第八章 内排序

我保证没有抄袭别人作业

1. 设 $\{s_1,...,s_m\}$ 为字母表 {a,b,...,z} 上的 m 组字符串，并且记l_i为字符串s_i的长度。尝试通过修改基数排序，对这些字符串按字典序排序，并且使得算法的时间复杂度为 $O(\sum_{i=1}^{m} l_i)$（简述思路并且给出算法时间复杂度的分析）

解:

for(int idx = 0; idx <= max(l_i); idx += 1){

对字符串按照 s[idx]进行排序(基于静态链的桶排序);

收集,收集过程中如果遇到长度等于 idx 的(即 s_i[idx] = 0)的,跳过这个字符串,它不参与下一次排序

}

这种排序算法能够得到正确的排序结果,因为前缀相同的字符串中:长度短的字符串最后用结尾的 0 参与了一次排序,一定排在长度长的字符串的前面

每一次排序共有 sum((int)(l_i <= idx))个字符串进行了排序,时间复杂度为 θ(sum((int)(l_i <= idx)) + 26),对 idx 求和得时间复杂度为 θ(sum(l_i + 1) + 26(maxl_i + 1))(因为每一个字符串都用最后的'\0'参与了一次排序),

但 l_i + 1 < 27l_i(为了估计最后一项),所以有时间复杂度为 $O(\sum_{i=1}^{m} l_i)$.

2. 给定元素值互异的数组 A[1,...,n]，尝试在 O(n)时间内找到最小的 k 个数（提示：以快速排序算法为模型，递归查找第 k 小的数）。

解:

```
/* [start, end) */
int find_k(int start, int end){
    if(start + 1 >= end){
        return 1;/* 只有一个元素无需排序 */
    }
    int idx1 = start, idx2 = end - 1;
    int save = array[start];
    while(idx1 != idx2){/* 和快速排序的过程相同 */
        while(array[idx2] >= save && idx2 > idx1){
            idx2 -= 1;
        }
        if(idx2 > idx1){
            array[idx1] = arrray[idx2];
            idx1 += 1;
        }
```

```
        while(array[idx1] <= save && idx1 < idx2){
            idx1 += 1;
        }
        if(idx1 < idx2){
            array[idx2] = array[idx1];
            idx2 -= 1;
        }
    }
    array[idx1] = save;
    if(idx1 > k){
        return find_k(start, idx1);
    }
    if(idx1 < k - 1){
        return find_k(idx1 + 1, end);
    }
    /* idx1 前面的 k 个数是最小的,
    * 或者 idx1 和它前面的 k-1 个数一同组成最小的 k 个数
    */
    if(idx1 == k || idx1 == k - 1){
        return 1;
    }
}
```

3. 给定如下排序算法:

sort(A,i,j)
    If $A[i] > A[j]$ then
        Swap(A[i],A[j])
    If $(j - i + 1) \geq 3$ then
        $t = (j - i + 1)/3$
        $sort(A,i,j - t)$
        $sort(A,i + t,j)$
        $sort(A,i,j - t)$
    Return A

尝试给出算法的正确性证明，并且求出其时间复杂度。

证明:(归纳)
对数组的元素个数进行归纳, 当数组中有 1,2 个元素时,都能得到正确的排序结果,现假设数组中元素个数小于等于 k 时,都能得到正确的排序结果,则当元素个数为 k+1 时

枚举 a[k](第 k+1 个元素)在最终正确的排序结果中可能出现的位置
1 a[k]在排序结果的前三分之一:则在第二次 sort 中,它会排到整个数组的中间(a[i+t]到 a[j-t]),从而在第三个 sort 中参与排序,最终的排序序列中也会出现在正确的位置,同时也需说明数组的其他元素在 a[k]加入后仍能排到正确的位置:类似对 a[k]的讨论,如果 a[i]应在排序结果的前三分之二,在经过前两轮 sort 后它会出现在数组的前三分之二,最后一次排序中由归纳假设,它会出现在合适的位置,如果应在后三分之一,则在第一次 sort 后,会出现在中间的一段,在第二次排序之后会出现在正确的位置(归纳假设)
2 a[k]在排序结果的中三分之一:类似之前的讨论,它在第二次 sort 中,它会排到整个数组的中间(a[i+t]到 a[j-t]),从而在第三个 sort 中参与排序,最终的排序序列中也会出现在正确的位置,数组中其他元素的位置也类似上面的讨论,由归纳假设得算法能得到正确的结果
3 a[k]在排序结果的后三分之一:在第二次排序中,a[k]直接排到正确的位置(第一次 sort 已经将前三分之二段中所有可能在后三分之一的元素送到中三分之一,第二次 sort 将后三分之一直接排好),对数组中其他元素的讨论仍类似上面的讨论

时间复杂度: $T(n) = 3 * T(2n / 3) + c$
$T(n) / n = 2 * (T(2n / 3) / (2n / 3)) + c / n$
将上式中的 n 用 2n / 3 带入,不断迭代直到 n = 1, 共迭代$[\log_{3/2}n]$次
则有 $t(n) / n = (c / n)(1 + 2 * 3 / 2 + 4 * (3 / 2)^2 + 3^{([\log_{3/2}n])}) = (c / n) * O(n^{(\log_{3/2}3)})$
所以时间复杂度为 $O(n^{(\log_{1.5}3)})$

4. 假设数组 A[1,…,n]中的各元素独立同分布，给定非严格单调递增函数 H，将数组 A[1,…,n]的元素等概率地映射为 0 到 m-1（$m \leq n$）的 m 个整数。这等价于将数组 A 划分为 m 个桶，并且 A[i]落入各个桶的概率相等，即 1/m。然后分别对各个桶进行插入排序，最后按顺序扫描各个桶，将数组 A[1,…,n]递增输出。假设映射操作的时间复杂度均为 O(1)，1）证明算法的正确性 2）这个算法的最好、最坏和平均时间复杂度是多少？

正确性:只需证明对任意的 m<n 排序后 m 都在 n 的前面,分情况讨论:
1 如果 h(m) < h(n)则在分桶的时候 m 就在 n 的前面,收集之后也在它的前面
2 如果 h(m) = h(n)则 mn 分到同一个桶,在插入排序后 m 也在 n 的前面
综上,在收集之后 m 总会在 n 的前面,所以排序算法是正确的
时间复杂度
设每一个桶中有 mi 个元素,则总的时间复杂度为 $O(sum(mi^2) + n)$其中 sum(mi) = n
最好时间复杂度为 mi 都相同的时候(均值不等式),时间复杂度为 $O(m*(n/m)^2 + n) = O(n^2 / m)$ (进行映射和扫描,提供 O(n)的复杂度)
最坏的时间复杂度为存在一个 mi = n 的时候,时间复杂度为 $O(n^2 + n) = O(n^2)$
平均的时间复杂度为 $O(n + (nm + n^2 - n)/m) = O(n^2 / m)$
$E(T(n)) = O(n + sum(E(mi^2))) = O(n^2 / m)$
/* 证明过程在最后一页和附件里的图片…
* 抱歉助教…排版没有安排好位置…
*/

5. 参照教材给出的两种索引排序方法，完成如下表格（增序）：
其中索引 1 的下标存放的是数组 A 中数据在排序后应该处于的位置，索引 2 的下标存放的是数组 A 中应该摆放的数据的位置。

原下标	0	1	2	3	4	5	6	7
数组 A	20	13	11	11’	19	89	6	4
索引 1 的下标	6	4	2	3	5	7	1	0
索引 2 的下标	7	6	2	3	1	4	0	5
结果	4	6	11	11’	13	19	20	89

6. 给定数组 A[1,...,n]，在空间复杂度 O(1)的条件下实现归并排序（时间复杂度尽可能小）。给出算法流程，并分析时间复杂度。

解:

```
/* 每一次找到第二组元素应在的位置,插入排序 */
void merge(int s1, int e1, int s2, int e2){
    int idx1 = s1;
    int idx2 = s2;
    while(idx2 < e2 && idx1 != idx2){
        if(a[idx1] < a[idx2]){
            idx1 += 1;
        }else{
            int tmp = a[idx2];
            for(int i = idx2; i > idx1; i -= 1){
                a[i] = a[i - 1];
            }
            a[idx1] = tmp;
            idx1 += 1;
            idx2 += 1;
        }
    }
}

void mergesort(int start, int end){
    if(start + 1 >= end){
        return ;
    }else{
        mergesort(start, (start + end) / 2);
        mergesort((start + end) / 2, end);
        merge(start, (start + end) / 2, (start + end) / 2, end);
    }
}
```

时间复杂度:

$T(n) = 2 * T(n/2) + (n/2)^2$

$= 4 * T(n/4) + (n/2)^2 + (n/4)^2 * 2$

...

$= O(n^2)$ (1/4+1/8+... < 1, 且最底层的元素(T(1))个数也是可以被 n^2 控制的)

时间复杂度为 $O(n^2)$

第四题平均情况下的过程