排序

TEACHING COURSEWARE POWPOINT

授课时间：2025.08.06

目录

PART-01 选择排序 TEACHING COURSEWARE

PART-02 插入排序 TEACHING COURSEWARE

PART-03 计数排序 TEACHING COURSEWARE

PART-04 总结 TEACHING COURSEWARE

01

选择排序

TEACHING
COURSEWARE

TEACH

选择排序

选择排序
概念：选择排序是一种简单直观的排序算法。
它的基本思想是：每次从待排序的数据元素中选出最小（或最大）的一个元素，存放在序列的起始位置，然后再从剩余的未排序元素中寻找到最小（大）元素，然后放到已排序的序列的末尾。以此类推，直到全部待排序的数据元素排完。
基础操作：
在未排序序列中找到最小元素，存放到排序序列的起始位置
从剩余未排序元素中继续寻找最小元素，放到已排序序列的末尾
重复第二步，直到所有元素均排序完毕

选择排序

第一轮（i=1）：
从索引 0 开始，寻找未排序部分 [64, 25, 12, 22, 11] 中的最小值
最小值是 11（索引 4），与索引 0 的元素 64 交换
数组变为：[11, 25, 12, 22, 64]（前 1 个元素已排序）
第二轮（i=2）：
从索引 1 开始，寻找未排序部分 [25, 12, 22, 64] 中的最小值
最小值是 12（索引 2），与索引 1 的元素 25 交换
数组变为：[11, 12, 25, 22, 64]（前 2 个元素已排序）
第三轮（i=3）：
从索引 2 开始，寻找未排序部分 [25, 22, 64] 中的最小值
最小值是 22（索引 3），与索引 2 的元素 25 交换
数组变为：[11, 12, 22, 25, 64]（前 3 个元素已排序）
第四轮（i=4）：
从索引 3 开始，寻找未排序部分 [25, 64] 中的最小值
最小值是 25（索引 3），无需交换
数组变为：[11, 12, 22, 25, 64]（全部元素排序完成）

选择排序

稳定性：
选择排序是不稳定的排序算法。
例如，对于序列 [2，2，1]，第一次选择会将第一个 2 与 1 交换，导致两个 2 的相对顺序发生改变（原先是第一个 2 在第二个 2 前面，排序后变为第二个 2 在前面）。
时间复杂度：
最好情况：
O(n^2)（即使数组已排序，仍需遍历所有元素寻找最小值）
最坏情况：
O(n^2)
（逆序数组）
平均情况：O(n^2)
选择排序的时间复杂度不受输入数据分布影响，始终为二次方级，适合小规模数据。

选择排序

```cpp
#include <iostream>
using namespace std;
int arr[105];
int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    // 选择排序核心逻辑
    for (int i = 0; i < n - 1; i++) {
        // 寻找[i, n-1]范围内的最小值下标
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        // 交换最小值与当前位置元素
        int temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
    }
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

02

插入排序

TEACHING
COURSEWARE

TEACH

插入排序

插入排序
它的工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。

1. 从第一个元素开始，该元素可以认为已经被排序
2. 取出下一个元素，在已经排序的元素序列中从后向前扫描
3. 如果该元素（已排序）大于新元素，将该元素移到下一位置
4. 重复步骤 3，直到找到已排序的元素小于或者等于新元素的位置将新元素插入到该位置后
5. 重复步骤 2~5

插入排序

过程（以 [12, 11, 13, 5, 6] 为例）：
第 1 轮（i=1，key=11）：12>11 → 12 后移，插入 11 → [11, 12, 13, 5, 6]
第 2 轮（i=2，key=13）：12<13 → 直接插入 → [11, 12, 13, 5, 6]
第 3 轮（i=3，key=5）：13>5→13 后移，12>5→12 后移，11>5→11 后移，插入 5 → [5, 11, 12, 13, 6]
第 4 轮（i=4，key=6）：13>6→13 后移，12>6→12 后移，11>6→11 后移，5<6→插入 6 → [5, 6, 11, 12, 13]

插入排序每轮迭代都会将一个元素从 “未排序区间” 移到 “已排序区间” 的正确位置，该元素的最终位置就此确定。经过 n-1 轮迭代后，所有元素的位置都被确定，数组完成排序。

插入排序的每一轮都在 “扩大已排序区间” —— 从初始的第 1 个元素开始，每轮将 1 个新元素（未排序区间的第 1 个元素）插入到已排序区间的正确位置，因此每轮结束后，已排序区间的长度增加 1。
最终，当所有元素都被处理后，整个数组即为有序。

插入排序

插入排序稳定性：
插入排序是稳定的排序算法。

当比较元素时，仅将待插入元素插入到 “大于它” 的元素之前，若遇到相等元素则不移动，因此能保持原有相对顺序。例如，序列 [3，2，2] 排序后仍为 [2，2，3]，两个 2 的相对位置不变。
时间复杂度：最好情况：O(n)（数组已完全排序，只需遍历一次，无需移动元素）最坏情况：O(n^2)（数组完全逆序，每个元素都需插入到最前面）平均情况：O(n^2)插入排序对接近有序的数据效率很高，适合小规模或部分有序的数据。

插入排序


```cpp
#include <iostream>
using namespace std;
int arr[105];
int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    // 插入排序核心逻辑
    for (int i = 1; i < n; i++) {
        int key = arr[i];  // 待插入的元素
        int j = i - 1;     // 已排序区间的最后一个元素下标
        // 将大于key的元素向后移动
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;  // 将key插入到正确位置
    }
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

03

计数排序

TEACHING
COURSEWARE

TEACH

计数排序

计数排序
概念：计数排序是一种非比较型整数排序算法。它的核心在于将输入的数据值转化为键存储在额外开辟的数组空间中。作为一种线性时间复杂度的排序，计数排序要求输入的数据必须是有确定范围的整数。
基础操作：
找出待排序数组中最大和最小的元素
统计数组中每个值为 i 的元素出现的次数，存入计数数组的第 i 项
对计数数组中的元素进行累加，从计数数组的第一个元素开始，每一项和前一项相加
反向填充目标数组：将每个元素 i 放在新数组的第 count [i] 项，每放一个元素就将 count [i] 减去 1

计数排序

过程（以 [4, 2, 2, 8, 3, 3, 1] 为例）：
确定最大值为 8，创建计数数组 count[0..8]
统计次数：count = [0,1,2,2,1,0,0,0,1]（索引对应元素值）
计算前缀和：count = [0,1,3,5,6,6,6,6,7]（表示元素应放的位置）
反向填充：按原数组顺序将元素放入结果数组，最终得 [1,2,2,3,3,4,8]

前缀和的含义是 “小于等于当前元素的总个数”

计数排序


```cpp
#include <iostream>
#include <climits>
using namespace std;
int arr[105];
int count[100005];
int output[105];
int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
```

计数排序


```
// 步骤1：找出数组中的最大值和最小值
int min_val = 1e9;
int max_val = -1e9;
for (int i = 0; i < n; i++) {
    if (arr[i] < min_val) min_val = arr[i];
    if (arr[i] > max_val) max_val = arr[i];
}
int range = max_val - min_val + 1;
// 步骤2：统计每个元素出现的次数
for (int i = 0; i < n; i++) {
    // 偏移处理：将最小值映射到索引0，支持负数
    int index = arr[i] - min_val;
    count[index]++;
}
```

计数排序


```
// 步骤3：计算前缀和，确定每个元素的位置
for (int i = 1; i < range; i++) {
    count[i] += count[i - 1];
}
// 步骤4：从后往前遍历，构建输出数组（保持稳定性）
for (int i = n - 1; i >= 0; i--) {
    int index = arr[i] - min_val;
    output[count[index] - 1] = arr[i];
    count[index]--;
}
```

计数排序


```
    // 步骤5：将排序结果复制回原数组
    for (int i = 0; i < n; i++) {
        arr[i] = output[i];
    }
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

04

总结

TEACHING
COURSEWARE

TEACH

总结

排序的稳定性

冒泡排序，选择排序，插入排序，计数排序分别是否稳定，为什么