排序算法

1 冒泡排序

冒泡排序（英语：Bubble Sort）是一种简单的排序算法。它重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序（如从大到小、首字母从A到Z）错误就把他们交换过来。

```
void bubble_sort(int *a, int len) {
    int i, j, tmp;
    for(i = 0; i < len - 1; i++)
        for(j = 0; j < len - 1 -i; j++)
            if(a[j] > a[j+1]) {
                tmp = a[j];
                a[j] = a[j+1];
                a[j+1] = tmp;
            }
}
```

2 选择排序

选择排序（Selection sort）是一种简单直观的排序算法。它的工作原理如下。首先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置，然后，再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。

```
//交换两数
void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
void selection_sort(int *a, int len) {
    int i, j, tmp;
    for(i = 0; i < len; i++) {
        for(tmp = i, j = i + 1; j < len; j++)
            if(a[j] < a[tmp])
                tmp = j;
        swap(&a[i], &a[tmp]);
    }
```

```
}
```

3 插入排序

插入排序（英语：Insertion Sort）是一种简单直观的排序算法。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。

实际上是进行移位，tmp用于保存当前数的值，即a[i]。a[0] ~ a[j-1] 是已经排序好的部分。若a[j-1] > tmp，则将a[j-1]后移一位，直到a[j-1]小于tmp，即找到了tmp该插入的位置，令a[j]等于tmp。

```c
void insertion_sort(int *a, int len){
    int i, j, tmp;
    for(i = 1; i < len; i++){
        for(tmp = a[i], j = i; j > 0 && a[j-1] > tmp; j--)
            a[j] = a[j-1];
        a[j] = tmp;
    }
}
```

4 希尔排序

希尔排序，也称递减增量排序算法，是插入排序的一种更高效的改进版本。插入排序在对几乎已经排好序的数据操作时，效率高，即可以达到线性排序的效率，但插入排序一般来说是低效的，因为插入排序每次只能将数据移动一位。

```c
void shell_sort(int *a, int len){
    int i, j, gap, tmp;
    //len >> 1 相当于 len / 2
    for(gap = len >> 1; gap > 0; gap = gap >> 1)
        for(i = gap; i < len; i++){
            tmp = a[i];
            for(j = i - gap; j >= 0 && a[j] > tmp; j -= gap)
                a[j+gap] = a[j];
            a[j+gap] = tmp;
        }
}
```

5 归并排序

归并排序（mergesort）把数据分为两段，从两段中逐个选最小的元素移入新数据段的末尾。是典型的分治算法，以O(NlogN)最坏情形运行时间运行。

实现：取两个已排序的输入数组A和B，一个输出数组C，以及三个计数器ap、bp、cp，它们初始值为对应数组的开始端。A[ap]和B[bp]中的较小者被拷贝到C中的下一个位置，相关的计数器向前推进一步。当两个输入表有一个用完时，将另一个表中剩余部分拷贝到C中。

```c
void merge_sort_recursive(int *a, int *reg, int start, int end){
    if(start >= end) return;
    int len = end - start, mid = (len >> 1) + start;
    int start1 = start, end1 = mid;
    int start2 = mid + 1, end2 = end;
    merge_sort_recursive(a, reg, start1, end1);
    merge_sort_recursive(a, reg, start2, end2);
    int k = start;
    while(start1 <= end1 && start2 <= end2)
```

```
        reg[k++] = a[start1] < a[start2] ? a[start1++] : a[start2++];
    while(start1 <= end1)
        reg[k++] = a[start1++];
    while(start2 <= end2)
        reg[k++] = a[start2++];
    for(k = start; k <= end; k++)
        a[k] = reg[k];
}

void merge_sort(int a[], const int len){
    int reg[len];
    merge_sort_recursive(a, reg, 0, len - 1);
}
```

6 快速排序

快速排序（quicksort）是目前已知的最快的排序算法，它的平均运行时间是O(NlogN)。在区间中随机挑选一个元素作基准，将小于基准的元素放在基准之前，大于基准的元素放在基准之后，再分别对小数区与大数区进行排序。

6.1 基本步骤

- 取数组S中任一元素v，称之为枢纽元（pivot）。
- 将S中除v外的元素分成两个不相交的集合，小于v的元素全部放在S1中，大于v的元素全部放在S2中。
- 返回quicksort(S1)后，继而v，最后quicksort(S2)。

6.2 选取枢纽元

当输入是随机的，选取第一个元素或最后一个元素作为枢纽元是可以接受的，但是如果输入是预排序或反序的，那么这种选择就是劣质的分割。

比较好的方法是三数中值分割法，一般使用左端、右端和中心位置的三个元素的中值作为枢纽元。

6.3 分割策略

分割阶段要做的就是把所有小元素移到数组左边而把所有大元素移到数组右边。设有计数器i和j，当i在j的左边时，将i右移，移过小于枢纽元的元素，将j左移，移过大于枢纽元的元素。当i和j停止时，i指向大于枢纽元的元素，而j指向小于枢纽元的元素，若i在j的左边，则将两个元素互换。

```
void swap(int *x, int *y){
    int tmp = *x;
    *x = *y;
    *y = tmp;
}

void quick_sort(int a[], int start, int end){
    int i, j, t, baseval;
    if(start < end) return;
    baseval = a[start];
    i = start; j = end;
    while(i != j){
        while(a[j] >= baseval && i < j) j--;
        while(a[i] <= baseval && i < j) i++;
        if(i < j) swap(&a[i], &a[j]);
    }
    a[start] = a[i];
```

```
    a[i] = baseval;
    quick_sort(a, start, i-1);
    quick_sort(a, i+1, end);
}
```

7 总结

排序算法	平均时间	最好情况	最坏情况	空间复杂度
冒泡排序	$O(n^2)$	O(n)	$O(n^2)$	O(1)
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)
插入排序	$O(n^2)$	O(n)	$O(n^2)$	O(1)
希尔排序	O(n log n)	$O(n \log^2 n)$	$O(n \log^2 n)$	O(1)
归并排序	O(n log n)	O(n log n)	O(n log n)	O(n)
快速排序	O(n log n)	O(n log n)	$O(n^2)$	O(log n)