浅谈数据结构题的几个非经典解法

南京外国语学校 许昊然

摘 要

数据结构题是信息学竞赛中十分常见的题目，以线段树、平衡树、分块均衡、“莫队算法”为代表的经典解决方法也早耳熟能详、路人皆知。本文将介绍数据结构题的一些普及度较低，但十分有用的非经典做法。本文的主要介绍内容包括对时间分治算法及其扩展、二进制分组算法、整体二分算法，其中包含作者的很多原创内容和经验总结，希望能起到抛砖引玉的作用。

1 二进制分组算法、对时间分治算法及其扩展

1.1 从动态半平面交谈起

动态半平面交是十分经典的数据结构问题。有一个平面，我们要完成一系列操作，包括：

1. 插入一个半平面
2. 删除某次插入的半平面
3. 查询一个点是否在当前半平面的交集内

这个问题有一个经典但代码极为复杂的算法：利用线段树套可持久化的平衡树进行维护，通过可持久化平衡树的切割与合并来保证时间复杂度，合并时的分类讨论细节极多。

如果把动态半平面交问题比作一个大怪物，那么上述的做法就一定是一根大铁棍了，虽然确实能一棍子把怪物拍死，但未免过于蛮力了。实际上，通过对时间分治算法的扩展，我们可以巧妙地解决这个问题，如同一把锋利的小刀，虽然看似无力，却能直刺怪物的心脏。我们将在下文中逐步解决这个问题。

1.2 “修改独立,允许离线”的数据结构题

我们首先约定，数据结构题中，称要求我们作出回答的操作为“询问”，称会对询问操作的答案造成影响的操作为“修改”。

我们发现，有不少数据结构题都满足以下两个条件。此时，对时间分治算法将是非常有用的利器。

1. 满足修改操作对询问的贡献独立，修改操作之间互不影响效果。

2. 题目允许使用离线算法。

我们应当如何利用这两个条件呢? 我们不妨假设我们要完成的操作序列是S. 那么我们将整个操作序列S等分为两份。

我们可以发现以下两个性质：

1. 显然，后一半操作序列中的修改操作对前一半操作序列中的查询的结果不会产生任何影响。

2. 后一半操作序列中的查询操作只受2方面影响：一是前一半操作序列中的所有修改操作；二是后一半操作序列中，在该询问之前的修改操作。

容易发现，因为后一半操作序列的修改操作完全不会影响前一半操作序列中的查询结果，因此前一半操作序列的查询实际是与后一半操作序列完全独立的，是与原问题完全相同的独立子问题，可以递归处理。

接下来，我们考虑后一半操作序列中的查询。我们发现，影响后一半操作序列答案的因素中，第二部分“后一半操作序列中，在该询问之前的修改操作”也是与前一半序列完全无关的（为什么完全无关? 别忘了我们的前提，修改操作互相独立互不影响，前一半序列中的修改操作不会对后一半序列中的修改操作的效果产生影响!）。因此，这部分因素也是与原问题完全相同的完全独立的子问题，可以递归处理。

我们发现，影响后一半操作序列答案的因素的第一部分“前一半操作序列中的所有修改操作”虽然与前一半序列密切相关，但有一个非常好的性质，就是：影响后半部分任意一个查询答案的东西，始终是一样的! 也就是前半部分操作中所有的修改操作。这时，我们发现原问题的动态修改操作不再存在，而被转化为了离线的、与原问题同样规模的“一开始给出所有的修改”然后“回答若干询问”的更简单问题，从而简化算法。

我们分析一下时间复杂度。不妨设“解决无动态修改操作的原问题”的复

杂度为$O(f(n))$，那么由主定理易得，我们对时间分治的总复杂度将是[31]

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right)+O(f(n)) \\ \text{解得}\quad T(n) &\le O(f(n)\log n) \end{aligned}$$

因此，只要数据结构题满足我们上文假定的两个要求：修改独立，允许离线，我们就可以以一个log的代价，把原问题中的动态修改给去掉，变为没有动态修改的简化版问题，从而使问题得到极大简化。

【例1】共点圆[32]

有一个平面，我们需要支持以下操作（为简化问题，保证所有点都在x轴上方，且横坐标非0）：

1. 给定一点，插入一个圆心位于该点且过原点的圆
2. 查询某点是否在所有圆的内部（含圆周）

本题的官方做法是对圆进行反演后，转化为凸包维护问题，使用splay树动态维护全凸壳，要求支持插入新点和查询点是否在凸包内部的操作，代码较为复杂。官方做法的时间复杂度是$O(n\log n)$。

其实，本题有更简单的做法。我们不妨首先进行一些简单的转化。

“一个点(x_0,y_0)处于圆心(x,y)且过原点的圆的内部”等价于

$$\begin{aligned} (x_0-x)^2+(y_0-y)^2 &\le x^2+y^2 \\ \text{即}\quad 2x_0\cdot x+2y_0\cdot y &\ge x_0^2+y_0^2 \end{aligned}$$

也就是说，符合条件的圆心(x,y)必须位于半平面$2x_0\cdot x+2y_0\cdot y\ge x_0^2+y_0^2$内。于是问题等价于，我们要求支持：

1. 插入一个新点
2. 给定一个半平面，查询是否所有的点都位于这个半平面内

容易发现，要支持的操作满足“修改独立，允许离线”这两个性质（每个插入的点都具有对询问答案的一票否决权，不受其他任何因素干预）。因此，我们直接套用上述上文的对时间分治算法，题目被简化为：

[31]注：事实上主定理要求$O(f(n))\ge O(n)$时该证明才成立。但因为序列的长度已经达到了n，单次处理复杂度几乎不可能小于$O(n)$，如果真的不满足可以具体问题具体分析

[32]题目来源http://tsinsen.com/A1381

开始时给定一堆点，要求回答一系列询问：

给定一个半平面，查询是否所有的点都位于这个半平面内

这个问题做法非常简单，我们不妨考虑求出“在与半平面垂直的方向上的投影最小的点”。如果这个点也在半平面内，那么必然所有点都在半平面内。进而容易发现，只有在凸包上的点才可能成为我们想要求的那个点，因此我们对开始的那堆点求一个凸包。进一步观察发现，我们所求的那个点相邻的两条边的斜率必然正好“夹住”了半平面的斜率，因此我们对凸包的边的斜率进行排序后，二分查找半平面的斜率即可找到我们所求的点，而简单的在$O(n\log n)-O(\log n)$时间内判定。

时间复杂度是$O(n\log^2 n)$，代码非常简单。尽管时间复杂度比官方解法的复杂度多了一个log。

但对时间分治真的不能做到和官方题解一样的复杂度吗？答案是，可以！我们发现，算法的瓶颈在于求凸包、斜率排序和二分查找，这些步骤似乎都难以优化。但我们漏掉了一个重要的性质：我们这是分治算法！我们解决了前一半序列和后一半序列的两个子问题后，已经得到了前一半序列的凸包和后一半序列的凸包，为何不直接把它们线性合并呢？对斜率的排序也可以做类似归并处理。而查询怎么办呢？没关系，我们把查询也按斜率归并排序，在凸包斜率和查询斜率都有序的情况下，只需$O(n)$扫一遍就可以得到我们想求的东西了。于是我们经过诸多优化，终于做到了处理复杂度$O(n)$，乘上分治带来的log后，得到了与官方题解一样的复杂度$O(n\log n)$。当然，这个做法并没有太多的现实意义，因为实际由于常数原因，$O(n\log n)$做法并不比$O(n\log^2 n)$的算法快多少，而代码量就大很多了。

【例2】经典问题

这是我们最初讨论的问题的简化版本。给定一个平面，要求支持：

1. 插入一个半平面
2. 查询一个点是否在当前半平面的交集内

这道题是动态半平面交问题去掉删除操作后的版本。容易发现，这道题也满足“修改独立，允许离线”的性质。运用对时间分治，我们可以把原问题简化为：

开始时给定若干半平面，要求回答一系列询问：

给定一个点，查询该点是否在当前半平面的交集内

这个问题想必大家都会做，直接用半平面交算法求出所有半平面的交集，然后任意找其内部一点，半平面交的顶点关于该点排极角序。查询时，只需二分查找找到查询点所位于的极角区间即可。这个算法复杂度是$O(n\log n)$，因此算上分治复杂度后，解决原问题复杂度是$O(n\log^2 n)$。

相比直接使用平衡树维护半平面交的顶点序，这个方法代码量非常小。而且，也可以使用与上一题中类似的方法优化复杂度至$O(n\log n)$，与平衡树维护法的复杂度相同。

1.3 “修改独立,要求在线”的数据结构题

但有一些数据结构题，虽然满足“修改独立”的性质，但出题人采用各种方法（操作加密、交互题等）强迫选手采用在线算法，此时，对时间分治就无能为力了。但我们依然有一个好方法应对这类问题：二进制分组算法。

我们将一个数拆成2的幂次从大到小的和的形式。示例：

$$\begin{aligned} 21 &= 16+4+1 \\ 22 &= 16+4+2 \\ 23 &= 16+4+2+1 \\ 24 &= 16+8 \end{aligned}$$

这有什么意义呢？我们不妨用上述拆分方法对修改操作分组，比如说，对前21个修改，我们会把前16个修改分为一组，第17～20个修改分为第二组，第21个修改作为第三组（因为21=16+4+1）。我们对每组用数据结构进行维护，并支持在线的查询。不妨假设对n元素的组进行数据结构维护以支持在线查询的复杂度是$O(f(n))-O(g(n))$，按照上述的分组方法，显然最多只会分出$O(\log n)$组，查询时在各组内都查询一遍即可，因此单次查询时间复杂度$O(g(n)\log n)$。为什么这么做是对的？注意大前提，修改对询问贡献独立、修改操作互不影响效果，因此修改操作任意分组、分开处理不会影响答案。

而执行修改操作时怎么做呢？事实上，我们只需考虑当前拆分方案和增加一个修改操作后新的分组方案，显然有一些分组不再存在了，而有一个分组多出来了。我们直接删掉不再存在的分组，暴力建出新增的分组即可（如果感觉难以理解，可以看本文附带的演示文稿，其中包含一个图形化的动画解释）。

我们来分析一下这么做的时间复杂度。我们考虑我们重建的所有组的元素总数。实际上可以发现，当添加第k个修改时，我们重建的组的元素个数是$lowbit(k)$，也就是k的二进制表示下最右侧的1所代表的权值[33]。

考虑$lowbit(k)$的取值及其数量，我们很容易推出，总时间复杂度是[34]

$$\begin{aligned}&\sum_{1\le k\le n} O(f(lowbit(k)))\\ =&\sum_{1\le i\le \log n}\left\lfloor\frac{n}{2^{i+1}}+0.5\right\rfloor * f\left(2^i\right)\\ \le&\sum_{1\le i\le \log n} O(f(n))=O(f(n)\log n)\end{aligned}$$

因此，我们发现，通过二进制分组，我们以一个log的代价，把原问题的动态修改操作给去掉了，转化成了没有修改的简化版问题，并且可以支持原问题的在线算法要求。

【例3】经典问题

这是刚才讨论的例2的强化版本。给定一个平面，要求在线支持：

1. 在线插入一个半平面
2. 在线查询一个点是否在当前半平面的交集内

与上道题一样，这题的操作满足“修改独立”的要求，只是题目要求在线算法。因此我们直接套用二进制分组算法，原问题被转化为：

开始时给定若干半平面，要求在线回答一系列询问：

给定一个点，在线查询该点是否在当前半平面的交集内

这个问题可以直接使用例2中给出的解决方法，因为那个方法就是在线的，该方法时间复杂度为$O(n\log n)-O(\log n)$，因此算上二进制分组的复杂度后，本题算法时间复杂度为$O(n\log^2 n)-O(\log^2 n)$。虽然不及使用经典算法用平衡树维护顶点序的$O(\log n)$每次操作，但实际本算法常数比经典算法小不少，而且代码量非常小，实现简单，几乎没有分类讨论等容易导致bug的内容。

顺带提一个小扩展。如果我们不用二进制分解，而用三进制或更高进制的分解呢？可以发现，随着进制数b的升高，单次查询的复杂度也会升高，但

[33]示例：$22=(10110)_2$所以$lowbit(22)=(10)_2=2$

[34]注：与上文一样，这里也要求$O(f(n))\ge O(n)$时该证明才成立。但因为序列的长度已经达到了n，单次处理复杂度几乎不可能小于$O(n)$，如果真的不满足可以具体问题具体分析

修改的总复杂度会降低！这对于某些非常特殊的题目（比如说题目保证操作中98%都是修改之类）时可能会有所用处。

1.4 对删除操作和变更操作的支持

有一类数据结构题，题目允许离线算法，它的修改操作包括插入操作与删除操作，其中虽然插入对询问的贡献独立，插入操作之间也互不影响效果，但删除操作可以取消某个插入操作的效果，这使得修改操作不再完全独立。比如我们在文章开头提到的“动态半平面交”问题。

这类问题的经典做法往往极为复杂，大多需要树套树等复杂的数据结构，有时还需要可持久化数据结构的帮忙，代码量往往极为惊人，细节也往往十分繁杂。但只要充分发挥对时间分治的威力，我们也可以得到简单优美的做法。

【例4】动态半平面交问题

有一个平面，我们要完成一系列操作，包括：

1. 插入一个半平面
2. 删除某次插入的半平面
3. 查询一个点是否在当前半平面的交集内

这个问题就是我们在文章最初提到的问题。现在，我们终于要解决它了。我们不妨继续考虑对时间分治的做法。现在，修改操作（也就是插入操作和删除操作）不完全独立了，因为现在一个插入操作可能会被后面的删除操作撤销掉。我们不妨仍然试图用对时间分治算法来解决。

我们不妨依然把操作序列等分为两半。我们考虑前一半操作中的查询，它们的答案显然与后半部分的插入和删除操作都完全无关，因此它是与原问题完全相同的子问题，可以递归解决。

考虑后一半操作的查询，它们的答案只与两部分内容相关：

1. 前一半操作序列中的所有插入操作中在该询问之前未被删除的部分
2. 后一半操作序列中在该询问之前的且未被删除的插入操作

我们发现，因为后一半序列中的插入操作显然不会在前一半序列被删除，因此“后一半操作序列中在该询问之前的且未被删除的插入操作”这一部分与前一半操作序列完全无关，也是与原问题完全相同的子问题，可以递归解决。

因此，实际我们要处理的内容是：

“前一半操作序列中的插入操作中未被删除的部分”对“后一半操作序列的询问”的贡献。

因此，我们实际的任务是：

一开始给定一组半平面，要求支持：

1. 删除一个半平面

2. 查询一个点是否在所有的半平面中

很不幸，这个问题想要直接解决是非常困难的。但我们发现这个问题中只有删除操作，没有插入操作。于是一个经典的解决方法派上用场了，那就是：时间倒流！

因为我们使用的是离线算法，我们完全可以预知转化后的问题中“初始的半平面集”“删除和查询序列”等所有我们需要的信息。我们先求出最后没有被删除的半平面是哪些，然后逆序处理操作序列，这样，查询没有变，而删除半平面变成了插入半平面！

利用时间倒流，我们再次把问题转化成了：

有一个平面，要求支持：

1. 插入一个半平面

2. 查询一个点是否在所有的半平面中

是不是很熟悉呢？就是我们刚刚解决的例2的问题。再次对时间分治，可以在$O(n\log^2 n)$或$O(n\log n)$（利用归并）的时间复杂度内解决该问题。于是，原问题得到了解决。

时间复杂度分析：假设当前待处理序列长度为n，那么时间复杂度是

$$T(n)=2T\left(\frac{n}{2}\right)+O\left(n\log^2 n\right)$$

使用主定理易解得

$$T(n)=O\left(n\log^3 n\right)$$

此方法代码量远小于经典的线段树套可持久化平衡树的做法。而且，如果使用上文中较复杂的$O(n\log n)$算法解决转化后的问题，我们可以降低时间复杂度到与经典做法相同的$O(n\log^2 n)$。

通过充分发挥对时间分治的威力，我们以一个简单的做法解决了在经典做法中十分困难的问题。现在，只要插入操作贡献独立，且题目允许离线，即使有删除操作或变更操作（变更操作实际等于先删除原操作，再插入新操作），我

们也可以利用对时间分治与时间倒流，以2个log的时间复杂度代价，把题目简化为没有动态插入、没有动态删除、没有动态变更的完全静态版问题！

那么，如果这道题改成要求在线算法，对时间分治或二进制分组依然可以解决吗？对这个问题，我没有找到解决方法，因为第二步中的“时间倒流”已经决定了题目必须允许离线算法。但至少，在应用一次二进制分组算法进行转化后，我们要支持的操作已经没有动态插入了，对某些题目或许也是一个有用的性质。有没有更强大的解决方法尚待读者探索。

2 整体二分算法及其应用

二分答案是常见的技巧，但在数据结构题中，这个方法往往不奏效，原因是我们往往需要预处理一些东西才能快速回答当前二分点的答案，而任何一个答案都可能被二分到，因此预处理的复杂度就不可接受了。但我们也有一个好方法来应对这类问题，那就是：整体二分。

为了规范问题，我们不妨进行以下约定：称当前二分答案的值为“判定标准”，每个修改，结合具体询问和判定标准，可以算出该修改对该询问的判定答案的贡献。询问的判定答案是各个修改的贡献的和，而每个询问都有一个要求的判定答案，根据要求的判定答案与实际的判定答案的关系，我们可以确定询问的真实答案在当前判定标准之上还是之下。

所谓整体二分，需要数据结构题满足以下性质：

1. 询问的答案具有可二分性
2. 修改对判定答案的贡献互相独立，修改之间互不影响效果
3. 修改如果对判定答案有贡献，则贡献为一确定的与判定标准无关的值
4. 贡献满足交换律、结合律，具有可加性
5. 题目允许离线算法

询问的答案具有可二分性显然是前提。我们发现，因为修改对判定标准的贡献相互独立，且贡献的值（如果有的话）与判定标准无关，所以如果我们已经计算过某一些修改对询问的贡献，那么这个贡献永远不会改变，我们没有必要当判定标准改变时再次计算这部分修改的贡献，只要记录下当前的总贡献，在进一步二分时，直接加上的新的贡献即可。

这样的话，我们发现，处理的复杂度可以不再与序列总长度直接相关了，而可以只与当前待处理序列的长度相关。

定义$T(C,S)$表示当前待二分区间长度为C，待二分序列长度为S，不妨设单次处理复杂度为$O(f(n))$，则有[35]

$$\begin{aligned} T(C,S) &= T\left(\frac{C}{2},S_0\right)+T\left(\frac{C}{2},S-S_0\right)+O(f(S)) \\ \text{解之得}\quad T(C,n) &\le O(f(n)\log C) \end{aligned}$$

这样一来，复杂度就可以接受了。通过整体二分算法，我们仅以一个log的代价，便实现了在有预处理的限制下的二分查找，与正常情况下二分查找带来的复杂度相同。

上面的说明可能比较难以理解，因此我提供了整体二分的伪代码框架和例7的参考程序，请务必结合例题仔细理解其正确性与时间复杂度的证明。

【例5】经典问题

给定一个数列，要求回答一系列询问：

给定l,r,k，查询序列的第l个元素到第r个元素组成的子序列的第k大值

我们考虑用整体二分算法解决此题。不妨假设当前整体二分答案为s，则我们只需统计出各个询问区间中大于s的数有多少个即可。如果至少有k个，那么答案必然大于s；否则答案必然小于s，而且，大于s的那部分元素始终对这个询问有着相同的贡献，因此，我们直接把这个询问的k值减掉s，然后分治时不再统计大于s的元素。

那么如何统计出各个询问区间中大于s的元素有多少个呢？很多人的第一想法可能是：开一个数组，如果序列中某个元素大于s，就在数组中对应的位置标记为1，然后求一遍部分和即可。$O(n)-O(1)$，完美的复杂度，不是么？很不幸，还真不是。

如果这么做，我们分析一下总复杂度，会惊讶的发现，总复杂度是：（定义$T(C,S)$表示解决待二分区间长度为C，待二分序列长度为S的问题的复杂度）

$$\begin{aligned} T(C,S) &= T\left(\frac{C}{2},S_0\right)+T\left(\frac{C}{2},S-S_0\right)+O(n) \\ \text{解之得}\quad T(C,n) &= O(nC) \end{aligned}$$

[35]注：与上文一样，这里也要求$O(f(n))\ge O(n)$时该证明才成立。但因为序列的长度已经达到了n，单次处理复杂度几乎不可能小于$O(n)$，如果真的不满足也请具体问题具体分析

Algorithm 1 Divide_And_Conquer(Q, AL, AR)

```
# Q是当前处理的询问序列
# 其中WANT域为要求的判定答案，CURRENT域为已累计的总贡献
# [AL, AR]是当前的答案范围区间
if AL = AR then
  将Q中所有询问的答案均设为AL
  return
end if
AM ← (AL + AR)/2
# Solve是主处理函数，只考虑参数满足判定标准[AL, AM]的修改
# 此时询问序列Q中各个询问的答案被存储至ANS数组
Solve( Q, AL, AM )
# Q1, Q2为两个临时数组，用于划分询问
for i = 1 to Length( Q ) do
  if Q[i].WANT ≤ Q[i].CURRENT + ANS[i] then
    # 当前判定答案不小于要求答案，说明真实答案应当不大于判定答案
    向数组Q1末尾添加Q[i]
  else
    # 当前判定答案小于要求答案，说明真实答案应当大于判定答案
    # 这个算法的关键：把当前贡献累计入总贡献，以后不再重复统计！
    Q[i].CURRENT ← Q[i].CURRENT + ANS[i]
    向数组Q2末尾添加Q[i]
  end if
end for
# 分治，递归处理
Divide_And_Conquer( Q1, AL, AM )
# 注意，这里的调用是Q2而不是Q
# 因为Q1对这部分询问的判定答案的贡献已经被累加到CURRENT域中
Divide_And_Conquer( Q2, AM + 1, AR )
```

与暴力无异！哪里出问题了?

仔细看一看前文中的复杂度式子与这个式子的区别！这是一个初学整体二分的同学非常容易犯的问题。我们的处理复杂度，绝对不能和序列总长n线性相关！只能和当前处理序列区间的长度相关！因此，正确的做法是，对当前处理区间中所有大于s的元素的位置排序，然后二分查找每个询问端点所处的位置。这样复杂度是$O(L\log L)-O(\log L)$，其中L是当前处理序列区间的长度。于是，总复杂度是$O((n+Q)\log n\log C)$，其中C是元素的值的规模。

【例6】矩阵乘法[36]

给定一个矩阵，要求回答一系列询问：

查询某个子矩阵的第k大值

本题做法与上个例题非常相似，只是一维变成了二维。我们应用整体二分算法，问题转化为：

给定一个矩阵，其中若干元素被标记了，要求回答一系列询问：

查询某个子矩阵中被标记元素的个数

额外要求：复杂度不允许与矩阵大小线性相关，只能与被标记元素个数相关

这个问题可以使用二维树状数组解决。这里有个tricky的地方，因为我们的复杂度必须不能与总元素个数线性相关，而只能与被标记的元素个数相关，因此我们做完查询后不能简单地直接清空树状数组（这是与总元素个数相关的操作），而必须一次次逆向修改回去（这才是只与被标记元素个数相关的）以完成清空操作，以待下次继续使用。

时间复杂度$O((n^2+Q)\log^2 n\log C)$。

【例7】ZJOI2013 K大数查询[37]

有n个数集（数集中可以有重复元素）。要求支持：

1. 往第l个数集到第r个数集中都插入一个数k。$(1\le k\le n)$

2. 问你如果把第l个数集到第r个数集中的所有数取出来放在一起，那么这些数中第k大的数是多少。$(1\le k\le 2^{31}-1)$

[36]题目来源http://tsinsen.com/A1333

[37]题目来源：ZJOI2013 Day1 第一题

我们不妨考虑应用一次整体二分算法，那么问题转变为了：

有一个序列，要求支持：

1. 对该序列的一段元素加1

2. 查询该序列的一段元素的和

额外要求：复杂度不允许与序列长度线性相关，只能与操作个数相关

如果没有这个额外要求，想必大家都会做，就是经典的线段树+懒标记。但有了这个额外要求呢？可能有人要说了，使用动态新建结点的线段树云云……

这些都是经典做法，但有更简单的方法吗？如果你想了半天还是没有想法，那么请翻到本文第1页从头再读一遍吧:-)。这个修改操作是满足“修改独立”性质的！直接应用对时间分治，问题被转化为：

开始时给出若干区间，要求回答若干查询：

给定一个区间，要求开始时给出的各个区间与该区间交集的长度的和。

显然每个开始时给出的区间对答案的贡献是一个分段一次函数。直接排序后扫一遍即可。

时间复杂度$O(n\log^3 n)$，如果对时间分治时使用归并排序可优化到$O(n\log^2 n)$。通过整体二分与对时间分治的综合应用，我们成功解决了这道题目。

整体二分算法是一种很有效的处理手段，能大幅简化很多题目。比如说，我们可以把文章开头给出的题目再加强一下，改成要求支持：

1. 插入一个带权的半平面

2. 删除某个插入的半平面

3. 查询覆盖了某个点的半平面中，权值最大的半平面的权值

这个问题如果不用整体二分进行处理的话，几乎无法解决。但整体二分之后，问题就成功转化成了文章开头的动态半平面交问题，从而得到了解决。

3 总结

对时间分治算法、二进制分组算法和整体二分算法，都是解决数据结构题的有效工具。适当地应用这些算法、充分地发挥这些算法的威力，可以大幅降低数据结构题的难度、降低编码难度。这些算法的本质，实际都是通过利用数据结构题的操作中隐藏的特殊性质，例如“修改独立”“贡献独立”“允许离线算法”等，来做到转化问题、简化问题的。这启示我们，要充分挖掘题目特殊

性、利用题目特殊性，从而得到更加优美高效的算法。

4 特别感谢

- 感谢CCF提供了这个交流的平台
- 感谢父母、学校对我的培养
- 感谢贾志鹏、罗雨屏、彭天翼、胡渊鸣、王悦同等诸多通过网络或现实与我交流讨论、给予了我诸多帮助和鼓励的同学
- 感谢罗雨屏、龙浩民等同学在LaTeX方面给我的帮助
- 感谢我的班主任和其他老师对我的支持

参考文献

[1] 《从〈Cash〉谈一类分治算法的应用》陈丹琦

[2] 顾昱洲在WC2013中的授课