# 算法习题课

陈铭

2024 年 11 月 9 日

## 1 PS1

### 1.1 P1

证明的核心在于两个循环不变量。
For 的是每次循环开始前，前面的都有序。
while 的是每次开始前，key 的左右两个子数组都有序且右子数组的每个元素都比 key 大。
给分比较难给，因为有的同学不显式地写出循环不变量但是在证明过程中蕴含了这部分内容，那我也都给分了。

### 1.2 P2

伪代码没啥好讲的，就是那个大就用大的去 mod 小的。
可终止性挺简单的，就是不可能一直 mod 下去，为什么正确就是一条式子：
$GCD(x, y) = GCD(x \bmod y, y) = ... = d$

### 1.3 Bonus

第一问只需要用异或，0 然后挨个异或，利用 1 和 1 异或是 0。
第二问只需注意到是模 3 的操作，通过移位操作可以找到每个数的第 i 个二进制的位，然后相加模 3 然后复原即可。

# 2 PS2

## 2.1 P3

第一问就是简单的 mergesort，空间复杂度 O(n) 在于可以复用数组。
第二问就是一个链表形式的 mergesort，降低空间复杂度的核心在于链表在 merge 操作上是可以实现 inplace 的更新的，具体做法可以就新建一个指针 p，然后你手头肯定有一开始的左右指针，然后就看哪个小就往 p 后面加，这样类似的做即可。

## 2.2 P4

第一问呢，没啥好讲的，拆成下面这样就完事了：

$$
\begin{aligned}
&n = x \cdot 2^{\frac{n}{2}} + y \\
&n^2 = x^2 \cdot 2^n + xy \cdot 2^{\frac{n}{2}+1} + y^2 \\
&= x^2 \cdot 2^n + y^2 + 2^{\frac{n}{2}} \left(x^2 + y^2 - (x-y)^2\right) \\
&= x^2 \left(2^n + 2^{\frac{n}{2}}\right) + y^2 \left(1 + 2^{\frac{n}{2}}\right) - 2^{\frac{n}{2}} (x-y)^2
\end{aligned}
$$

第二问很多同学都没拿到分，注意到题目中问的是平方在渐进意义上的速度快于乘法，他说的是可能存在一种算法使得比乘法快。那么你就不能从正面证明，因为你不可能穷举所有的算法，正确的做法是假设存在一种平方的做法比乘法快，你又知道：

$$xy = \frac{x^2 + y^2 - (x-y)^2}{2}$$

那么如果存在更快的平方的算法，那么肯定存在一种乘法的算法在渐进意义下只会差不多 3 倍，也就是同阶。

## 2.3 P6

第一问类似 mergesort 地二分去做，每次分解成找两个子数组的 majority element，然后最后再看两部分 majority element 的哪个是整体的 majority element。
这个的可终止性是显然的，正确性你只需注意到整个数组的 majority element 一定是数组对半分后某个数组的 majority element。
Bonus：

O(n) 复杂度不难想到要用遍历数组的方式，而且不难发现如果给你一个数字，你是可以在 O(n) 时间内验证这个数字是否是 majority element 的。
问题就变成了能否在一次遍历的时候就找到可能是 majority element 的元素，注意到 majority element 一定出现大于 n/2 次，所以如果把其他数字看成-1 的作用，这个数字看成 +1 的作用，那么不管什么时候遍历一遍后都会是正的。
所以只需要在遍历前维护两个量，一个是候选元素，一个是当前候选元素与不是候选元素个数的差值，一旦变成 0，就把下一个作为新的候选元素。最后再遍历一遍检查即可。

# 3 PS3

## 3.1 P2

第一问非常简单，就看比上面大还是比下面大，然后交换上去和交换下去就行，因为深度是 $\lg n$ 所以时间复杂度是 $O(\lg n)$
第二问有不少同学搞错，一个典型的错误是认为第 k 大的元素一定在前 $\log k$ 层，还有一个错误是认为可以先用 k 个建另一个堆，然后再挨个插入，前者算法本身是错的，后者时间复杂度不对。
正确的做法是先把 root 加到一个堆里面，然后每次 pop 根节点，pop 完将原堆的两个子节点加入到新堆中维护，由原堆的最大性，这样可以保证新堆里在第 i 次 pop 完永远有第 i+1 大的元素。

## 3.2 Bonus2

我看到最好的做法是朱士杭同学的做法，不少同学认为每次维护最大性时的时间开销一定是 $\lg n$，这是显然不对的，这道题就是要证在均摊意义下是 $\lg n$。

## 4 Bonus Problem

In the best case, namely the accordingly array of the heap has been sorted increasingly(assuming a max-heap).

Now doing the HEAPSORT, the minimum element in the root is swapped with the last element. Then do the HEAPUPDATE operation in Problem 2, which needs $O(\log n)$ time. For all nodes they would repeat this. Although after some swap and update, the layer would be shallower, but in the following I would prove that it doesn't matter.

6

Now let's just consider the deeper $\frac{\log n}{2}$ layers, there would be $n - 2^{\frac{\log n}{2}} = n - \sqrt{n}$ nodes need updating and at least swap $\frac{\log n}{2}$ times. So the whole best-case running time of HEAPSORT:

$$Total\ Cost \geq (n - \sqrt{n}) * \frac{\log n}{2} \geq \frac{n}{2} * \frac{\log n}{2} = \frac{n \log n}{4}$$

$\therefore Total\ Cost \in \Omega(n \log n)$, the conclusion is proved that the best-case running time of HEAPSORT is $\Omega(n \log n)$

图 1: 优秀答案

核心思想就是，如果 $n >> \sqrt{n}$ 的话，那么剩下的 $n - \sqrt{n}$ 个元素，里面至少有很大一部分要交换 $\frac{\log n}{2}$ 次。

### 3.3 P5

第一问注意问的是要你设计 $O(n)$ 的 flip 的算法，所以不用考虑别的操作的开销，那么其实只需要每次找到最大，然后翻两次即可翻到底下，这样最多只需要 $2n$ 次，所以就是 $O(n)$。
第二问注意到要证明这个是下界，需要构造极端情况，比如最大和最小，第二大和第二小在一起然后依次下来，按第一问的算法就需要至少 n 次 flip，也就是说 $\Omega(n)$ 是下界。

### 3.4 P6

第一问写出递推式子，用主方法不难得到复杂度是 $O(n^{\log_{\frac{3}{2}} 3})$。
第二问证明不难，只需注意到每次排序，都能把数组中大的一半移到后面，然后第六行排完，后面的 1/3 就是正确的位置，最后再排前面 2/3 的即可。
第三问的反例很好举，比如 [4,3,2,1]
第四问只要注意到，算法只会交换逆序对即可列出式子，不难发现就是 n(n-1)/2