`

New algorithm for Half-plane Intersection and its Practical Value

— Thesis for Chinese Team Selecting Contest 2006

半平面交的新算法及其实用价值

——中国代表队 2006 年选拔赛论文

By Zeyuan Zhu

Dedicate to my beloved mother, Xiaoli Xu.

New algorithm for Half-plane Intersection and its Practical Value

Zeyuan Zhu[1]
Grade 12, Nanjing Foreign Language School, Jiangsu, China
朱泽园, 高三, 南京市外国语学校, 江苏, 中国

2005-12-29

Keywords: Half-plane, Intersection, Feasible Region, Algorithm, Polygon, Practical

Abstract

Aim: Present a new *O*(*n*log*n*) algorithm for half-plane intersection (**abbr. HPI**), which is one of the most heatedly discussed problems in computer science; emphasize its advantages in practical application, and to some extent, reduce the complexity to O(n). However, the new algorithm will be extraordinarily easy to be implemented.

主旨:半平面的交是当今学术界热烈讨论的问题之一，本文将介绍一个全新的O(nlogn)半平面交算法，强调它在实际运用中的价值，并且在某种程度上将复杂度下降至O(n)线性。最重要的是，我将介绍的算法非常便于实现.

§1 introduces what half-plane intersection is. §2 prepares a linear algorithm for convex polygon intersection (**abbr. CPI**). Equipped with such knowledge, a common solution for HPI is briefly discussed in §3. Then, my new algorithm emerges in §4 detailedly. Not only as a conclusion of the whole paper, §5 also discuss its further usage practically and compares it with the older algorithm described in §3.

§1什么是半平面交. §2凸多边形交预备知识. §3简要介绍旧D&C算法. §4揭开我的新算法S&I神秘面纱. §5总结和实际运用.

Timestamps: Came up with it in April 2005; implemented partly in June 2005[2]; problem set in July 2005[3]; publicized as a post in USENET, November 6, 2005[4].

[1] E-mail: zhuzeyuan[at]hotmail[dot]com

[2] The sub-problem of HPI appeared in USA Invitational Computing Olympiad contest.

[3] Set an HPI problem in Peking University Online Judge, with brief introduction about the algorithm.

[4] URL: http://groups.google.com/group/sci.math/msg/68e9d9fc634f0d92

1. Introduction

A line in plane is usually represented as $ax+by=c$. Similarly, its inequality form $ax+by\le c$ represents a ***half-plane*** (also named ***h-plane*** for short) as one side of this line. Notice that $ax+by\le c$ and $-ax-by\le -c$ show opposite h-planes unlike $ax+by=c$ and $-ax-by=-c$. Half-Plane Intersection (**abbr. HPI**) considers the following problem:

众所周知，直线常用 ax+by=c 表示，类似地半平面以 ax+by≤(≥)c 为定义。

Given *n* half-planes, $a_ix+b_iy\le c_i$ (1≤i≤n), you are to determine the set of all points that satisfying all the *n* inequations. 给定n个形如$a_ix+b_iy\le$ci的半平面，找到所有满足它们的点所组成的点集

As **Figure i** describes, the feasible region, which is the intersection, forms a shape of convex hull but possibly unbounded. However, we shall add four h-planes forming a rectangle, which is large enough to make sure the area after intersections finite. *In the following sections, we suppose the feasible region is* ***bounded*** *with a finite area.*

合并后区域形如凸多边形，可能无界。此时增加 4 个半平面保证面积有限

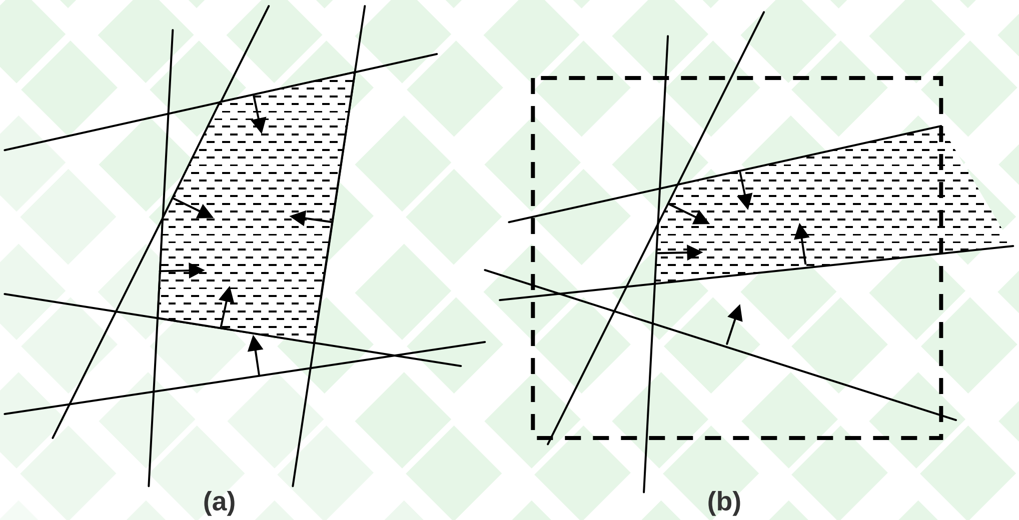


Figure i Two examples of HPI. (b) gives an example of unbounded intersection area.

Each h-plane builds up at most one side of the convex polygon, hence, the convex region will be bounded by at most *n* edges. Pay attention the intersection sometimes yields a line, a ray, a line-segment, a point or an empty region. 每个半平面最多形成相交区域的一条边，因此相交区域不超过 n 条边。

注意相交后的区域，有可能是一个直线、射线、线段或者点，当然也可能是空集。

2. Convex Polygon Intersection **(abbr. CPI)**

Intersecting two convex polygons **A** and **B** into a single one, can be properly solved via Line Segment Intersection in $O(n\log n)$ time, when there are $O(n)$ edges. We will sketch out an easier and more efficient way, named ***plane sweep method***.
求两个凸多边形 A 和 B 的交（一个新凸多边形）。我们描绘一个平面扫描法。

The main idea is to calculate intersections of edges as cutting points, and break boundaries of A and B, into *outer edges* and *inner edges*. The segments of *inner edges* establish ties to each other, and form the shape of a polygon, which is the expected polygon after intersection. In **Figure ii**, *inner edges* are indicated by thick segments, which form a bold outline of the intersection. 主要思想: 以两凸边形边的交点为分界点，将边分为内、外两种。内边互相连接，成为所求多边形。

Suppose there is a vertical sweep line, performing left-to-right sweep. The x-coordinates to be swept are called ***x-events***. At anytime, there are at most four intersections from sweep line to either given polygon[5]:

假设有一个垂直的扫描线，从左向右扫描。我们称被扫描线扫描到的 x 坐标叫做 x 事件。任何时刻，扫描线和两个多边形最多 4 个交点

- to the upper hull of **A** (name that intersection **Au** for short)
- to the lower hull of **A** (name that intersection **Al** for short)
- to the upper hull of **B** (name that intersection **Bu** for short)
- to the lower hull of **B** (name that intersection **Bl** for short)

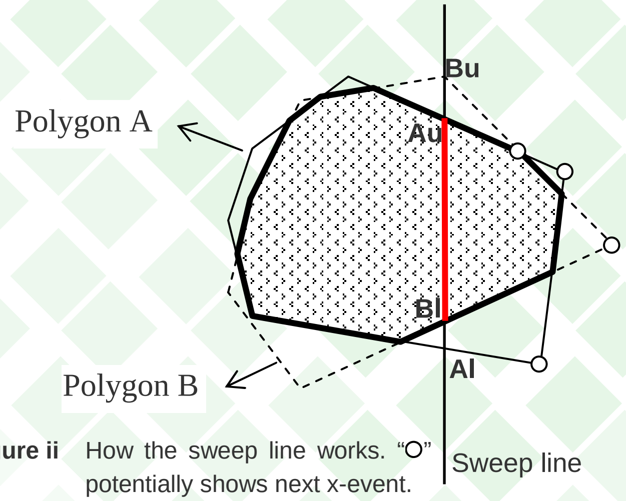


Figure ii How the sweep line works. "○" potentially shows next x-event.

Look at **Figure ii**, the lower one between intersections **Au** and **Bu**, and the upper one between intersections **Al** and **Bl**, form an interval of the current inner region – the red

[5] Assume there is no edge in polygons parallel to the sweep line. If such situation happens, we could rotate the plane in proper angle, or else, we need good sense to judge a great many special cases.

segment in bold. Au、Bu 中靠下的，和 Al、Bl 中靠上的，组成了当前多边形的内部区域。

Obviously, the sweep line may not go through all the x-event with rational coordinates. Call the edges where **Au**, **Al, Bu** and **Bl** are: ***e1, e2, e3*** and ***e4*** respectively. The next x-event should be chosen among four endpoints of ***e1, e2, e3*** and ***e4***, and four potential intersections: ***e1*∩*e3***, ***e1*∩*e4***, ***e2*∩*e3*** and ***e2*∩*e4***.

当然，我们不能扫描所有有理数！称 Au, Al, Bu, Bl 所在的边叫做 e1,e2,e3,e4，下一个 x 事件将在这四条边的端点，以及两两交点中选出。

The above operation could be implemented with $O(n)$ running time, since there are $O(n)$ x-events, and the maintenance of **Au**, **Al, Bu** and **Bl** takes only $O(1)$.

3. Common solution: Divide-and-Conquer Algorithm (abbr. D&Q)

The basic approach is simple, depends on divide-and-conquer idea.

- Divide: Partition the n h-planes into two sets of size $\lfloor \frac{n}{2} \rfloor$ and $\lceil \frac{n}{2} \rceil$.
- Conquer: Compute the feasible region (intersection) recursively of both two sub-sets.
- Combine: Compute the intersection of two convex polygons, by linear CPI algorithm described in §2.
- 分: 将 n 个半平面分成两个 n/2 的集合.
- 治: 对两子集合递归求解半平面交.
- 合: 将前一步算出来的两个交(凸多边形)利用第 2 章的 CPI 求解.

The total time complexity of the solution can be calculated via recursive equation: 总时间复杂度可以用递归分析法.

$$T(n) = 2T(\tfrac{n}{2}) + O(n)$$
$$T(n) = O(n\log n)$$

4. My New Solution: Sort-and-Incremental Algorithm (abbr. S&I)

Definition of h-plane’s polar angle:

- for the h-plane like *x-y≥constant*, we define its polar angle to ¼π.
- for the h-plane like *x+y≤constant*, we define its polar angle to ¾π.
- for the h-plane like *x+y≥constant*, we define its polar angle to -¼π.

- for the h-plane like $x\text{-}y \leq constant$, we define its polar angle to -¾π.

半平面的极角定义: 比如 x-y 常数的半平面，定义它的极角为¼ π .

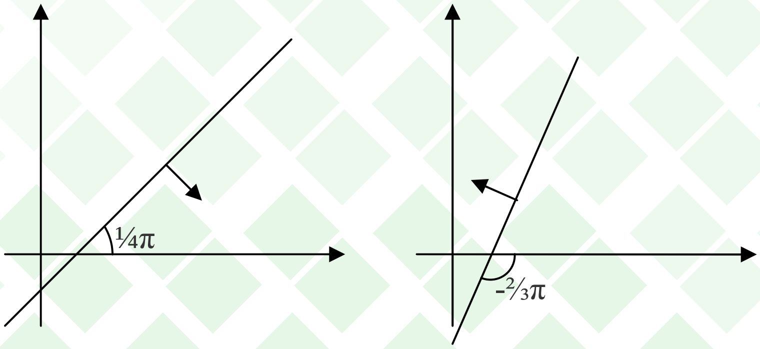


Figure iii Definition of half-plane’s polar angle.

Definition of h-plane’s constant:

- for the h-plane like $ax+by \leq c$, we say its constant is c.

My new ***Sort-and-Incremental Algorithm*** seems lengthy since I am going to introduce it in details:

Step 1: Separate the h-planes into two sets. One has polar angles of (-½π, ½π], the other has those of (-π, -½π]∪ (½π, π].[6]

Step 1: 将半平面分成两部分，一部分极角范围(-½ π , ½ π]，另一部分范围(- π , -½ π]∪(½ π , π]

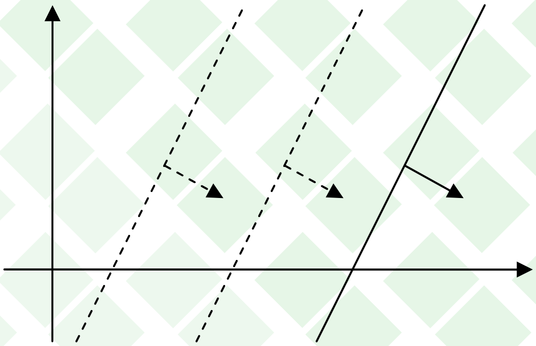
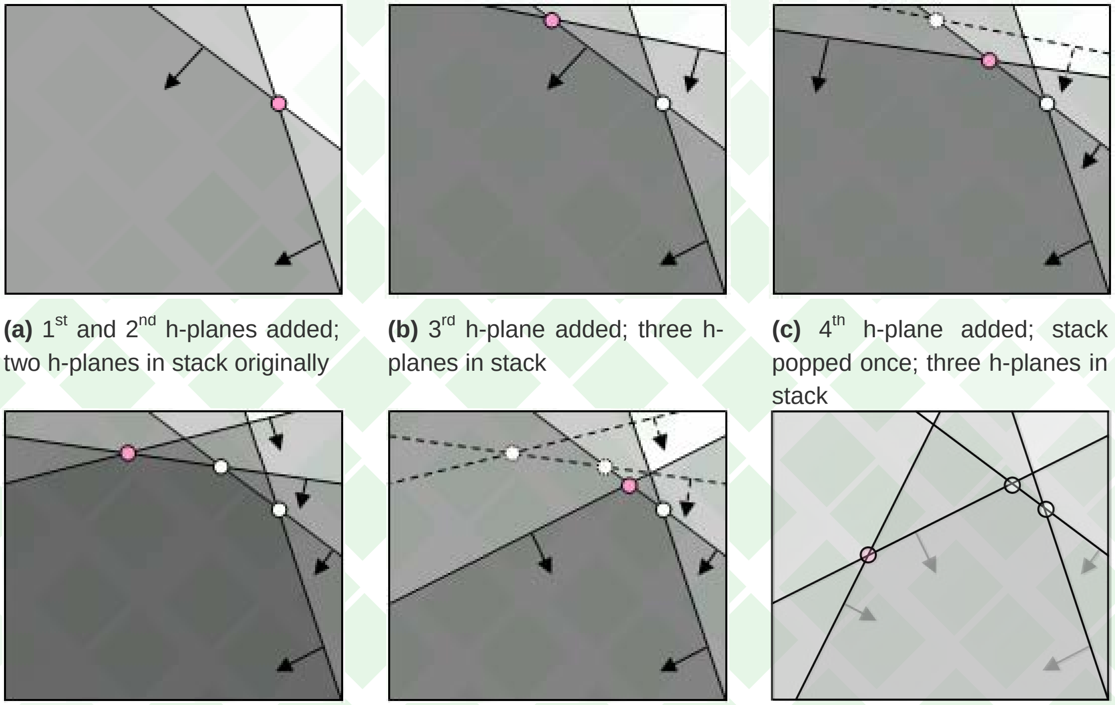


Figure iv For half-planes with the same polar angle, we must keep the one that is covered by others.

Step 2: Consider the h-plane set of polar angles in (-½π, ½π] (the other set should also go through step 3 and 4 similarly). Sort them by the polar angle. For the h-planes with the same polar angle, we can keep only one down (delete all others) according to the constant of these h-planes – see **Figure iv** above.

[6] In fact, we only need to separate them into two parts, one contains (a, $a+\pi$], the other contains ($a+\pi$, $a+2\pi$]. but my separation will make the explanation clearer

Step 2:考虑(-½ π , ½ π]的半平面(另一个集合类似地做 Step3/4)，将他们极角排序。对极角相同的半平面，根据常数项保留其中之一。



(a) 1st and 2nd h-planes added; two h-planes in stack originally

(b) 3rd h-plane added; three h-planes in stack

(c) 4th h-plane added; stack popped once; three h-planes in stack

(d) 5th h-plane added; four h-planes in stack

(e) 6th h-plane added; top two h-planes popped; three h-planes in stack

(f) 7th h-plane added; four h-planes in stack

Figure v An example for **Step 3**. Old intersections to the left of the new ones are to be deleted. Dotted portion indicates the part to be deleted.

Step 3: Starting from two h-planes with the least polar angle, compute their intersection. Push them two into a stack.

从排序后极角最小两个半平面开始，求出它们的交点并且将他们押入栈。

Each time, add one more h-plane by increasing order of polar angles, and calculate the intersection of it and the top h-plane in stack. If this intersection is to the right of the intersection of top two h-planes in stack, we pop the stack once. Later, we check the intersection of two new top h-planes in stack... Do this repeatedly until it is to the left of the top intersection. At last, we push the current h-plane into the stack.

每次按照极角从小到大顺序增加一个半平面，算出它与栈顶半平面的交点。如果当前的交点在栈顶两个半平面交点的右边，出栈（pop）。前问我们说到出栈，出栈只需要一次么？Nie!我们要继续交点检查，如果还在右边我们要继续出栈，直到当前交点在栈顶交点的左边。

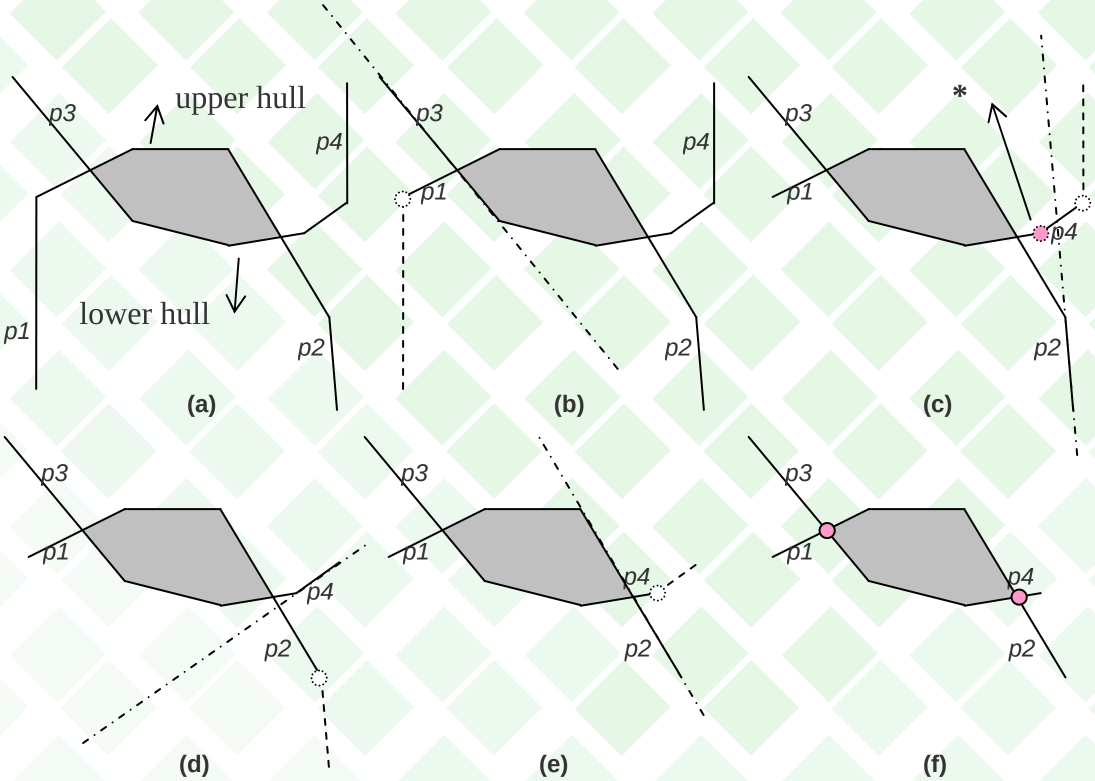
Step 4: The intersections of adjacent half-plane pairs in stack form half a convex polygon. For the two sets, we have two halves the convex polygon - (-½π, ½π] gives an upper hull and (-π, -½π]∪ (½π, π] gives a lower hull. Merging them in linear time can be done via CPI in *section 2*, but we can make use of *iteration*.

Step 4:相邻半平面的交点组成半个凸多边形。我们有两个点集，(-½ π , ½ π]给出上半个，(- π , -½ π]∪(½ π , π]给出下半个

Based on the knowledge of deque (double-ended queue), we repeatedly remove an end-point that is known to be infeasible. Look at **Figure vi**.

At the beginning, four pointers *p1*, *p2*, *p3* and *p4* indicate leftmost/rightmost edges of both upper and lower hulls. *p1* and *p3* move rightwards, while *p2* and *p4* walks leftwards. At anytime, if the leftmost intersection is against the feasible region provided by *p1* or *p3*, we are sure the leftmost one is to be removed. Naturally, *p1* or *p3* walks rightwards to its adjacent edge. The judgment holds analogously for rightmost intersection with *p2* and *p4*.

初始时候，四个指针 p1, p2, p3 and p4 指向上/下凸壳的最左最右边。p1, p3 向右走，p2, p4 向左走。任意时刻，如果最左边的交点不满足 p1/p3 所在半平面的限制，我们相信这个交点需要删除。p1 或 p3 走向它右边的相邻边。类似地我们处理最右边的交点。重复运作直到不再有更新出现——迭代。



(a)

(b)

(c)

(d)

(e)

(f)

Figure vi An example for **Step 4**. In figure (c), only one intersection besides *p4* can be removed, since the other (pink one marked with *) is in proper side of *p2*'s h-plane. That is why every time we try to remove only a single rightmost or leftmost intersection.

Do the above removing repeatedly until there is no more update. Meanwhile, the number of intersections for a single hull can decrease to zero, but still with non-empty feasible region:

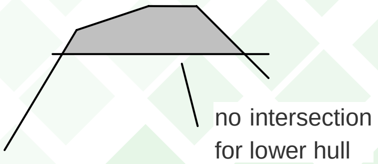


Figure vii Number of intersections for a single hull can decrease to zero.

Everything except sorting (Step 2) in S&I algorithm remain linear – *O*(*n*) running time. Usually we use quick-sort. The total complexity is *O*(*n*log*n*), with fairly small constant factor hidden. **It can be reduced to *O*(*n*), see the next section.**

除了 Step2 中的排序以外，S&I 算法的每一步都是线性的。通常我们用快速排序实现 Step2，总的时间复杂度为 O(nlogn)，隐蔽其中的常数因子很小

5. Practical Value and Linear approach

Great ideas need landing gear as well as wings. S&I algorithm seems to work in the same time complexity as D&C algorithm, but some overwhelming advantages of implementing S&I holds:

Great ideas need landing gear as well as wings. S&I 算法似乎和 D&C 算法时间复杂度相同，但是它有着压倒性(overwhelming)的优势。

- It is much easier to code S&I program than D&C one. The program in C++ programming language takes less than **3KB**.
  新的 S&I 算法代码容易编写，相对于 D&C 大大简单化，C++程序语言实现 S&I 算法仅需 3KB 不到.
- The coefficient hidden under S&I algorithm’s complexity is extraordinarily smaller than D&C, since we no longer need *O*(*n*log*n*) number of intersection calculations. In general speaking, S&I program runs approximately five times faster than D&C one.
  S&I 算法复杂度中的系数，远小于 D&C，因为我们不再需要 O(nlogn)次交点运算. 通常意义上来讲，S&I 程序比 D&C 快五倍。

  Remark: intersection calculations play the most important role in both algorithms due to its operational speed (so slow, equivalent to hundreds of addition operations). CPI, the preparative algorithm which will be called several times from D&C, requires *O*(*n*) number of calculations, wherefore it rises the total running time up. Besides, S&I calculates *O*(*n*) times in any case.

- If the given h-planes are all in (-½π, ½π] (or any span of π), S&I program can be shorten remarkably (to approximately **twenty** lines in C++), but D&C program may not. An informatics problem appeared in USA Invitational Computing Olympiad contest with such purpose. 如果给定半平面均在(-½π, ½π]（或任意一个跨度为π的区间），S&I 算法可被显著缩短，C++程序只需要约二十行。USAICO 比赛中就出现了这样一题。

Asymptotical Optimization to linear: The bottleneck of this algorithm is sorting. Pay attention the sorting is NOT a ***comparison sort*** (sorting based on comparison)! The key words for elements to be sorted are polar angles, which can be certainly determined by gradient – a decimal fraction. Since then, we can replace the $O(n\log n)$ quick-sort to $O(n)$ ***radix-sort***. **The asymptotical complexity of algorithm can decrease to *O*(*n*)**. Anyway $O(n)$ approach usually runs slower than $n\log n$ ones for its additional memory usage!
本算法瓶颈是排序，这里的排序不是比较排序，因此可以将快速排序替换成基数排序，降低程序渐进时间复杂度到线性。

The sentiment of my creation: An invention does not attribute to someone who comes up with ideas. Most people have ideas. The difference between the average person and the inventor is that the inventor for some reason believes only himself, and has the urge to see his ideas through to fruition. *Henri Matisse*, a French painter and sculptor, ever said 'there is nothing more difficult for a truly creative painter than to paint a rose, because before he can do so, he has first to forget all the roses that were ever painted.' Equipped with both idealistic and practical spirit of innovation, I am on the way. How about you?

Bibliography

1. Kurt Mehlhorn. *Data Structures and Algorithms Vol 3*. Springer-Verlag, New York, 1984.
2. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to algorithms 2nd Edition*. MIT Press, New York, 2001
3. Gill Barequet. *Computational Geometry Lecture 4*, Spring 2004/2005.
4. Kavitha Telikepalli. *Algorithms and Data Structures, Lecture 3*, Winter 2003/2004.