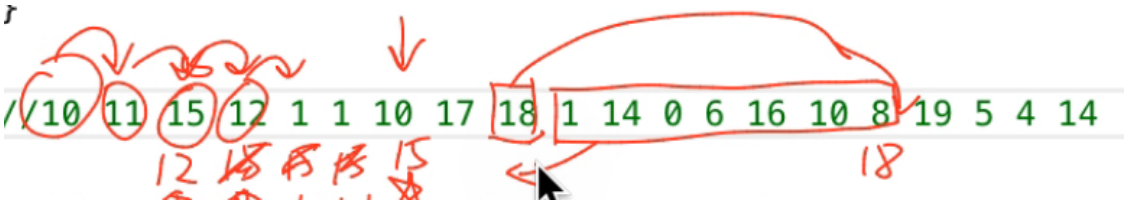
排序与算法性能——C笔记

一、冒泡搜索

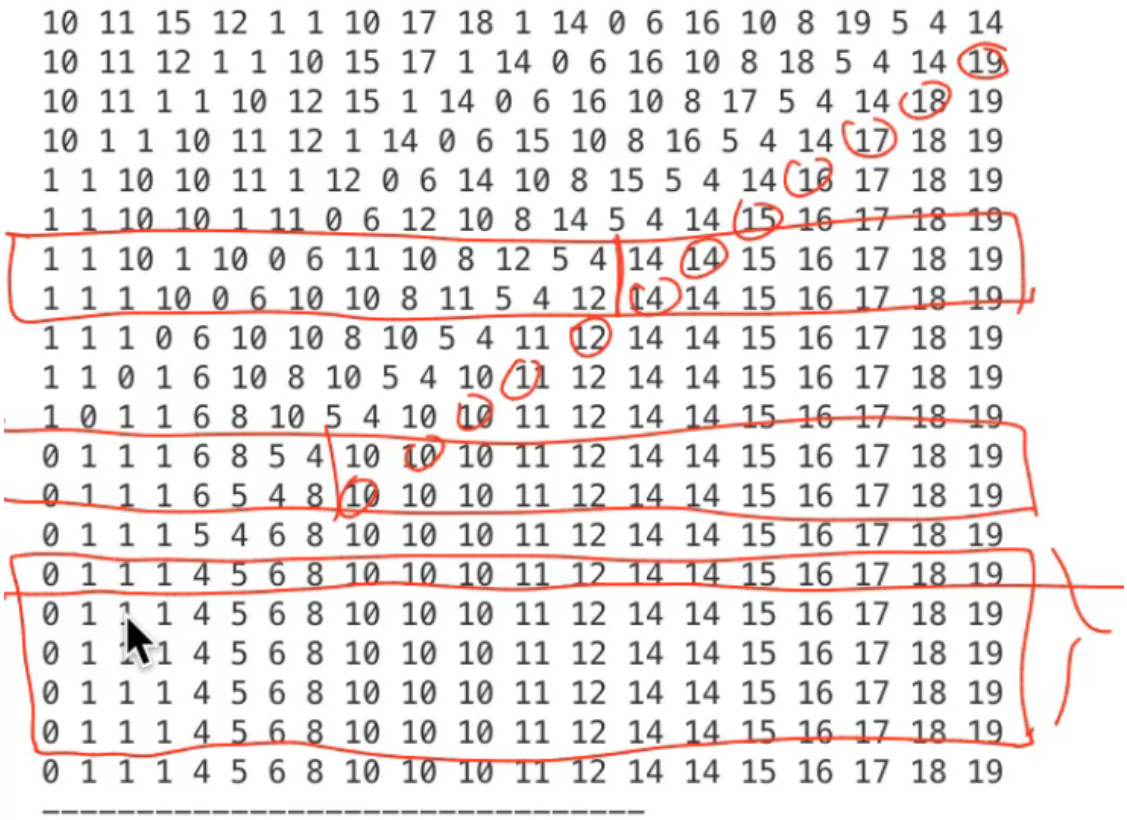
1. 第一轮18的移动相当于18移到8的后面，然后1 14 0 6 16 10 8这组整体往前挪一位，冒泡一轮会做很多次swap，swap性能问题，有人提出冒泡的变种：先比较再移动，缺点是**代码复杂，性能提升可能是微乎其微**，所以没必要，但这个问题本身还是很有价值的。



2. 冒泡排序代码的实现：

```
void sort(int a[], int begin, int end)
{
    if(end>begin){
        // bubble
        for(int i=begin; i<end; i++){
            if(a[i]>a[i+1]){
                // swap
                swap(a[i], a[i+1]);
            }
        }
        sort(a, begin, end-1);
    }
}
```

3. 冒泡过程中可能导致**重复循环**，冒泡数据是局部变动。



4. 代码更新后：

```
void sort(int a[], int begin, int end)
{
    int lastswap=-1;
    if(end>begin){
        // bubble
        for(int i=begin; i<end; i++){
            if(a[i]>a[i+1]){
                // swap
                lastswap=i;
                swap(a[i], a[i+1]);
            }
        }
        printf("%d: ", lastswap);
        sort(a, begin, end-1);
    }
}
```

没有发生两个数据都排好的情况时（冒泡一轮一般只排好一个），`end-1=lastswap`，得到数据如下：

```
10 11 15 12 1 1 10 17 18 1 14 0 6 16 10 8 19 5 4 14
18: 10 11 12 1 1 10 15 17 1 14 0 6 16 10 8 18 5 4 14 19
17: 10 11 1 1 10 12 15 1 14 0 6 16 10 8 17 5 4 14 18 19
16: 10 1 1 10 11 12 1 14 0 6 15 10 8 16 5 4 14 17 18 19
15: 1 1 10 10 11 1 12 0 6 14 10 8 15 5 4 14 16 17 18 19
14: 1 1 10 10 1 11 0 6 12 10 8 14 5 4 14 15 16 17 18 19
12: 1 1 10 1 10 0 6 11 10 8 12 5 4 14 14 15 16 17 18 19
11: 1 1 1 10 0 6 10 10 8 11 5 4 12 14 14 15 16 17 18 19
10: 1 1 1 0 6 10 10 8 10 5 4 11 12 14 14 15 16 17 18 19
9: 1 1 0 1 6 10 8 10 5 4 10 11 12 14 14 15 16 17 18 19
8: 1 0 1 1 6 8 10 5 4 10 10 11 12 14 14 15 16 17 18 19
7: 0 1 1 1 6 8 5 4 10 10 10 11 12 14 14 15 16 17 18 19
6: 0 1 1 1 6 5 4 8 10 10 10 11 12 14 14 15 16 17 18 19
5: 0 1 1 1 5 4 6 8 10 10 10 11 12 14 14 15 16 17 18 19
4: 0 1 1 1 4 5 6 8 10 10 10 11 12 14 14 15 16 17 18 19
-1: 0 1 1 1 4 5 6 8 10 10 10 11 12 14 14 15 16 17 18 19
-1: 0 1 1 1 4 5 6 8 10 10 10 11 12 14 14 15 16 17 18 19
-1: 0 1 1 1 4 5 6 8 10 10 10 11 12 14 14 15 16 17 18 19
```

5. 将代码中的递`sort(a, begin, end-1);`修改为`sort(a, begin, lastswap);`，可以做到加速排序。
6. 已经排好序的数据冒泡只需要跑一轮，逆序n^2轮，如果数据是基本已经排好序，冒泡排序比较适合。

二、插入排序

1. 把前面n-1个数据排好序，最后一个选择地放到排好序的n-1个数据里面。
2. 插入排序代码的实现（不是尾递归）：

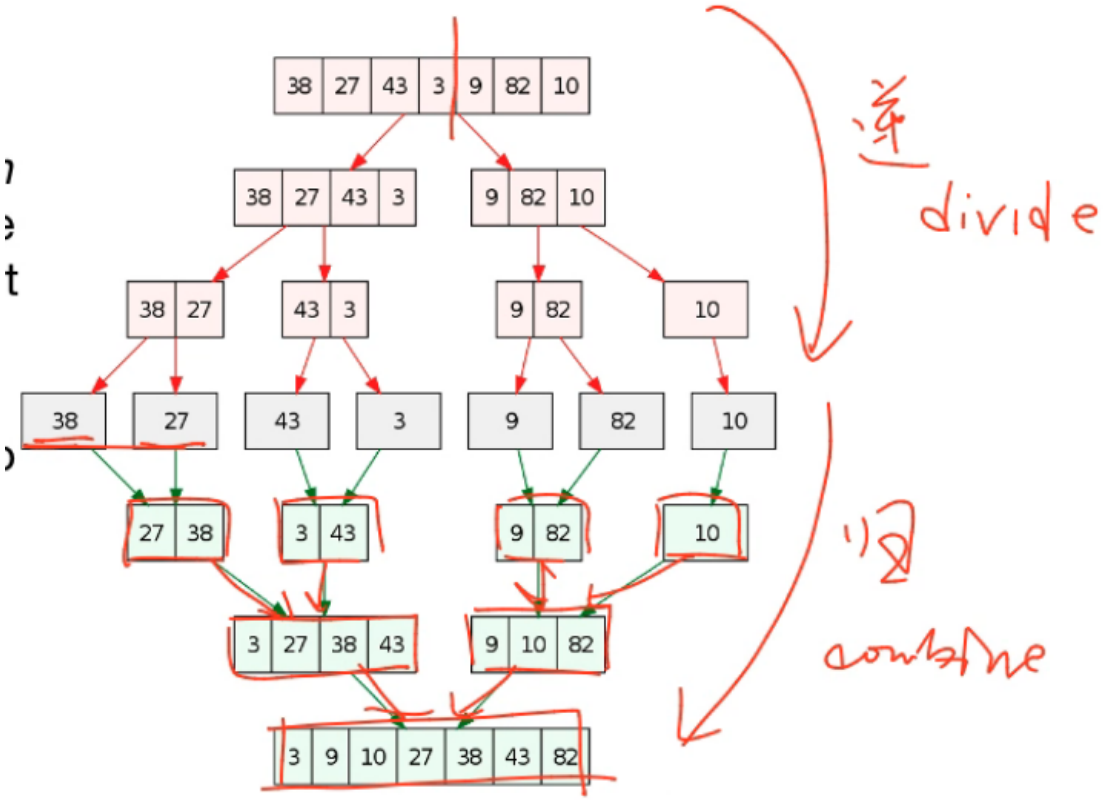
```
void sort(int a[], int begin, int end)
{
    if(end>begin){
        sort(a, begin, end-1);
        for(int i=end; i>0; i--){
            if(a[i]<a[i-1]){
                swap(a[i], a[i-1]);
            }else{
                break; //不加break的话性能就和冒泡一样了
            }
        }
    }
}
```

三、归并排序

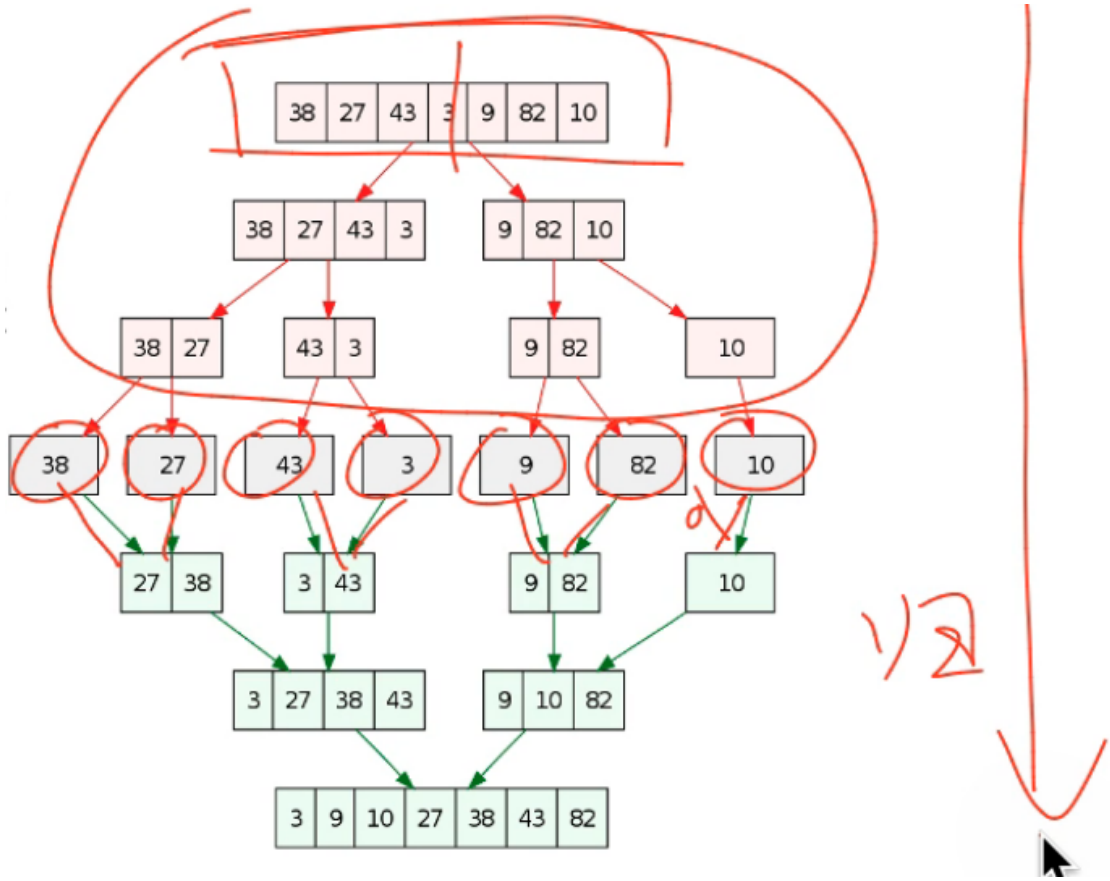
1. 将两个已经排好序的数组合并成一个排好序的大数组。
2. 合并两个排好序的数组的代码实现（类似之前写过的链表合并）：

```
void merge(int a1[], int len1, int a2[], int len2, int t[], int lent)
{
    int p1=0, p2=0, p=0;
    while(1){
        if(a1[p1]<a2[p2]/*省略后面两个循环加一个||p2>=len2来表示p2已经凉了*/){
            t[p]=a1[p1];
            p1++;
        }else if(a1[p1]>=a2[p2]/*||p1>=len1*/){
            t[p]=a2[p2];
            p2++;
        }
        p++;
        if(p1>=len1||p2>=len2){
            break;
        }
    }
    for(; p1<len1; p1++){
        t[p++]=a1[p1];
    }
    for(; p2<len2; p2++){
        t[p++]=a2[p2];
    } //这两个循环其实可以省略掉
}
```

3. 归并排序实现过程：



4. 分割成一个元素，数组就是由一个个元素组成，是不是可以不做递，效率更高，只做归：



四、快排

1. 归并排序`merge`又费时间又费空间。
2. 找一个`pivot`，比它小的放在左边，比它大的放在右边，左右两边排好序，整个数组就排好序了。

伪代码：

```
quicksort(A, begin, end)
    if begin<end then
        p:=partition(A, begin, end);
        quicksort(A, begin, p-1);
        quicksort(A, p+1, end);
```

比较麻烦的是`partition`，怎么把小的放左边，大的放右边。

3. `partition`主流算法：

   选最后一个作为pivot，i=begin，j遍历begin到end，如果aj<pivot，则交换ai和aj，最后交换ai和pivot，返回i。

4. pivot的选择，大致为中位数最好，左右匀称，无法做到最优秀的情况，有时候快排也是循环n^2，随机选取pivot is a good idea。