

# 排序方法

## 排序方法总览：

排序算法	时间复杂度 (平均 / 最坏)	空间复杂度	稳定性
冒泡排序 (Bubble Sort)	$O(n^2)$ / $O(n^2)$	$O(1)$	✓ 稳定
选择排序 (Selection Sort)	$O(n^2)$ / $O(n^2)$	$O(1)$	✗ 不稳定
插入排序 (Insertion Sort)	$O(n^2)$ / $O(n^2)$	$O(1)$	✓ 稳定
希尔排序 (Shell Sort)	$O(n^{1.3} \sim n^2)$ / $O(n^2)$	$O(1)$	✗ 不稳定
归并排序 (Merge Sort)	$O(n \log n)$ / $O(n \log n)$	$O(n)$	✓ 稳定
快速排序 (Quick Sort)	<b><math>O(n \log n)</math> / <math>O(n^2)</math></b>	$O(\log n)$	✗ 不稳定
堆排序 (Heap Sort)	$O(n \log n)$ / $O(n \log n)$	$O(1)$	✗ 不稳定
计数排序 (Counting Sort)	$O(n + k)$ / $O(n + k)$	$O(k)$	✓ 稳定
基数排序 (Radix Sort)	$O(d(n + k))$	$O(n + k)$	✓ 稳定
桶排序 (Bucket Sort)	$O(n + k)$ / $O(n^2)$	$O(n + k)$	✓ 稳定

✓ 稳定排序：冒泡、插入、归并、计数、基数、桶

✓ 原地排序 ( $O(1)$  空间)：冒泡、选择、插入、希尔、快排、堆排

⚡ 最快平均性能：快速排序

↔ 最稳定性 (最坏情况也好)：归并排序、堆排序

💾 空间最省：堆排序 ( $O(1)$ )

🎯 小规模最佳：插入排序

🎯 整数大量数据最佳：计数 / 基数 / 桶排序

# 一、冒泡排序：

重复遍历数组，比较每对相邻元素，如果顺序错误就交换它们。

每一趟遍历会把当前未排序区的最大（或最小）元素“冒泡”到区间的一端。

做  $n-1$  趟（或遇到某趟没有发生交换就提前结束）。



## 核心代码

```
// ===== 冒泡排序开始 =====
// 外层循环：控制趟数，共 n-1 趟
for (int i = 0; i < n - 1; i++) {
    // 内层循环：每一趟比较相邻元素
    for (int j = 0; j < n - 1 - i; j++) {

        // 如果前一个比后一个大 → 交换
        if (a[j] > a[j + 1]) {
            temp = a[j];
            a[j] = a[j + 1];
            a[j + 1] = temp;
        }
    }
}
// ===== 冒泡排序结束 =====
```

### 1. 外层 i 循环

决定需要冒泡的趟数

共执行  $n-1$  趟

### 2. 内层 j 循环

每一趟进行相邻元素比较

范围为  $0 \sim n-1-i$

因为每冒一次泡，最后一个元素已经是最大，无需再参与比较。

### 3. 交换操作

如果  $a[j] > a[j+1]$

就交换两个元素，让“更大”的往后走。

## 示例代码

### 题目描述

复制 Markdown

给出三个整数  $a, b, c (0 \leq a, b, c \leq 100)$ ，要求把这三位整数从小到大排序。

### 输入格式

输入三个整数  $a, b, c$ ，以空格隔开。

### 输出格式

输出一行，三个整数，表示从小到大排序后的结果。

### 输入输出样例

输入 #1	复制	输出 #1
1 14 5		1 5 14
输入 #2	复制	输出 #2
2 2 2		2 2 2

```
1 #include <stdio.h>
2 int main()
3 {
4     int a, b, c, i, j, t;
5     scanf("%d %d %d", &a, &b, &c);
6     int m[3]={a, b, c};
7     for(i=0;i<2;i++){
8         for(j=0;j<2-i;j++){
9             if(m[j]>m[j+1]){
10                 t=m[j];
11                 m[j]=m[j+1];
12                 m[j+1]=t;
13             }
14         }
15     }
16     for(i=0;i<3;i++){
17         printf("%d ", m[i]);
18     }
19     return 0;
20 }
```

# 二、选择排序

选择排序把数组分成两个区间：**左边是已排序区（初始为空），右边是未排序区（初始为整个数组）。**每一轮在未排序区中找到最小元素（或者最大），把它放到未排序区的起始位置（即追加到已排序区末尾）。重复  $n-1$  次后排序完成。  
**每轮做的是“选择最小并放到前面”，不是像冒泡那样交换相邻元素多次。**



## 核心代码

```
// 选择排序：每轮在未排序区 [i..n-1] 找最小元素，放到 i
for (int i = 0; i < n - 1; ++i) {
    int minidx = i; // 假定当前位置 i 为最小
    // 在未排序区寻找真正的最小索引
    for (int j = i + 1; j < n; ++j) {
        if (a[j] < a[minidx]) {
            minidx = j;
        }
    }
    // 将最小元素与位置 i 交换 (若 minidx != i)
    if (minidx != i) {
        int t = a[i];
        a[i] = a[minidx];
        a[minidx] = t;
    }
}
```

- ① **minidx** 用来记录当前未排序区的最小元素下标；
- ② 内层循环比较  $a[j] < a[minidx]$  找到最小值索引；
- ③ 交换只在必要时进行 ( $minidx \neq i$ )，避免多余赋值；
- ④ 交换为整元素交换，如果元素是大结构体，交换代价高，可考虑交换指针或索引数组。

## 示例代码

```
1 #include <stdio.h>
2 int main() {
3     int a[] = {64, 25, 12, 22, 11};
4     int n = sizeof(a) / sizeof(a[0]);
5     // 选择排序：每轮在未排序区 [i..n-1] 找最小元素，放到 i
6     for (int i = 0; i < n - 1; ++i) {
7         int minidx = i; // 假定当前位置 i 为最小
8         // 在未排序区寻找真正的最小索引
9         for (int j = i + 1; j < n; ++j) {
10            if (a[j] < a[minidx]) {
11                minidx = j;
12            }
13        }
14        // 将最小元素与位置 i 交换 (若 minidx != i)
15        if (minidx != i) {
16            int t = a[i];
17            a[i] = a[minidx];
18            a[minidx] = t;
19        }
20    }
21    printf("Sorted array: ");
22    for (int i = 0; i < n; ++i) printf("%d ", a[i]);
23    printf("\n");
24    return 0;
25 }
```

Sorted array: 11 12 22 25 64

请按任意键继续 . . . |

# 三、插入排序

把数组看成左侧已排序区和右侧未排序区。每次从未排序区取出第一个元素 `key`, 将它插入到左侧已排序区的正确位置 (通过将比 `key` 大的元素向右移动腾出位置), 使左侧仍保持有序。重复直到全部元素被“插入”。



## 核心代码

```
// 插入排序主循环: 从索引 1 开始, 把 a[i] 插入到 [0..i-1] 有序区
for (int i = 1; i < n; ++i) {
    int key = a[i];           // 要插入的值 (备份)
    int j = i - 1;

    // 将所有比 key 大的元素向右移动一位
    // 注意条件是 a[j] > key (而不是 >=), 这样可以保证稳定性:
    // 相等元素不会被移动 past key, 从而保持原序。
    while (j >= 0 && a[j] > key) {
        a[j + 1] = a[j]; // 右移元素
        j--;
    }
    // 插入 key 到位置 j+1 (j 最终是比 key 小或等于的索引)
    a[j + 1] = key;
}
```

## 示例代码

```
1 #include <stdio.h>
2
3 int main() {
4     int a[] = {5, 2, 9, 1, 5};
5     int n = sizeof(a) / sizeof(a[0]);
6
7     for (int i = 1; i < n; ++i) {
8         int key = a[i];
9         int j = i - 1;
10        while (j >= 0 && a[j] > key) {
11            a[j + 1] = a[j];
12            j--;
13        }
14        a[j + 1] = key;
15
16        // 打印当前轮次数组状态
17        printf("After inserting index %d: ", i);
18        for (int k = 0; k < n; ++k) printf("%d ", a[k]);
19        printf("\n");
20    }
21    return 0;
22 }
```

```
After inserting index 1: 2 5 9 1 5
After inserting index 2: 2 5 9 1 5
After inserting index 3: 1 2 5 9 5
After inserting index 4: 1 2 5 5 9
```

# 练习：

## 题目描述

[复制 Markdown](#) [中文](#) [展开](#) [进入 IDE 模式](#)

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。

## 输入输出样例

输入 #1	输出 #1
5 4 2 4 5 1	1 2 4 4 5

## 说明/提示

对于 20% 的数据, 有  $1 \leq N \leq 10^3$ ;

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

有以下程序, 用选择排序法对数组中的数据按由大到小排序。

```
void sort (int a[], int n)
{
    int i, j, k, t;
    for(i=0;i<n-1;i++)
    {
        _____(1)_____
        for(j=i+1;j<n;j++)
            if(a[j]>a[k])
                k=j;
        if(_____ (2) _____){
            t=a[k];
            _____(3)_____
            a[i]=t;
        }
    }
}
```

```
void main()
{
    int A[10], i;
    printf("enter the array\n");
    // 将通过键盘输入的 10 数据依次存放到数组中
    for(i=0;i<10;i++)
        _____(4)_____
    // 调用函数 sort 完成对数组 A 中所有元素的排序
    _____(5)_____
    printf("the sorted array: \n");
    for(i=0;i<10;i++)
        printf("%d ",A[i]);
    printf("\n");
}
```

(1) \_\_\_\_\_  $k = i;$

(2) \_\_\_\_\_  $a[k] > a[i]$

(3) \_\_\_\_\_  $a[k] = a[i];$

(4) \_\_\_\_\_  $\text{scanf}("%d", &A[i]);$

(5) \_\_\_\_\_  $\text{sort}(A,10);$