照与基准元素的大小关系分成两个子数组——一个子数组包含所有小于基准的元素，另一个子数组包含所有大于基准的元素。
3. **递归排序**：递归地对这两个子数组分别进行快速排序。

```
#include <bits/stdc++.h>
using namespace std;

#define maxN 100001
int arr[maxN], n;

//快速排序
int partition(int *start, int *end);
void quicksort(int *start, int *end);

int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    selectionsort(arr, arr + n - 1);
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}

int partition(int *start, int *end) {
    int *pivot = start;
    int *i = start;
    for (int *j = start + 1; j <= end; j++) {
        if (*j < *pivot) {
            i++;
            swap(*i, *j);
        }
    }
    swap(*start, *i);
    return i - start;
}

void quicksort(int *start, int *end) {
    if (start >= end) {
        return;
    }
    int p = partition(start, end);
    quicksort(start, start + p - 1);
    quicksort(start + p + 1, end);
}
```

5. 基于二叉搜索树的快速排序$O(n^2)$

基本思路类似于快速排序，只不过使用二叉搜索树的形式，二叉搜索树的中序遍历具有有序性

```
#include <bits/stdc++.h>
using namespace std;

struct node
{
    int val;
    node *left, *right;
    node (int v = 0, node *n1 = nullptr, node* n2 = nullptr): val(v), left(n1),
right(n2){}
};

//中序遍历二叉树
void print(node *);
//前序遍历找到插入点；
void addNode(node *&, int);
//后序遍历删除所有节点
void clean(node *&);

int main()
{
    int n, val;
    cin >> n >> val;
    node *root = new node(val);
    for (int i = 1; i < n; i++)
    {
        cin >> val;
        addNode(root, val);
    }
    print(root);

    return 0;
}

void print(node *root)
{
    if (!root) return;
    print(root->left);
    printf("%d ", root->val);
    print(root->right);
}

void addNode(node*& root, int val)
{
    if (!root) root = new node (val);
    else {
        if (val >= root->val){
            addNode(root->right, val);
        }
        else if(val < root->val){
            addNode(root->left, val);
        }
```

```
    }
}

void clean(node *&root){
    if (root) return;
    clean(root->left);
    clean(root->right);
    delete root;
}
```

模块化

```
#include <bits/stdc++.h>
using namespace std;

class TreeNode {
public:
    int value;
    TreeNode *left;
    TreeNode *right;

    TreeNode(const int val) : value(val), left(nullptr), right(nullptr) {}
};


class BinarySearchTree {
private:
    TreeNode *root;

    static TreeNode *insertRecursive(TreeNode *node, int value);
    static void inOrderRecursive(const TreeNode *node);
    static void deleteNode(TreeNode *&r);

public:
    BinarySearchTree() : root(nullptr) {}
    ~BinarySearchTree() {
        clean();
    }

    void insert(const int value) {
        root = insertRecursive(root, value);
    }

    void inOrderTraversal() const {
        inOrderRecursive(root);
    }
    void clean() {
        deleteNode(root);
        root = nullptr;
    }

};

TreeNode *BinarySearchTree::insertRecursive(TreeNode *node, int value) {
    if (node == nullptr) {
```

```
        return new TreeNode(value);
    }

    if (value < node->value) {
        node->left = insertRecursive(node->left, value);
    } else{
        node->right = insertRecursive(node->right, value);
    }
    return node;
}

void BinarySearchTree::inOrderRecursive(const TreeNode *node) {
    if (node != nullptr) {
        inOrderRecursive(node->left);
        printf("%d ", node->value);
        inOrderRecursive(node->right);
    }
}

void BinarySearchTree::deleteNode(TreeNode *&r) {
    if (r == nullptr) {
        return;
    }
    deleteNode(r->left);
    deleteNode(r->right);
    delete r;
    r = nullptr;
}

int main() {
    BinarySearchTree bst;

    int n, val;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> val;
        bst.insert(val);
    }

    bst.inOrderTraversal();

    return 0;
}
```

这篇文章是前几天学二叉树的时候写的，顺带复习排序算法，尤其是冒泡排序、插入排序、选择排序考试更加常考；

非常稚嫩的一篇文章，有许多潜在的问题。希望大家帮忙勘误，找出文章中的错误或者表述不严谨的地方。

也希望阅读这篇文章的同学收益！

祝大家程序设计基础考试顺利！

作者：arctan37