An optimal algorithm for finding segments intersections

Ivan J. Balaban

Keldysh Institute of Applied Mathematics, 125047, Miusskaya Sq. 4, Moscow, Russia.

Abstract

This paper deals with a new deterministic algorithm for finding intersecting pairs from a given set of N segments in the plane. The algorithm is asymptotically optimal and has time and space complexity $O(N \log N + K)$ and $O(N)$ respectively, where K is the number of intersecting pairs. The algorithm may be used for finding intersections not only line segments but also curve segments.

INTRODUCTION

Reporting of segments set intersections is one of the fundamental problem of computational geometry. It is known, that within the model of algebraic decision tree any algorithm solving this problem needs $\Omega(N \log N + K)$ time [1, 5, 10]. The well–known Bentley and Ottman's algorithm based on the plane sweep has $O((N+K)\log N)$ time and $O(N)$ space complexity [2]. Later, Chazelle obtained an $O(N \log^2 N / \log\log N + K)$, $O(N + K)$ algorithm [4]. Time optimal algorithm $O(N \log N + K)$ was proposed by Chazelle and Edelsbrunner with space requirement $O(N + K)$ [5]. Mulmuley using randomized approach offered an algorithm with expected running time $O(N \log N + K)$ and space requirement $O(N + K)$ [9]. Algorithm of Clarkson and Shor also based on the randomized approach has expected running time $O(N \log N + K)$ and space requirement $O(N)$ [6].

In this paper we present a deterministic, asymptotically optimal for both time $O(N \log N + K)$ and space $O(N)$ algorithm for finding intersecting segments pairs. As by product, we construct an $O(N \log^2 N + K)$, $O(N)$ algorithm that could be of practical interest due to its relative simplicity.

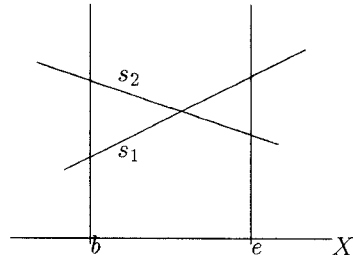

Figure 1: $s_1 <_b s_2$, $s_2 <_e s_1$.

PRELIMINARIES

Suppose we have a set S_0 consisting of N segments in the plane. The problem is to find all intersecting pairs. The set of these pairs we denote as $Int(S_0)$ and $|Int(S_0)|$ we denote as K.

To find $Int(S_0)$ we use a collection of vertical strips on the plane organized in a tree structure, connect each of the strips with some subset of S_0 and introduce procedures for finding intersections of such subsets. These procedures require appropriate ordering of the subsets. To obtain this ordering we use a method analogous to plane sweep but instead of sweeping the plane by a vertical line we use preorder traversal of the strips tree (i.e sweeping the tree by its branch).

So, the primary objects of our algorithm are segments, ordered and unordered sets of segments, pairs of intersecting segments and vertical strips.

Let $\langle b, e\rangle$ denote the *vertical strip* $b \le x \le e$ and let l and r be abscissae of endpoints of segment s $(l < r)$. We call the segment s

- *spanning the strip* $\langle b, e\rangle$ if $l \le b < e \le r$,
- *inner for the strip* $\langle b, e\rangle$ if $b < l < r < e$,
- *crossing the strip* $\langle b, e\rangle$ if $[l, r[\cap[b, e[\neq \emptyset$.

Two segments s_1 and s_2 are called *intersecting within the strip* $\langle b, e\rangle$ if their intersection point lies within this strip.

For two segments sets S and S' we denote by $Int(S, S')$ a set $\{\{s, s'\} | s \in S,\ s' \in S'$ and s intersect $s'\}$. Notations $Int_{b,e}(S)$ and $Int_{b,e}(S, S')$ will be

used to describe subsets of $Int(S)$ and $Int(S, S')$ consisting of segments pairs intersecting within the strip $\langle b, e\rangle$. Hereafter the brackets $\{\}$ are used to define unordered sets, and the brackets $()$ are used to define ordered sets.

We shall use the following order relation among segments [2, 10]: $s_1 <_a s_2$ if segments s_1 and s_2 intersect a vertical line $x = a$ and the intersection point with s_1 lies under the intersection point with s_2 (see Fig.1).

Staircase D is the pair $(Q, \langle b, e\rangle)$, where segments set Q possesses the following properties

- $\forall s \in Q$ s spans the strip $\langle b, e\rangle$.
- $Int_{b,e}(Q) = \emptyset$.
- Q is ordered by $<_b$.

Intersections of segments of Q with $\langle b, e\rangle$ are called *stairs* of D. *Intersection of the staircase D with a segments set S* denoted as $Int(D, S)$ is the set $Int_{b,e}(Q, S)$.

We call the staircase D *complete relative to a set S* if each segment of S either does not span the strip $\langle b, e\rangle$ or intersects one of the stairs of D.

Lemma 1 *If a staircase D is complete relative to a set S with S consisting of segments crossing the strip $\langle b, e\rangle$ then*

$$|S| \leq Ends_{b,e}(S) + |Int(D, S)|,$$

where $Ends_{b,e}(S)$ is the number of endpoints of S within the strip $\langle b, e\rangle$.

If a point p of a segment s is placed above the ith and below the $i+1$th stairs of D, then the number i is called *location of s on the staircase D* and denoted as $Loc(D, s)$. (see Fig.2).

Let $Loc(D, s) = i$. To find $Int(D, \{s\})$ it is necessary to reach the first stair that does not intersect s moving down from the ith stair and then to repeat this procedure moving up from the $i+1$th stair. This procedure has time complexity $O(1 + |Int(D, \{s\})|)$.

Thus, if we knew $Loc(D, s)$ for all $s \in S$ then we could find $Int(D, S)$ in $O(|S| + |Int(D, S)|)$ time. It is easy to find $Loc(D, s)$ for any segment $s \in S$ in $O(log|Q|)$ time by binary search. If the set $S = (s_1, ..., s_n)$ is ordered by $<_x$, where $x \in [b, e]$, it is possible to find $Loc(D, s_i)$, $i = 1, ..., n$, in $O(|Q| + |S|)$ time sequentially scanning the stairs of D and the segments of S. It follows from this observation that

Proposition 1 *Given a staircase $D = (Q, \langle b, e\rangle)$ and a segments set S, the set $Int(D, S)$ can be found in $O(|S| \log |Q| + |Int(D, S)|)$ time. If S is ordered by $<_x$, $x \in [b, e]$, it is possible to find $Int(D, S)$ in $O(|S| + |Q| + |Int(D, S)|)$ time.*

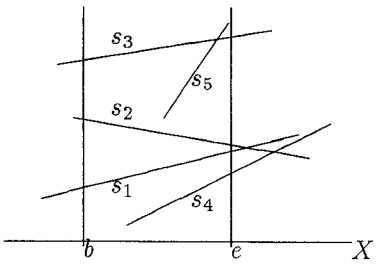

Figure 2: $D = ((s_1, s_2, s_3), \langle b, e\rangle)$, $Loc(D, s_4) = 0$, $Loc(D, s_5) = 2$ *or* 3, $Int(D, \{s_4, s_5\}) = \{\{s_3, s_5\}\}$

INTERSECTIONS IN A STRIP

Consider the following problem: *Let L be a set of segments spanning the strip $\langle b, e\rangle$ and ordered by $<_b$. We wish to find $Int_{b,e}(L)$ and to reorder L by $<_e$. We denote the reordered set L by R.*

Let us split the set L into subsets Q and L' so that the staircase $D = (Q, \langle b, e\rangle)$ be complete relative to L'. The intersections of segments of L is computed in two phases. The first phase is to find the intersection D with L'. The second phase is to find all intersections in L' . To find the intersections in L' we may split the set again etc.

As a result the problem of finding the intersections in L is reduced to the following two problems: how to split L efficiently and how to find intersections between D and L'.

The following procedure splits a set L ordered by $<_b$ into subsets Q and L' ordered by $<_b$ so that staircase $(Q, \langle b, e\rangle)$ is complete relative to L'.

```
  procedure Split_{b,e}(L, Q, L');
{Let L = (s_1, ..., s_k), s_i <_b s_{i+1}}
    L' := ∅; Q := ∅;
    For j = 1, ..., k do
      If the segment s_j doesn't intersect
      the last segment of Q within ⟨b, e⟩ and
      spans this strip then
        add s_j to the end of Q
      else
        add s_j to the end of L';
      endif;
    endfor;
  end procedure.
```

This procedure runs in $O(|L|)$ time. Thus, given L we may find $Int_{b,e}(L)$ and R using the following recursive procedure:

```
  procedure SearchInStrip_{b,e}(L, R)
    Split(L, Q, L');
    If L' = ∅ then R := Q; exit; endif;
```

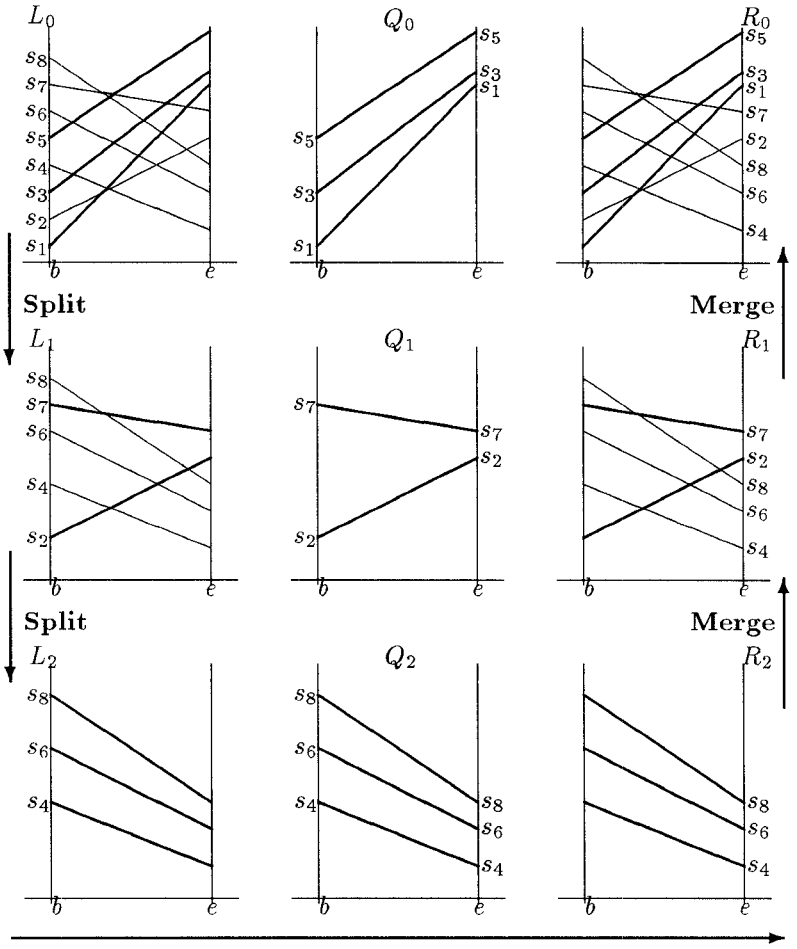

Figure 3: Example of SearchInStrip execution.

Find $Int_{b,e}(Q, L')$;
SearchInStrip$_{b,e}(L', R')$;
$R := Merge_e(Q, R')$;
end procedure.

Here $Merge_x(S1, S2)$ is a procedure of merging sets $S1$ and $S2$ ordered by $<_x$. Running time of the procedure SearchInStrip equals the sum of running times of its calls. It is easy to understand that the ith procedure call runs in $O(|L_i| + |Int_{b,e}(Q_i, L'_i)|)$ time, where L_i, Q_i, L'_i are appropriate sets ($L_0 = L$, $L_{i+1} = L'_i$). Taking into account Lemma 1 we conclude that the procedure SearchInStrip$_{b,e}(L, R)$ runs in $O(|L| + |Int_{b,e}(L)|)$ time.

INTERMEDIATE ALGORITHM

Let's apply proposed approach to find $Int(S)$, where S is some set of segments. Let all segments of S be placed within a strip $\langle b, e\rangle$. Thus, at the beginning we have a pair S, $\langle b, e\rangle$. The following procedure is used.

The set S is splitted into subsets Q and S' so that the staircase $D = (Q, \langle b, e\rangle)$ be complete relative to S'. It is necessary to find the intersection D with S' and then all intersection in S'. To find the S' intersections we cut the strip $\langle b, e\rangle$ and the set S' along the vertical line $x = c$ into the strips $\langle b, c\rangle$, $\langle c, e\rangle$ and the sets S'_{ls}, S'_{rs} respectively, where c is the median of endpoints between b and e. Then, we recursively apply the procedure to the pairs S'_{ls}, $\langle b, c\rangle$ and S'_{rs}, $\langle c, e\rangle$.

The key fact is that according to Lemma 1 $|S'| \leq Ends_{b,e}(S') + Int(D, S')$, thus, the number of additional segments appearing during cutting is proportional to the number of found intersections.

Let us give a somewhat more detailed explanation of the algorithm.

Without loss of generality, we could assume that all end and intersection points have different abscissae. Within the scope of this paper the abscissae of segment's ends can be normalized by replacing each of them by its rank in their left-to-right order. We consider these abscissae as the integers in the range $[1, 2N]$. *So, let p_i, i and $s(i)$ be the ith endpoint its abscissa and the segment to which it belongs, respectively.*

The central point of our algorithm is a recursive pro-

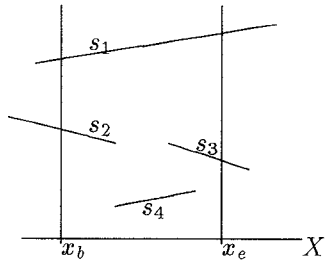

Figure 4: $S_v = \{s_1, s_2, s_3, s_4\}$, $I_v = \{s_4\}$, $L_v = (s_2, s_1)$, $R_v = (s_3, s_1)$.

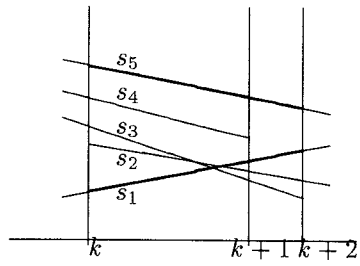

Figure 5: $D_v = ((s_1, s_5), \langle k, k+2\rangle)$, $L_v = (s_1, s_2, s_3, s_4, s_5)$, $L_{ls(v)} = (s_2, s_3, s_4)$, $R_{ls(v)} = (s_3, s_2, s_4)$, $L_{rs(v)} = (s_3, s_2)$, $R_{rs(v)} = (s_3, s_2)$, $R_v = (s_3, s_2, s_1, s_5)$.

cedure TreeSearch. We connect each of the procedure calls with a node in some binary tree referred to as *recursion tree*. We mark all values, sets and parameters of a call by a label of the corresponding node and identify calls and corresponding nodes. As a result we shall analyze our algorithm examining the recursion tree. Let RT be the set of all nodes of recursion tree, and V be the set of internal nodes. We shall define some values, sets and notations inside the bodies of procedures.

procedure IntersectingPairs(S_0).
Sort the $2N$ endpoints by abscissa and find p_i, $s(i)$, $i = 1, ..., 2N$; $S_r := S_0$;
TreeSearch($S_r, 1, 2N$);
end procedure.

procedure TreeSearch(S_v, b, e).
1. **If** $e - b = 1$ **then**
   L_v:=sort S_v by $<_b$; SearchInStrip$_{b,e}(L_v, R_v)$; exit;
   endif;
2. Split S_v into Q_v and S'_v so that staircase $D_v := (Q_v, \langle b, e\rangle)$ be complete relative to S'_v;
3. Find $Int(D_v, S'_v)$;
4. $c := \lfloor (b+e)/2 \rfloor$;
5. Place segments of S'_v crossing the strip $\langle b, c\rangle$ into $S_{ls(v)}$ and the strip $\langle c, e\rangle$ into $S_{rs(v)}$;
6. TreeSearch($S_{ls(v)}, b, c$);
7. TreeSearch($S_{rs(v)}, c, e$);

end procedure.

Hereafter $ls(v)$, $rs(v)$ and $ft(v)$ denote, respectively, the left son, the right son and the father node of the node v.

Our task is to show how to perform all operations connected with node v in $O(|S_v| + |Int(D_v, S'_v)| + (e_v - b_v)\log N)$ time and to prove that $\sum_v |S_v| = O(N \log N + K)$ (obviously $\sum_v |Int(D_v, S'_v)| \le K$).

The procedure TreeSearch is similar to the procedure SearchInStrip. The main difference is that SearchInStrip calls itself without changing the strip, and TreeSearch cuts the current strip into two parts, then, applies itself to both. Another difference is that the set S_v is not ordered as L. As a result we can't directly use the procedure Split for the efficient splitting the set S_v.

To resolve this problem we represent a set S_v as a union of three sets: a set L_v ordered by $<_b$, an unordered set I_v, and a set R_v ordered by $<_e$. We place segments of S_v intersecting the line $x = b$ into L_v, segments intersecting the line $x = e$ into R_v, segments inner for $\langle b, e\rangle$ into I_v (see Fig.4).

Now we may apply the procedure Split to the set L_v and construct Q_v in $O(|L_v|) = O(|S_v|)$ time. But we encounter a new problem. Given sets L_v, R_v and I_v, it is necessary to find corresponding sets of the sons of v.

The unordered sets $I_{ls(v)}$ and $I_{rs(v)}$ are easily constructed. The set $L_{ls(v)}$ will be found by applying the procedure Split$_{b,e}(L_v, Q_v, L_{ls(v)})$ for the third step of TreeSearch. The set $L_{rs(v)}$ can be obtained from $R_{ls(v)}$ in linear time by inserting (if p_c is the left endpoint) or deleting (if p_c is the right endpoint) the segment $s(c)$ But how to obtain $R_{ls(v)}$ from L_v, R_v and I_v without sorting?

For the leaves we execute step 1 and obtain R_v from L_v. Imagine that L_v and I_v are known and both of v sons are leaves. At first we execute the procedure Split$(L_v, Q_v, L_{ls(v)})$ and find Q_v and $L_{ls(v)}$. Now we have to find $Int(D_v, S'_v) = Int(D_v, L_{ls(v)}) \cup Int(D_v, I_v) \cup Int(D_v, R_{rs(v)})$, but we don't know $R_{rs(v)}$, and we may find $Int(D_v, L_{ls(v)}) \cup Int(D_v, I_v)$ only. Then, according to TreeSearch, we apply SearchInStrip to $L_{ls(v)}$ and receive $R_{ls(v)}$. The set $L_{rs(v)}$ is obtained from $R_{ls(v)}$ by inserting or deleting the segment $s(c)$. After that, we apply SearchInStrip to $L_{rs(v)}$ and find $R_{rs(v)}$. Now we may complete the computation $Int(D_v, S'_v)$ by computing $Int(D_v, R_{rs(v)})$ and may build R_v by merging Q_v and $R_{rs(v)}$ (See Fig.5).

The resulting procedures are listed below.

procedure IntersectingPairs(S_0).
Sort the $2N$ endpoints by abscissa and find p_i, $s(i)$, $i = 1, ..., 2N$;
$L_r := (s(1))$; $I_r := S_0 \setminus (\{s(1)\} \cup \{s(2N)\})$;
TreeSearch($L_r, I_r, 1, 2N, R_r$);

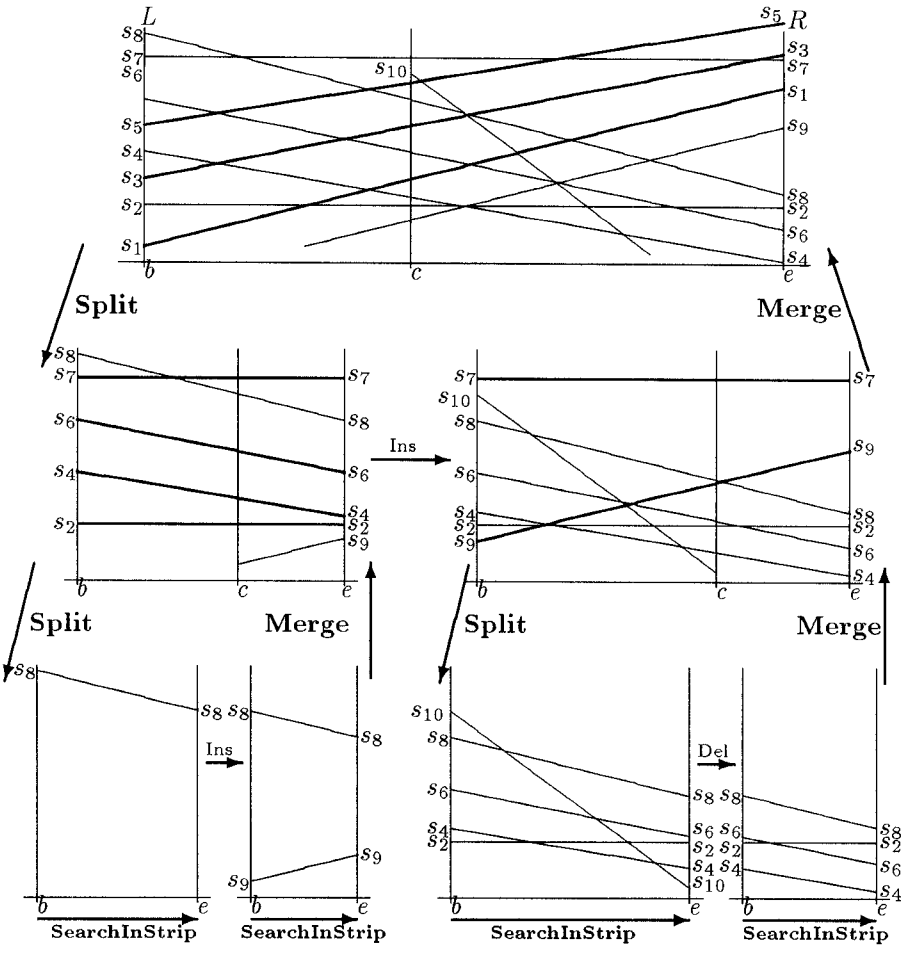

Figure 6: Examle of TreeSearch execution. Ins. denote Insert, Del. denote Delete (step 7).

end procedure.

procedure TreeSearch(L_v, I_v, b, e, R_v).

1. **if** $e - b = 1$ **then**
   SearchInStrip$_{b,e}(L_v, R_v)$; exit;
   endif;
2. Split$_{b,e}(L_v, Q_v, L_{ls(v)})$; $D_v := (Q_v, \langle b, e \rangle)$;
3. Find $Int(D_v, L_{ls(v)})$;
4. $c := \lfloor (b + e)/2 \rfloor$;
5. Place segments of I_v
   inner for the strip $\langle b, c \rangle$ into $I_{ls(v)}$,
   inner for the strip $\langle c, e \rangle$ into $I_{rs(v)}$;
6. TreeSearch($L_{ls(v)}, I_{ls(v)}, b, c, R_{ls(v)}$);
7. **If** p_c is the left endpoint of $s(c)$ **then**
   $L_{rs(v)}$:= insert $s(c)$ into $R_{ls(v)}$
   else
   $L_{rs(v)}$:= delete $s(c)$ from $R_{ls(v)}$;
   endif;
8. TreeSearch($L_{rs(v)}, I_{rs(v)}, c, e, R_{rs(v)}$);
9. Find $Int(D_v, R_{rs(v)})$;
10. **For** $s \in I_v$ **do**
    find $Loc(D_v, s)$ using binary search;
    endfor
11. Find $Int(D_v, I_v)$ using values
    found in the previous step;
12. $R_v := Merge_e(Q_v, R_{rs(v)})$;

end procedure.

We must be careful executing the 9th step of our algorithm as the sets $R_{rs(v)}$ and $L_{ls(v)}$ could have common segments (segments spanning $\langle b, e \rangle$). We have found their intersections with D_v during the 3rd step and should not output these intersections again.

At first, we evaluate the space requirement of this algorithm. The algorithm uses the recursive procedure TreeSearch. A sequence of the procedure calls may occupy memory. The sequence can be represented by a path from the root of the recursion tree to a node. We call this node and the corresponding call *active*. The active call occupies $O(N)$ space, each of its "ancestors" retains $O(|I_v| + |Q_v|)$ memory, other structures use $O(N)$ memory. It is easy to show that for any path pt from the root of recursion tree to some of its nodes $\sum_{v \in pt}(|I_v| + |Q_v|) = O(N)$ ($|I_v| \leq e_v - b_v$, $e_v - b_v \leq$

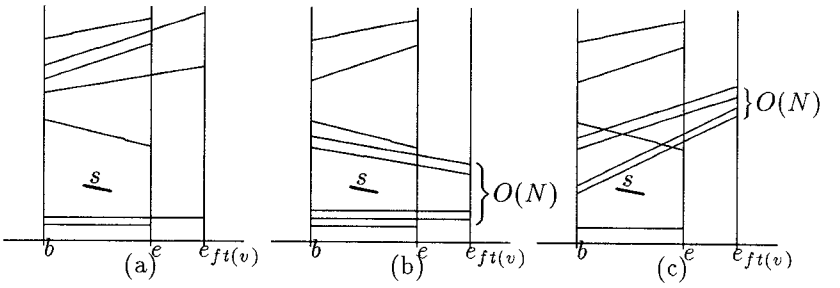

Figure 7: In the situations b,c a number of $D_ft(v)$ stairs intersect trapezoid where s are placed.

$\lceil (e_{ft(v)} - b_{ft(v)})/2 \rceil$). Thus, the storage requirement of this algorithm is $O(N)$.

Lemma 2 $\forall v \in V \ |S'_v| \leq b_v - e_v + |Int(D_v, S'_v)|$.

Proof. The statement directly results from Lemma 1 and the obvious fact that for any set $S \subset S_0$ the number of segment's ends lying in the strip $\langle b_v, e_v \rangle$ is less than $e_v - b_v$.□

Theorem 1 $\sum_{v \in V} |S'_v| \leq 2N \lceil \log N + 1 \rceil + K$.

Proof. The statement directly results from Lemma 2 and the following relation $\sum_v (e_v - b_v) \leq 2N \lceil \log N + 1 \rceil$.□

Theorem 2 $\sum_{v \in RT} |S_v| \leq N \lceil 4 \log N + 5 \rceil + 2K$.

Proof. For all nodes except for the root r the following equality holds $|S_v| \leq |S'_{ft(v)}|$, hence, $\sum_{v \in RT} |S_v| \leq |S_r| + \sum_{v \in RT \setminus r} |S_{ft(v)}| \leq N + 2 \sum_{v \in V} |S'_v| \leq N \lceil 4 log N + 5 \rceil + 2K$.□

Initial sorting and initialization of the sets L_r and I_r can be performed in $O(N \log N)$ time. The running time of the procedure TreeSearch is evaluated by summing up the running time of all its calls. Each external node call adds to this sum a value $O(|L_v| + |Int_{b,e}(L_v)|)$. As for the internal node calls, let's find the time required for the step 10 and for all the other steps separately. The step 10 takes $O(|I_v| \log N)$ time, and all the others run in $O(|S_v| + |Int(D_v, S'_v)|)$ time (see Preliminaries). If we sum up these values, we come to the conclusion that our algorithm runs in time $O(N \log^2 N + K)$. Note that time complexity of the algorithm is $O(N \log N + K)$ if we don't take into account the running time of the 10th step.

Thus, to obtain time optimal algorithm we have to find $Loc(D_v, s)$ at the 10th step within $O(1)$ time bounds.

OPTIMAL ALGORITHM.

For better understanding the ideas of this section see [3, 7, 8, 11].

The main idea is to find $Loc(D_v, s)$ using information about the location of s in the staircases of sons of v. Suppose $Loc(D_v, s)$ is defined, then the value $Loc(D_{ft(v)}, s)$ is not arbitrary and should belong to some interval of $[0..|Q_{ft(v)}|]$. Let us hold lower and upper bounds of the intervals in two arrays SB_v and SE_v. If $Loc(D_v, s) = i$, then $Loc(D_{ft(v)}, s)$ belong to $[SB_v[i], ..., SE_v[i]]$ and the interval $[SB_v[i], ..., SE_v[i]]$ is a minimal one having this property. In the situation illustrated in the figure 7a the presence of these arrays may considerably simplify the search of $Loc(D_{ft(v)}, s)$, but in the situations presented in figures 7b,c the arrays are useless. Therefore it is necessary to change the method of staircase constructing to eliminate the situations given in the figures 7b,c and to make $SE_v[i] - SB_v[i]$ equal to $O(1)$ for any v and i.

Now the staircase D_v is constructed in the following way:

- Split the set L_v in two subsets Or_v and $L_{ls(v)}$ so that
  1. $\forall s \in Or_v$ s spans the strip $\langle b, e \rangle$.
  2. $Int_{b,e}(Or_v) = \emptyset$.
  3. $\forall s \in Or_v \ |Int_{b,e}(Q_{ft(v)}, \{s\})| \leq 1$.
  4. $\forall s \in L_{ls(v)}$ $Or_v \cup \{s\}$ don't satisfy the conditions 1-3.
- Let $Q_{ft(v)} = (s_1, ..., s_k)$ and $n = \lfloor k/4 \rfloor$. Construct the set Inh_v containing those of the segments $s_4, s_8, s_{12}, ..., s_n$ which don't intersect the segments of Or_v within the strip $\langle b, e \rangle$.
- $Q_v = Or_v \cup Inh_v$. Construct staircase $D_v = (Q_v, \langle b, e \rangle)$.

Now the situation Fig.7c is impossible due to the properties of the set Or_v and the situation Fig.7b is impossible due to the properties of the set Inh_v.

Let the segments of Or_v be called the *original segments* of D_v, and their intersections with the strip $\langle b, e \rangle$ be called the *original stairs*. The segments of Inh_v are referred to as the *inherited segments*, and the corresponding stairs as the *inherited stairs*.

Theorem 3 $\forall v\ \forall i,\ SE_v[i] - SB_v[i] \leq 5$.

Proof. The demonstration is illustrated in figure 8. $[A, B]$ and $[C, D]$ are the ith and $i+1$th stairs of the staircase D_v. Obviously, $SE_v[i] - SB_v[i]$ is equal to the number of $D_{ft(v)}$ stairs intersecting the trapezoid $ABCD$. Suppose, this number is greater than five. The sides AB and CD of the trapezoid intersect not more than two stairs of $D_{ft(v)}$ (due to condition 3 during constructing Or_v). Therefore, at least four sequential stairs of $D_{ft(v)}$ don't intersect any stair of D_v that contradicts the procedure of D_v constructing.□

As a result, we may seek $Loc(D_v, s)$ sequentially scanning stairs of D_v beginning from $SB_{v'}[Loc(D_{v'}, s)]$, where v' is the son of v with $Loc(D_{v'}, s)$ already computed. In the procedure below each segment s has additional field bs for keeping the current value of $SB_{v'}[Loc(D_{v'}, s)]$. Thus, the improved procedure TreeSearch is as follows:

procedure NewTreeSearch($L_v, I_v, Q_{ft(v)}, b, e, R_v$).
1. **if** $e - b = 1$ **then**
   SearchInStrip$_{b,e}$(L_v, R_v); exit;
   endif;
2. Construct the staircase D_v;
3. Construct array SB_v;
4. **For** $s \in L_{ls(v)}$ **do**
   find $Loc(D_v, s)$; $s.bs := SB_v[Loc(D_v, s)]$);
   endfor;
5. Find $Int(D_v, L_{ls(v)})$;
6. $c := \lfloor (b+e)/2 \rfloor$;
7. Place segments of I_v
   inner for the strip $\langle b, c\rangle$ into $I_{ls(v)}$,
   inner for the strip $\langle c, e\rangle$ into $I_{rs(v)}$;
8. NewTreeSearch($L_{ls(v)}, I_{ls(v)}, Q_v, b, c, R_{ls(v)}$);
9. **If** p_c is the left endpoint of $s(c)$ **then**
   $L_{rs(v)}$:= insert $s(c)$ into $R_{ls(v)}$
   else
   $L_{rs(v)}$:= delete $s(c)$ from $R_{ls(v)}$;
   endif;
10. NewTreeSearch($L_{rs(v)}, I_{rs(v)}, Q_v, c, e, R_{rs(v)}$);
11. **For** $s \in R_{rs(v)}$ **do**
   find $Loc(D_v, s)$; $s.bs := SB_v[Loc(D_v, s)]$;
   endfor;
12. Find $Int(D_v, R_{rs(v)})$;
13. **For** $s \in I_v$ **do**
   scan the stairs of D_v beginning from $s.bs$
   until we detect $Loc(D_v, s)$;
   $s.bs := SB_v[Loc(D_v, s)]$;
   endfor;
14. Find $Int(D_v, I_v)$ using values
   found in the previous step;
15. $R_v := Merge_e(Q_v, R_{rs(v)})$;
end procedure.

When we find intersection of a staircase with some segments set, we should output intersections with original stairs only.

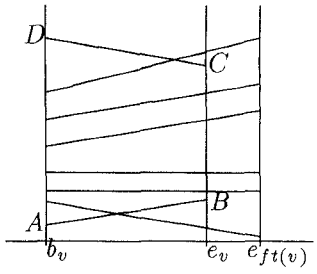

Figure 8: If more than five stairs of $D_{ft(v)}$ intersect trapezoid $ABCD$, then at least fore sequential stairs of $D_{ft(v)}$ have no intersections with stairs of D_v.

Obviously, the steps 4-11 of NewTreeSearch can be done in $O(|S_v| + |Int(D_v, S'_v)|)$ time (see Preliminaries). The procedure NewSplit receiving L_v and $Q_{ft(v)}$ as an input constructs Or_v and $L_{ls(v)}$ in $O(|L_v| + |Q_{ft(v)}|)$ time.

procedure NewSplit$_{b,e}$($L_v, Q_{ft(v)}, Q_v, L_{ls(v)}$).
{Let $L_v = (s_1, ..., s_k)$, $Q_{ft(v)} = (s'_1, ..., s'_n)$}
  $L_{ls(v)} := \emptyset$; $Or_v := \emptyset$; $i := 1$;
  For $j = 1, ..., k$ **do**;
    While $s'_i <_b s_j$ *and* $i \leq n$ **do**
      $i := i + 1$;
    endwhile;
    If s_j doesn't intersect s'_{i-2}, s'_{i+1} and
    the last segment of Or_v within the strip $\langle b, e\rangle$ and
    span it **then**
      add s_j to the end of Or_v
    else
      add s_j to the end of $L_{ls(v)}$;
    endif;
  endfor;
end procedure.

The following procedure builds set Q_v and array SB_v using Or_v and $Q_{ft(v)}$ in time linear in the input.

procedure BuildQSB$_{b,e}$($Or_v, Q_{ft(v)}, Q_v, SB_v$).
{Let $Or_v = (s_1, ..., s_k)$, $Q_{ft(v)} = (s'_1, ..., s'_n)$}
  $SB_v[0] := 0$; $Q_v := \emptyset$; $i := 1$; $l := 1$;
  For $j = 1, ..., k$ **do**
    While $s'_i <_b s_j$ *and* $i \leq n$ **do**
      If i *mod* $4 = 0$ *and* s'_i doesn't intersect s_j and
        s_{j-1} within the strip $\langle b, e\rangle$ **then**
        add s'_i to the end of Q_v;
        $l := l + 1$; $SB_v[l] := i$;
      endif;
      i:=i+1;
    endwhile;
    Add s_j to the end of Q_v; $l := l + 1$;
    If s_j intersects s'_{i-1} within $\langle b, e\rangle$ **then**
      $SB_v[l] := i - 2$

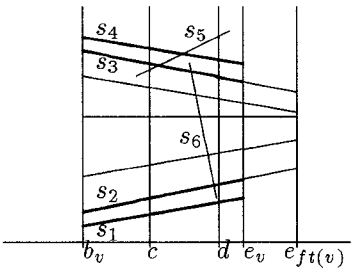

Figure 9: $Or_v = (s_1, s_4)$, $Inh_v = (s_2, s_3)$, $S = \{s_5, s_6\}$, $Int^{or}_{c,d}(D_v, S) = \{\{s_4, s_5\}\}$, $Int^{sn}_{c,d}(D_v, S) = \{\{s_3, s_5\}\}$, $Int^{ml}_{c,d}(D_v, S) = \{\{s_2, s_6\}, \{s_3, s_6\}\}$.

```
    else
      SB_v[l] := i − 1;
    endif;
  endfor;
end procedure.
```

It follows from this observation that

Proposition 2 *All work executed by NewTreeSearch in internal node v can be done in $O(|Q_{ft(v)}| + |S_v| + |Int(D_v, S'_v)|)$ time.*

Given a staircase $D = (Q, \langle b, e\rangle)$, a strip $\langle c, d\rangle \subset \langle b, e\rangle$ and a segment s intersecting a segment $s' \in Q$ within the strip $\langle c, d\rangle$, then the pair $\{s, s'\}$ is referred to as

- *original intersection* of the segment s with the staircase D within the strip $\langle c, d\rangle$ if s' is an original segment of D;
- *multiple inherited intersection* of the segment s with the staircase D within the strip $\langle c, d\rangle$ if s' is an inherited segment of D, and the segment s intersects at least one more inherited segment of D within the strip $\langle c, d\rangle$;
- *single inherited intersection* of the segment s with the staircase D within the strip $\langle c, d\rangle$ in other cases.

Let $Int^{or}_{c,d}(D, S)$, $Int^{ml}_{c,d}(D, S)$ and $Int^{sn}_{c,d}(D, S)$ denote the sets of the original, multiple inherited and single inherited intersections of the segments of a set S with a staircase D within a strip $\langle c, d\rangle$, respectively (see Fig.9).

Lemma 3 $|S'_v| \leq e_v - b_v + |Int^{or}_{b_v,e_v}(D_v, S'_v)| + |Int^{or}_{b_v,e_v}(D_{ft(v)}, S'_v)| + |Int^{ml}_{b_v,e_v}(D_{ft(v)}, S'_v)|$.

Proof. Let us suppose that for some node v the above stated property is not true. Then set S'_v must contain a segment s spanning the strip $\langle b_v, e_v\rangle$, having no original intersections with the staircases D_v and $D_{ft(v)}$ and intersecting at most one inherited stair of $D_{ft(v)}$ (inside the strip). This contradicts the condition 4 during D_v construction.□

Lemma 4 $|Int^{ml}_{c,d}(D_v, \{s\})| \leq \frac{1}{2}(|Int^{or}_{c,d}(D_{ft(v)}, \{s\})| + |Int^{ml}_{c,d}(D_{ft(v)}, \{s\})|)$.

Proof. Let $|Int^{ml}_{c,d}(D_v, \{s\})| = n$, $n \geq 2$ by the definition. Since Q_v could inherit each 4th segment of $Q_{ft(v)}$, $|Int_{c,d}(Q_{ft(v)}, \{s\})| \geq 4n - 3$ (see Fig.9). But $|Int_{c,d}(Q_{ft(v)}, \{s\})| = |Int^{or}_{c,d}(D_{ft(v)}, \{s\})| + |Int^{sn}_{c,d}(D_{ft(v)}, \{s\})| + |Int^{ml}_{c,d}(D_{ft(v)}, \{s\})|$ and $|Int^{sn}_{c,d}(D_{ft(v)}, \{s\})| \leq 1$. As a result, we have $|Int^{or}_{c,d}(D_{ft(v)}, \{s\})| + |Int^{ml}_{c,d}(D_{ft(v)}, \{s\})| \geq 4n - 4 \geq 2|Int^{ml}_{c,d}(D_v, \{s\})|$.□

Let $H(i)$ be the set of the recursion tree nodes with depth i. Let h_m be the height of the tree minus one and $anc(v, i)$ be the ancestor of the node v having depth i. We denote the sums $\sum_{v\in H(i)} |Int^{or}_{b_v,e_v}(D_{anc(v,j)}, S'_v)|$ and $\sum_{v\in H(i)} |Int^{ml}_{b_v,e_v}(D_{anc(v,j)}, S'_v)|$ as $M_{or}(i, j)$ and $M_{ml}(i, j)$, respectively. According to Lemma 4 $M_{ml}(i, j) \leq 1/2(M_{or}(i, j-1) + M_{ml}(i, j-1))$. Evidently, we have $M_{ml}(i, 1) = 0$.

Lemma 5 $\sum_{i=j+1}^{h_m} M_{or}(i, i-j) \leq K \ \forall j \in [0, .\,, h_m - 1]$.

Proof. The sum presents the intersecting pairs (possibly not all) counted in some way. To prove the statement it is sufficient to show that each pair is counted at most once. $\sum_{i=j+1}^{h_m} M_{or}(i, i-j) = \sum_{i=j+1}^{h_m} \sum_{v\in H(i)} |Int^{or}_{b_v,e_v}(D_{anc(v,i-j)}, S'_v)|$. Assume that the pair $\{s1, s2\}$ intersecting in a point p is counted twice and belongs to $Int^{or}_{b_v,e_v}(D_{anc(v,h_v-j)}, S'_v)$ and $Int^{or}_{b_w,e_w}(D_{anc(w,h_w-j)}, S'_w)$ simultaneously, where h_v and h_w are the depth of nodes v and w, respectively. Then, the point p should belong to the strips $\langle b_v, e_v\rangle$ and $\langle b_w, e_w\rangle$ simultaneously. Therefore, either the node v is an ancestor of w or the node w is an ancestor of v. W.l.o.g. assume the latter is the case, then, the node $anc(w, h_w - j)$ is an ancestor of the node $anc(v, h_v - j)$. Since $Or_{anc(v,h_v-j)} \cap Or_{anc(w,h_w-j)} = \emptyset$, the pair $\{s1, s2\}$ cannot belong to $Int^{or}_{b_v,e_v}(D_{anc(v,h_v-j)}, S'_v)$ and $Int^{or}_{b_w,e_w}(D_{anc(w,h_w-j)}, S'_w)$ simultaneously.□

Lemma 6 $\sum_{i=j+1}^{h_m} M_{ml}(i, i-j) \leq K \ \forall j \in [0, .., h_m - 1]$.

Proof. $\sum_{i=j+1}^{h_m} M_{ml}(i, i-j) \leq \frac{1}{2}\sum_{i=j+2}^{h_m}(M_{or}(i, i-(j+1)) + M_{ml}(i, i-(j+2))) \leq \frac{1}{2}\sum_{i=j+2}^{h_m} M_{or}(i, i-(j+1)) + \frac{1}{4}\sum_{i=j+3}^{h_m}(M_{or}(i, i-(j+2)) + M_{ml}(i, i-(j+2))) \leq \ldots \leq \sum_{n=1}^{h_m-j} \frac{1}{2}^n \sum_{i=j+n+1}^{h_m} M_{or}(i, i-(j+n)) \leq \sum_{n=1}^{h_m-j} \frac{1}{2}^n K \leq K$.□

Theorem 4 $\sum_{v\in V} |S'_v| \leq 2N\lceil \log N + 1\rceil + 3K$.

Proof. $\sum_{v\in V} |S'_v| \leq \sum_{v\in V}(b_v - e_v + |Int^{or}_{b_v,e_v}(D_v, S'_v)| + |Int^{or}_{b_v,e_v}(D_{ft(v)}, S'_v)| + |Int^{ml}_{b_v,e_v}(D_{ft(v)}, S'_v)|) \leq 2N\lceil logN + 1\rceil + \sum_{i=1}^{h_m} M_{or}(i, i) + \sum_{i=2}^{h_m} M_{or}(i, i-1) + \sum_{i=2}^{h_m} M_{ml}(i, i-1) \leq 2N\lceil logN + 1\rceil + 3K$.□

Theorem 5 $\sum_{v\in V}|Int(D_v,S'_v)| \le 2N\lceil\log N+1\rceil + 5K$.

Proof. $|Int(D_v,S'_v)| = |Int^{or}_{b_v,e_v}(D_v,S'_v)|+|Int^{sn}_{b_v,e_v}(D_v,S'_v)|+|Int^{ml}_{b_v,e_v}(D_v,S'_v)|$. According to the definition of the single inherited intersection $|Int^{sn}_{b_v,e_v}(D_v,S'_v)| \le |S'_v|$, hence, $\sum_{v\in V}|Int(D_v,S'_v)| \le \sum_{v\in V}(|Int^{or}_{b_v,e_v}(D_v,S'_v)|+|S'_v|+|Int^{ml}_{b_v,e_v}(D_v,S'_v)|)$. To prove the theorem it is sufficient to apply Lemmas 5, 6 and theorem 4 to the last sum.□

Theorem 6 $\sum_{v\in V}|S_v| \le N\lceil 4\log N+5\rceil + 6K$.

Proof. For all nodes except for the root r $|S_v| \le |S'_{ft(v)}|$, hence, $\sum_{v\in RT}|S_v| \le |S_r| + \sum_{v\in RT\setminus r}|S_{ft(v)}| \le N + 2\sum_{v\in V}|S'_v| \le N\lceil 4logN+5\rceil + 6K$.□

As $Or_v \in S_v$, hence, $\sum_{v\in V}|Or_v| \le N\lceil 4logN+5\rceil + 6K$ too.

Theorem 7 $\sum_{v\in V}|Q_v| \le 2N\lceil 4\log N+5\rceil + 12K$.

Proof. Let us designate the sums $\sum_{v\in H(i)}|Or_v|$ and $\sum_{v\in H(i)}|Q_v|$ through $F(i)$ and $G(i)$, respectively. According to construction $G(1) = F(1)$. Each internal node has two sons, each son inherits not more than a quarter of the father stairs, hence, $G(i+1) \le F(i+1) + \frac{1}{2}G(i)$. Therefore, $\sum_{i=1}^{h_m}G(i) \le 2\sum_{i=1}^{h_m}F(i)$.□

Theorem 8 *The procedure IntersectingPairs(S_0) with new version of the procedure TreeSearch runs in time* $O(N\log N+K)$.

Proof. Results from the proposition 2 and the theorems 5, 6, 7.

Theorem 9 *The memory requirement of resulting algorithm is* $O(N)$.

Proof. Evidently the active call occupies $O(N)$ memory each of its "ancestors" retains $O(|I_v|+|Q_v|)$ memory, other structures use $O(N)$ memory. It is sufficient to prove that for any path pt from the root of recursion tree to some of its node $\sum_{v\in pt}(|I_v|+|Q_v|) = O(N)$. We have $|I_v| \le e_v - b_v$, $\sum_{v\in pt}(e_v-b_v) \le N$, hence $\sum_{v\in pt}|I_v| \le N$. If $v \in pt$ and $w \in pt$ then $Or_v \cap Or_w = \emptyset$, hence $\sum_{v\in pt}|Or_v| \le N$. According to construction $|Q_v| \le |Or_v| + \frac{1}{4}|Q_{ft(v)}|$, hence $\sum_{v\in pt}|Q_v| \le \frac{4}{3}\sum_{v\in pt}|Or_v| \le \frac{4}{3}N$.□

CONCLUSIONS

In this work we have proposed two algorithms for finding all K intersecting pairs among N segments in the plane. They have $O(N\log^2 N+K)$- and $O(N\log N+K)$-time complexity and require $O(N)$ space. The algorithms can operate not only with line segments. Segments may be any connected 2D objects which have at the most one intersection with any vertical line and permit the following operations to be performed in $O(1)$ time

- Given a segment and a vertical line. Find their intersection point.
- Given a vertical strip and a pair of segments. Determine if the segments intersect within the strip.

In this case the algorithms have time and space complexity stated above, the only difference is that K denote the number of intersection point (in assumption that all intersection points are different).

References

[1] Ben-Or, M. Lower bounds for algebraic computation trees. In *Proceedings of the 15th Annual ACM Symposium Theory of Computing.* (1983), 80–86.

[2] Bentley, J.L., and Ottman, T. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput. C-28* (1979), 643-647.

[3] Chazelle, B. Filtering search: A new approach to query-answering. *SIAM J. Comput. Vol.15* (1986), 703-725.

[4] Chazelle, B. Reporting and counting segment intersections. *J. Comput. Sys. Sci. 32* (1986), 156-182.

[5] Chazelle, B., and Edelsbrunner, H. An optimal algorithm for intersecting line segments in the plane. *Journal of the ACM, Vol. 39* (1992), 1-54.

[6] Clarkson, K.L., and Shor, P.W. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry, Vol. 4* (1989), 387–421.

[7] Edelsbrunner, H., Guibas L.J., Stolfi J. Optimal point location in monotone subdivision. *SIAM J. Comput. Vol.15(2)* (1986), 317-340.

[8] Lueker, G. A data structure for orthogonal range queries. In *Proceedings of the 19th annual IEEE Symposium on Foundations of Computer Science.*(1979), 28–34.

[9] Mulmuley, K. A fast planar partition algorithm. In *Proceedings of the 29th annual IEEE Symposium on Foundations of Computer Science.*(1988), 580–589.

[10] Preparata, F.P., and Shamos, M.I. *Computational geometry: An introduction.* Springer-Verlag, New York, 1985.

[11] Willard, D. Polygon retrieval. *SIAM J. Comput. Vol.11* (1982), 149–165.