# 算法第五次作业

**10205101530-**赵晗瑜

## 9.1-2

Prove the lower bound of $\lceil 3n/2 \rceil - 2$ comparisons in the worst case to find both the maximum and minimum of $n$ numbers. (*Hint:* Consider how many numbers are potentially either the maximum or minimum, and investigate how a comparison affects these counts.)

[译]

在最坏情况下证明 $\lceil 3n/2 \rceil$-2 比较的下限，以找到 n 个数字的最大值和最小值。（提示：考虑有多少数字可能是最大值或最小值，并研究比较如何影响这些计数 .)

[ans]

1. 最初，所有 $n$个数字都可能是最大值或最小值。
2. 令$MAX$表示潜在的最大值构成的集合，$MIN$表示潜在的最小值构成的集合。
3. 当我们在原数组中比较两个元素a和b时，不妨设$a <= b$，我们把a从$MAX$中剔除，把b从$MIN$中剔除，从而a和b均不可能同时出现在$MIN$和$MAX$中。
    - 当n为偶数时，每两个数为一组进行一次比较，一共 $n/2$ 次比较．比较完成后，所有元素均参与一次比较，由上文的分析可知$MIN$和$MAX$不相交，且 $|MAX| = |MIN| = n/2$
    - 当 n为奇数时，先不考虑最后一个元素，按照偶数的情况比较．对于最后一个元素，与倒数第二个元素进行比较．一共进行了 $(n-1)/2 + 1 = \lceil n/2 \rceil$ 次比较，且 $|MAX|$和 $|MIN|$中其一为 $\lceil n/2 \rceil$，另一个为$\lfloor n/2 \rfloor$
4. 综上，经过$\lceil n/2 \rceil$次比较后$MIN$和$MAX$是不相交的，并且 $|MIN|$和 $|MAX|$ 中其一为 $\lceil n/2 \rceil$，另一个为$\lfloor n/2 \rfloor$
5. 对$MIN$做线性比较，一共比较$|m| - 1$次可以得到最小值，$MAX$同理。综上，总比较次数为

$$\lceil n/2 \rceil + \lfloor n/2 \rfloor - 1 + \lceil n/2 \rceil - 1 = \lceil n/2 \rceil + n - 2 = \lceil 3n/2 \rceil - 2$$

## 9.3-1

In the algorithm SELECT, the input elements are divided into groups of 5. Will the algorithm work in linear time if they are divided into groups of 7? Argue that SELECT does not run in linear time if groups of 3 are used.

[译]

在算法 SELECT 中，输入元素被分成 5 个一组。如果将它们分成 7 个一组，算法会在线性时间内工作吗？ 如果使用 **3** 组，则认为 SELECT 不会在线性时间内运行。

[ans]

Assume the input elements are divided into groups of k. Similar like the analysis in the book, at least half of the $\lceil n/k \rceil$ groups contribute at least $\lceil n/2 \rceil$ elements that are greater than x, except for the one group that has fewer than k elements if k does not divide n exactly, and the one group containing x itself. Discounting these two groups, it follows that the number of elements greater than x is at least

$$\lceil k/2 \rceil (\lceil \frac{1}{2} \lceil n/k \rceil \rceil - 2) \geq \frac{k}{2} (\frac{1}{2} \frac{n}{k} - 2) = \frac{n}{4} - k$$

Similarly, at least $\frac{n}{4} - k$ elements are less than x. Thus, in the worst case, step 5 calls SELECT recursively on at most $\frac{3n}{4} + k$ elements.

So when n is greater than some constant, we have

$$T(n) \leq T(\lceil \frac{n}{k} \rceil) + T(\frac{3n}{4} + k) + O(n)$$

Assume T(n) runs in linear time, substituting it into the recurrence yields:

$$\begin{aligned} T(n) &\leq T(\lceil \frac{n}{k} \rceil) + T(\frac{3n}{4} + k) + O(n) \\ &\leq T(\lceil \frac{n}{k} \rceil) + T(\frac{3n}{4} + k) + an \\ &\leq c\frac{n}{k} + c(\frac{3n}{4} + k) + an \\ &= cn + (\frac{c}{k}n + an - \frac{c}{4}n + ck) \\ &\leq cn \end{aligned}$$

where the last step holds as long as

$$\frac{c}{k}n + an - \frac{c}{4}n + ck \leq 0$$

So we need to find some k such that there exists constants c and a such that

$$\frac{c}{k}n + an - \frac{c}{4}n + ck \leq 0$$

We have

$$\frac{c}{k}n + an - \frac{c}{4}n + ck = c(\frac{n}{k} - \frac{n}{4} + k) + an \leq 0$$

Because both c and a are positive, so it could only be

$$\frac{n}{k} - \frac{n}{4} + k \leq 0$$

Let $f(k) = \frac{n}{k} - \frac{n}{4} + k$ , so

$$f(4) = 4 > 0, f(5) = -\frac{n}{20} + 5 \leq 0 \text{ when } n \geq 100$$

So we can always find a $n_0$ such that $f(k) \leq 0$ when $k \geq 5$

Thus the algorithm work in linear time if the input elements are divided into groups of 7, but doesn't run in linear time if they are divided into groups of 3. When k=4, the algorithm works in $\Omega(nlgn)$。

## 9.3-3

Show how quicksort can be made to run in $O(n \lg n)$ time in the worst case, assuming that all elements are distinct.

[译]

展示在最坏的情况下如何在 $O(nlgn)$时间内运行快速排序，假设所有元素都是不同的。

[ans]

为了在最坏情况下也保证对$O(nlgn)$的复杂度，需要改变对$pivot$的选择。可以每次都选择序列的 中位数作为$pivot$，对中位数的选择只需要$O(n)$，因此不需要额外的复杂度。

## 15.1-3

Consider a modification of the rod-cutting problem in which, in addition to a price $p_i$ for each rod, each cut incurs a fixed cost of $c$. The revenue associated with a solution is now the sum of the prices of the pieces minus the costs of making the cuts. Give a dynamic-programming algorithm to solve this modified problem.

[译]

我们对钢条切割问题进行一点修改，除了切割下的钢条段具有不同的价格$p_i$外，每次切割还要付出固定的成本$c$。这样，切割方案的收益就等于短钢条的价格之和减去切割成本。设计一个动态规划算法解决修改后的钢条切割问题。

[ans]

- 动态规划的第一个基本特点：所求解的问题满足最优子结构，问题可以分解为规模更小的子问题，问题的最优解依赖于子问题的最优解，并且这些子问题可以独立求解。
- 动态规划的第二个基本特点：相同的子问题只需要求解一次，如果子问题的解会被多次引用，可以将子问题的解保存起来。

该问题仍然满足最优子结构。只需对算法稍加修改，即在计算每种方案的收益时减去切割成本。因此，最优收益公式变成了：

$$r_n = max(p_n, \max_{1<=i<=n-1}(p_i + r_{n-1} - c))$$

需要注意的是，不切割的方案不会有切割成本。

下面是伪代码:

```
//参数n：钢条长度
//参数p：价格表p[1..n]
//参数c：切割成本
//返回值：长度n的钢条的最大收益
//自上而下的求解方式：任何子问题的解都只依赖于规模更小的子问题。我们可以将子问题按照规模由小到大的顺序进行求解。当求解某个子问题时，它所依赖的那些规模更小的子问题都已求解完毕，并且结果已经保存。每个子问题也只需要求解一次。
BOTTOM-TO-UP-CUT-ROD(p,c,n)
    create a new array r[0..n]
    r[0]=0
    for i=1 to n
        q=p[i]
        for j=i-1 to 1
            q=max(q,p[j]+r[i-j]-c)
        r[i]=q
    return r[n]
```