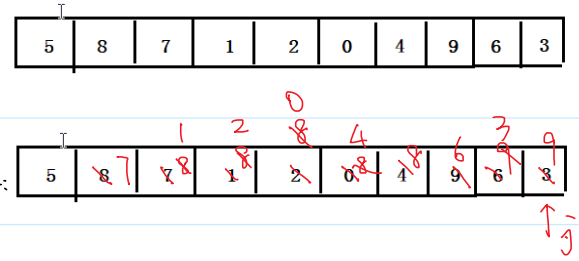
冒泡排序

2022年3月28日 9:50

冒泡排序




```
void bubble_sort(int arr[], int n) {
    // i表示冒泡的次数
    for (int i = 1; i < N; i++) {
        for (int j = 0; j < N - i; j++) {
            if (arr[j] > arr[j + 1]) { // 不能写成 >=
                swap(arr, j, j + 1);
            }
        }
        print_arr(arr, n);
    }
}
```

Microsoft Visual Studio 调试控制台

```
5 7 1 2 0 4 8 6 3 9
5 1 2 0 4 7 6 3 8 9
1 2 0 4 5 6 3 7 8 9
1 0 2 4 5 3 6 7 8 9
0 1 2 4 3 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
```

```
void bubble_sort(int arr[], int n) {
    // i表示冒泡的次数
    for (int i = 1; i < N; i++) {
        bool isSorted = true;
        for (int j =,0; j < N - i; j++) {
            if (arr[j] > arr[j + 1]) { // 不能写成 >=
                swap(arr, j, j + 1);
                isSorted = false;
            }
        }
        // 冒泡之后，判断isSorted
        if (isSorted) return;
        print_arr(arr, n);
    }
}
```

有序度++
逆序度--

Microsoft Visual Studio 调试控制台

```
5 7 1 2 0 4 8 6 3 9
5 1 2 0 4 7 6 3 8 9
1 2 0 4 5 6 3 7 8 9
1 0 2 4 5 3 6 7 8 9
0 1 2 4 3 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
```

时间复杂度：

最好情况：原数组有序，$O(n)$

最坏情况：原数组逆序 $O(n^2)$

比较次数：$(n-1)+(n-2)+\cdots+1 = \frac{n(n-1)}{2}$

交换次数：$(n-1)+(n-2)+\cdots+1 = \frac{n(n-1)}{2}$

平均情况：比较次数：大于交换的次数，小于 $\frac{n(n-1)}{2}$ $\Rightarrow O(n^2)$

交换次数：$\frac{n(n-1)}{4}$

有序度和逆序度

[3 2 1 4]

有序度：3

逆序度：3

有序度 + 逆序度 = $\frac{n(n-1)}{2}$

排序的过程：增加有序度，减少逆序度，

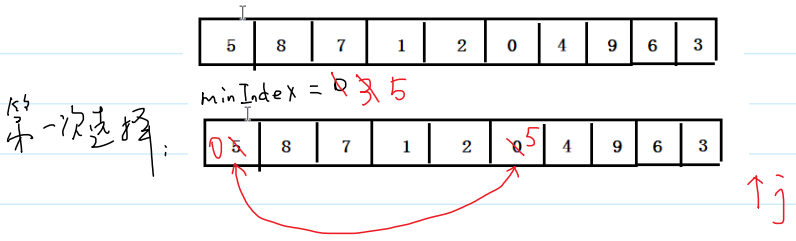
最终达到满有序度。

#2. 空间复杂度：$O(1)$ 原地排序

#3. 稳定性：稳定。 arr[j] > arr[j+1] 未发生交换。

选择排序

2022年3月28日 10:51




```c
void selection_sort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int minIndex = i - 1;
        for (int j = i; j < N; j++) {
            // 更新索引
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        // 交换i-1和minIndex所在位置的元素
        swap(arr, i - 1, minIndex);
        print_arr(arr, n);
    }
}
```

Microsoft Visual Studio 调试控制台

```
0 8 7 1 2 5 4 9 6 3
0 1 7 8 2 5 4 9 6 3
0 1 2 8 7 5 4 9 6 3
0 1 2 3 7 5 4 9 6 8
0 1 2 3 4 5 7 9 6 8
0 1 2 3 4 5 7 9 6 8
0 1 2 3 4 5 6 9 7 8
0 1 2 3 4 5 6 7 9 8
0 1 2 3 4 5 6 7 8 9
```

时间复杂度：$O(n^2)$

比较次数：$(n-1)+\cdots+1=\frac{n(n-1)}{2}$

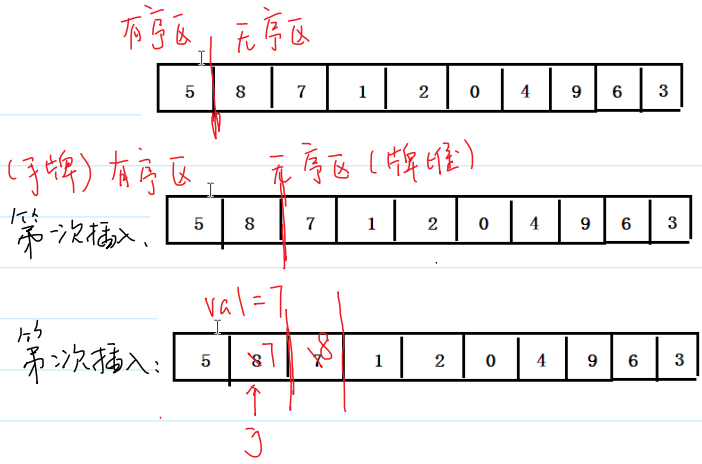
交换次数：$n-1$

空间复杂度：$O(1)$ 原地排序

稳定性：不稳定（发生长距离的交换）

插入排序

2022年3月28日 11:10




```c
void insertion_sort(int arr[], int n) {
    // i 代表无序区的第一个元素
    for (int i = 1; i < N; i++) {
        int val = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > val) {
            arr[j + 1] = arr[j]; → 交换
            j--;
        }
        // j == -1 || arr[j] <= val
        arr[j + 1] = val;
        print_arr(arr, n);
    }
}
```

选择Microsoft Visual Studio 调试控制台

```
5 8 7 1 2 0 4 9 6 3
5 7 8 1 2 0 4 9 6 3
1 5 7 8 2 0 4 9 6 3
1 2 5 7 8 0 4 9 6 3
0 1 2 5 7 8 4 9 6 3
0 1 2 4 5 7 8 9 6 3
0 1 2 4 5 7 8 9 6 3
0 1 2 4 5 6 7 8 9 3
0 1 2 3 4 5 6 7 8 9
```

时间复杂度:

最好情况: 原数组有序 $O(n)$

比较次数: $n-1$

交换次数: 0

最坏情况: 原数组逆序 $O(n^2)$

比较次数: $(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}$

交换次数: $(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}$

平均情况: $O(n^2)$

比较次数: 大于交换的次数, 小于 $\frac{n(n-1)}{2}$

交换次数: $\frac{n(n-1)}{4}$ (逆序度的个数)

插入排序: 当元素基本有序时, 性能非常好.

#2 空间复杂度: $O(1)$ 原地排序

#3 稳定性: 稳定

希尔排序

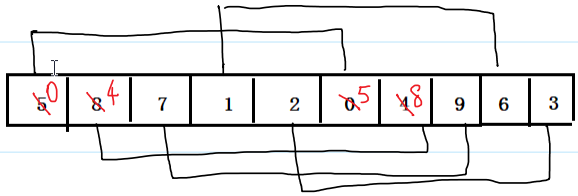
2022年3月28日 14:31

缩小增量排序（插入排序的改进版本）

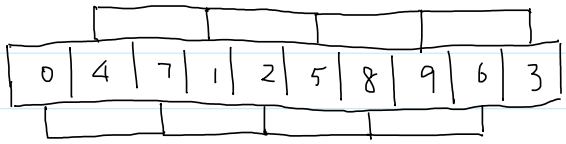
gap：$\frac{n}{2}$，$\frac{n}{4}$，…，1

组内使用简单的插入排序

gap = 5



gap = 2



gap = 1

```
void shell_sort(int arr[], int n) {
    int gap = n / 2;
    while (gap != 0) {
        // 组内插入排序
        for (int i = gap; i < n; i++) {
            int val = arr[i];
            int j = i - gap;
            while (j >= 0 && arr[j] > val) {
                arr[j + gap] = arr[j];
                j -= gap;
            }
            arr[j + gap] = val;
        }
        // 缩小增量
        gap /= 2;
        print_arr(arr, n);
    }
}
```

```
Microsoft Visual Studio 调试控制台
0 4 7 1 2 5 8 9 6 3
0 1 2 3 6 4 7 5 8 9
0 1 2 3 4 5 6 7 8 9
```

#1. 时间复杂度：

比 $O(n^2)$ 小，和具体的 gap 序列相关。

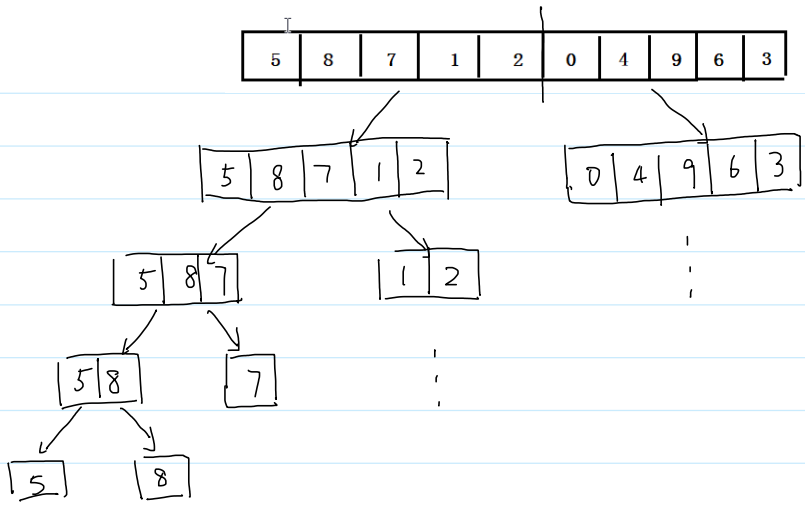
#2. 空间复杂度

$O(1)$ 原地排序

#3. 稳定性：不稳定

归并排序

2022年3月28日 15:05




```
void merge_sort1(int arr[], int left, int right) {
    // 边界条件
    if (left >= right) return;
    int mid = left + (right - left >> 1);
    // 对左半区间排序
    merge_sort1(arr, left, mid);
    // 对右半区间排序
    merge_sort1(arr, mid + 1, right);
    // 合并两个区间
    merge(arr, left, mid, right);   O(n)

    print_arr(arr, N);
}
```

时间复杂度：

$T(n) = 2T(\frac{n}{2}) + O(n)$

$a = 2,\ b = 2,\ d = 1$

$d = \log_b a$

$T(n) = n^d \log_b n = n \log_2 n$

Master Theorem: (拓展)

$T(n) = aT(\frac{n}{b}) + O(n^d)$

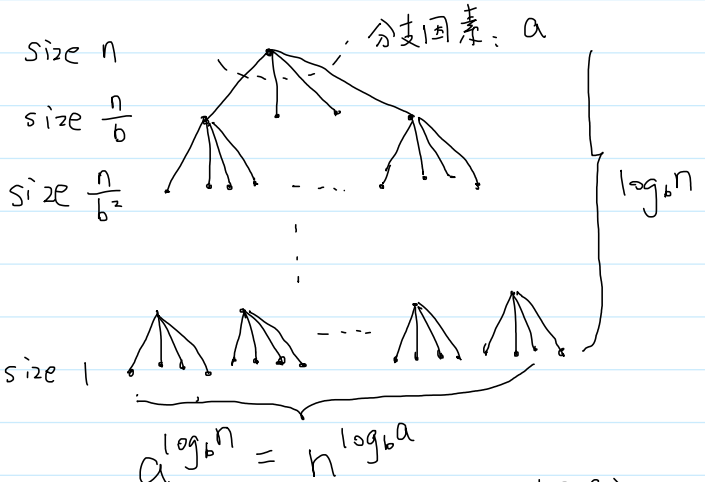
$a > 0,\ b > 1,\ d \geq 0$ 的常量.

计算第K层的工作量：

$a^K \times O(\frac{n}{b^K})^d = n^d \times O(\frac{a}{b^d})^K$

↓

几何级数



$a^{\log_b n} = n^{\log_b a}$

$\log_b(a^{\log_b n}) = \log_b n \cdot \log_b a = \log_b(n^{\log_b a})$

① $\frac{a}{b^d} < 1 \Leftrightarrow d > \log_b a$

$T(n) = O(n^d)$

② $\frac{a}{b^d} = 1 \Leftrightarrow d = \log_b a$

$T(n) = O(n^d \log_b n)$

③ $\frac{a}{b^d} > 1 \Leftrightarrow d < \log_b a$

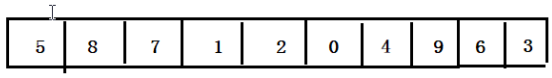
$T(n) = n^d \times O(\frac{a}{b^d})^{\log_b n} = n^d \times O(\frac{a^{\log_b n}}{n^d}) = O(a^{\log_b n}) = O(n^{\log_b a})$

空间复杂度：O(1)

稳定性：稳定

快速排序

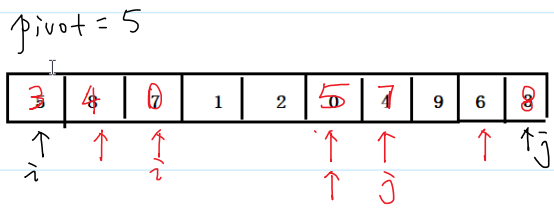
2022年3月28日 16:22



① 选取基准值 (pivot)

② 分区 (partition)

把所有比基准值小的元素移动到基准值的左边；把所有比基准值大的元素移动到基准值的右边.



③ 对左边区间进行快速排序
对右边区间进行快速排序

```
void quick_sort1(int arr[], int left, int right) {
    // 边界条件
    if (left >= right) return;
    // idx 为基准值的索引
    int idx = partition(arr, left, right);  → O(n)
    quick_sort1(arr, left, idx - 1);
    quick_sort1(arr, idx + 1, right);
}
```

时间复杂度：

最好情况：每次分区都分成相等的两份

$T(n) = 2T(\frac{n}{2}) + O(n) \Rightarrow T(n) = O(n\log n)$

最坏情况：每次基准都位于最左边或者最右边.

$T(n) = T(n-1) + n = n + (n-1) + \cdots + 1 + T(1) = O(n^2)$

平均情况：$T(n) = T(\frac{3}{4}n) + T(\frac{1}{4}n) + O(n)$ $T(n) = O(n\log n)$?

空间复杂度：$O(\lg n)$ → 递归的深度

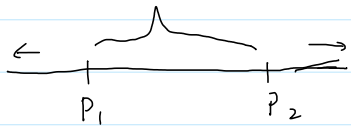
稳定性：不稳定 (长距离地交换两个元素)

改进策略：

① 基准值选取：随机选择，选择多个元素的中位数.

② 分区操作的优化

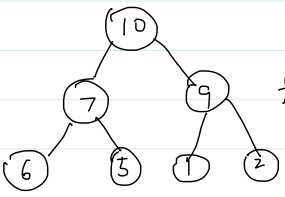
③ 选择多个基准值



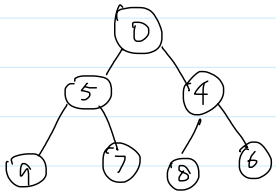
堆排

2022年3月28日 17:24

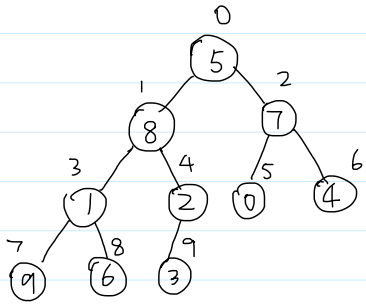
二叉堆：
- 大顶堆：根结点的键大于左右子树所有结点的键，并且左右子树都是大顶堆。
- 小顶堆 - - - - - 小于 - - - - - 小顶堆



大顶堆。



小顶堆。



5	8	7	1	2	0	4	9	6	3

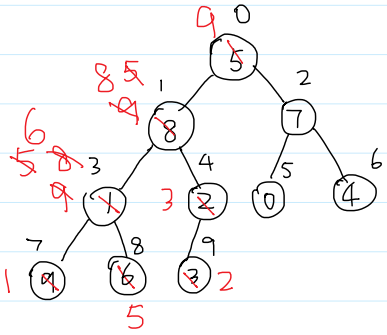
left(i) = 2i + 1

right(i) = 2i + 2

堆排算法：

① 构建大顶堆。

找到第一个非叶子结点，从后往前构建大顶堆



② 把堆顶元素和无序区的最后一个元素交换，无序区的长度 -1。

③ 把无序区重新调整成大顶堆。

④ 重复②③的操作，直到无序区的长度为 1