Design and Analysis of Algorithms I

Divide and Conquer

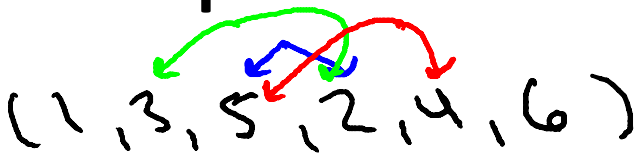
Counting Inversions I

The Problem

Input : array A containing the numbers 1,2,3,..,n in some arbitrary order

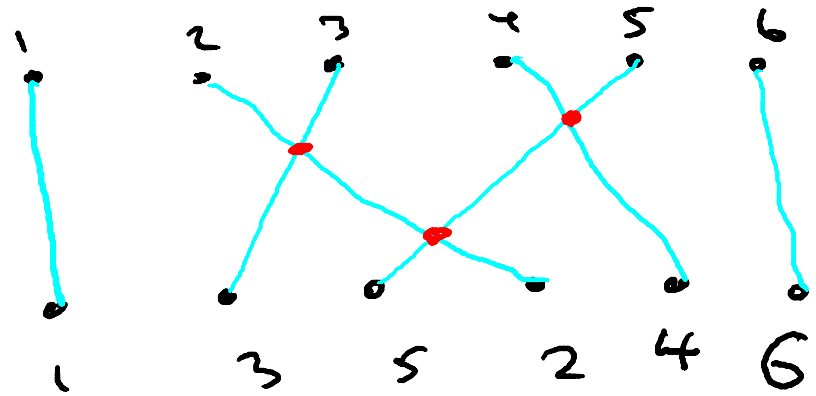
Output : number of inversions = number of pairs (i,j) of array indices with i<j and A[i] > A[j]

Examples and Motivation

Example 

Inversions :

(3,2), (5,2), (5,4)



Motivation : numerical similarity measure between two ranked lists eg: for collaborative filtering

Ter
ver

What is the largest-possible number of inversions that a 6-element array can have?

- 15
- 21
- 36
- 64

In general, $\binom{n}{2} = n(n-1)/2$

High-Level Approach

Brute-force : $\theta(n^2)$ time

Can we do better ? Yes!

KEY IDEA # 1 : Divide + Conquer

Call an inversion (i,j) [with i<j]

Left : if i,j < n/2

Right : if i,j > n/2

Split : if i<=n/2 < j

Note : can compute these recursively

need separate subroutine for these

High-Level Algorithm

Count (array A, length n)

if n=1, return 0

else

X = Count (1st half of A, n/2)

Y = Count (2nd half of A, n/2_

Z = CountSplitInv(A,n) ← CURRENTLY UNIMPLEMENTED

return x+y+z

Goal : implement CountSplitInv in linear (O(n)) time then count will run in O(nlog(n)) time [just like Merge Sort]

Tim Roughgarden

Design and Analysis of Algorithms I

Divide and Conquer

Counting Inversions II

Piggybacking on Merge Sort

KEY IDEA # 2 : have recursive calls both count inversions and sort.
[i.e. , piggy back on Merge Sort]

Motivation : Merge subroutine naturally uncovers split inversions [as we'll see]

High-Level Algorithm (revised)

Sort-and-Count (array A, length n)

if n=1, return 0

else

Sorted version of 1st half → (B,X) = Sort-and-Count(1st half of A, n/2)

Sorted version of 2nd half → (C,Y) = Sort-and-Count(2nd half of A, n/2)

Sorted version of A → (D,Z) = CountSplitInv(A,n) ← CURRENTLY UNIMPLEMENTED

return X+Y+Z

Goal : implement CountSplitInv in linear (O(n)) time

=> then Count will run in O(nlog(n)) time [just like Merge Sort]

Pseudocode for Merge:

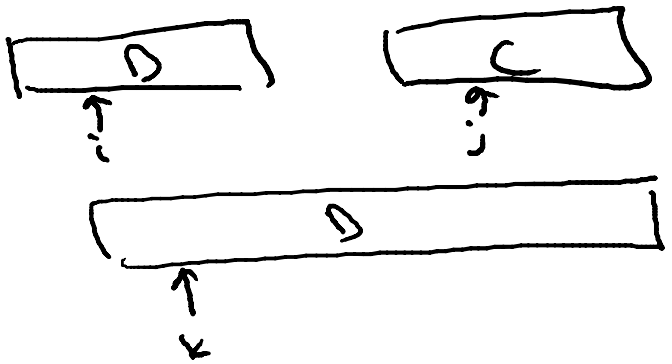
D = output [length = n]

B = 1st sorted array [n/2]

C = 2nd sorted array [n/2]

i = 1

j = 1



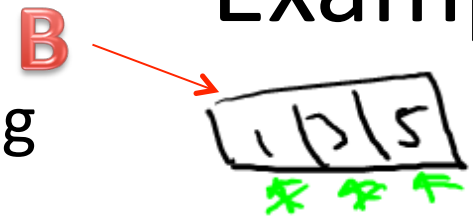

```
for k = 1 to n
        if B(i) < C(j)
                D(k) = B(i)
                i++
        else [C(j) < B(i)]
                D(k) = C(j)
                j++
end
```

(ignores end cases)

Suppose the input array A has no split inversions. What is the relationship between the sorted subarrays B and C?

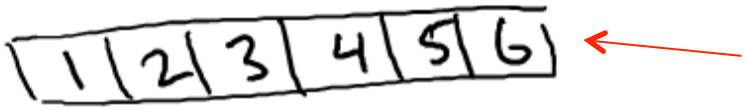
- B has the smallest element of A, C the second-smallest, B, the third-smallest, and so on.
- All elements of B are less than all elements of C.
- All elements of B are greater than all elements of C.
- There is not enough information to answer this question.

Example

Consider merging 

and 

.

Output : 

⇒When 2 copied to output, discover the split inversions (3,2) and (5,2)

⇒ when 4 copied to output, discover (5,4)

General Claim

Claim the split inversions involving an element y of the 2nnd array C are precisely the numbers left in the 1st array B when y is copied to the output D.

Proof : Let x be an element of the 1st array B.

1. if x copied to output D before y, then x < y

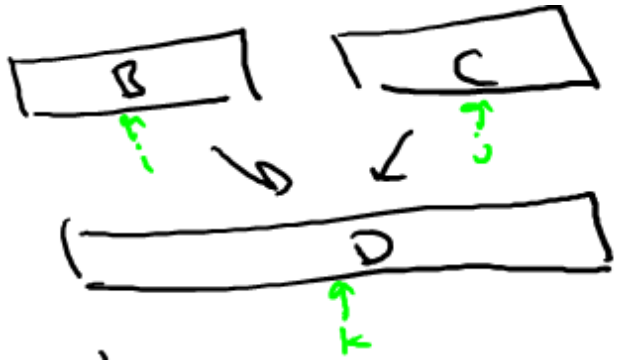
⇒ no inversions involving x and y

2. If y copied to output D before x, then y < x

=> X and y are a (split) inversion. Q.E.D

Merge_and_CountSplitInv

-- while merging the two sorted subarrays, keep running total of number of split inversions



-- when element of 2nd array C gets copied to output D, increment total by number of elements remaining in 1st array B

merge running total

Run time of subroutine : O(n) + O(n) = O(n)

=> Sort_and_Count runs in O(nlog(n)) time [just like Merge Sort]

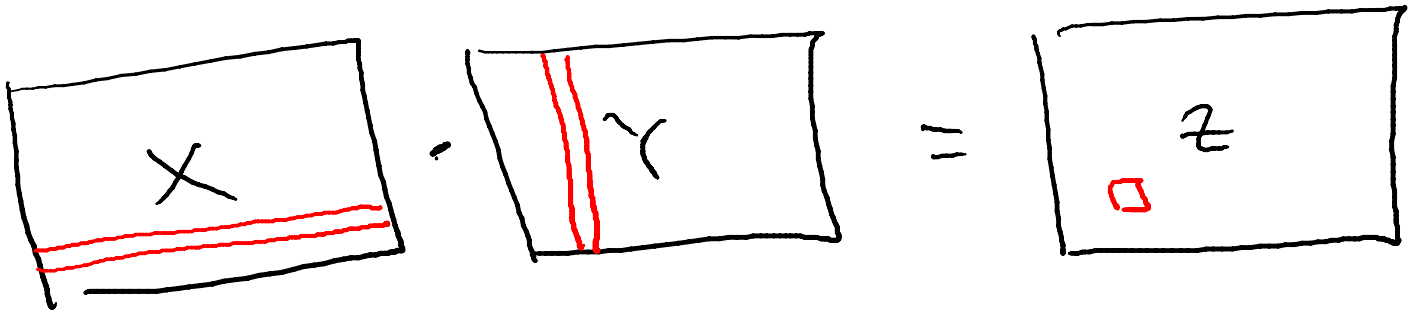
Tim Roughgarden

Design and Analysis of Algorithms I

Divide and Conquer

Matrix Multiplication

Matrix Multiplication



(all n X n matrices)

Where z_{ij} = (ith row of X) . (jth column of Y)

$$= \sum_{k=1}^{n} X_{ik} \cdot Y_{kj}$$

Note : input size $= \theta(n^2)$

Tim Roughgarden

Example (n=2)

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{pmatrix}$$

$$z_{ij} = \sum_{k=1}^{n} X_{ik} \cdot Y_{kj}$$

$\theta(n)$

Ter
ver

What is the asymptotic running time of the straightforward iterative algorithm for matrix multiplication?

- ○ $\theta(n \log n)$
- ○ $\theta(n^2)$
- ○ $\theta(n^3)$
- ○ $\theta(n^4)$

The Divide and Conquer Paradigm

1. DIVIDE into smaller subproblems

2. CONQUER subproblems recursively.

3. COMBINE solutions of subproblems into one for the original problem.

Applying Divide and Conquer

Idea :

Write $X = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$ and $Y = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$

[where A through H are all n/2 by n/2 matrices]

Then : (you check)

$$X \cdot Y = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

Recursive Algorithm #1

Step 1 : recursively compute the 8 necessary products.

Step 2 : do the necessary additions $(\theta(n^2)\ time)$

Fact : runtime is $\theta(n^3)$ [follows from the master method]

Strassen's Algorithm (1969)

Step 1 : recursively compute only 7 (cleverly chosen) products

Step 2 : do the necessary (clever) additions + subtractions (still $\theta(n^2)$ time)

Fact : better than cubic time!

[see Master Method lecture]

$X = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$ $Y = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$

The Details

The Seven Products : $P_1 = A(F-H)$, $P_2 = (A+B)H$, $P_3 = (C+D)E$, $P_4 = D(G-E)$, $P_5 = (A+D)(E+H)$, $P_6 = (B-D)(G+H)$, $P_7 = (A-C)(E+F)$

Claim : $X \cdot Y = \begin{pmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{pmatrix} = \begin{pmatrix} P_5+P_4-P_2+P_6 & P_1+P_2 \\ P_3+P_4 & P_1+P_5-P_3-P_7 \end{pmatrix}$

Proof: $AE + \cancel{AH} + \cancel{DE} + \cancel{DH} + \cancel{DG} - \cancel{DE} - \cancel{AH} - \cancel{BH} + \cancel{BG} + \cancel{BH} - \cancel{DG} - \cancel{DH} = AE + BG$ Q.E.D

Question : where did this come from ? (remains open!)

Tim Roughgarden

Divide and Conquer

Closest Pair I

Design and Analysis of Algorithms I

The Closest Pair Problem

Input : a set $P = \{p_1, ..., p_n\}$ of n points in the plane R^2.

Notation : $d(p_i, p_j)$ = Euclidean distance
So if $p_i = (x_i, y_i)$ and $p_j = (x_j, y_j)$

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Output : a pair $p*, q* \in P$ of distinct points that minimize d(p,q) over p,q in the set P

Tim Roughgarden

Initial Observations

Assumption : (for convenience) all points have distinct x-coordinates, distinct y-coordinates.

Brute-force search : takes $\theta(n^2)$ time.

1-D Version of Closest Pair :

1. Sort points (O(nlog(n)) time)
2. Return closest pair of adjacent points (O(n) time)

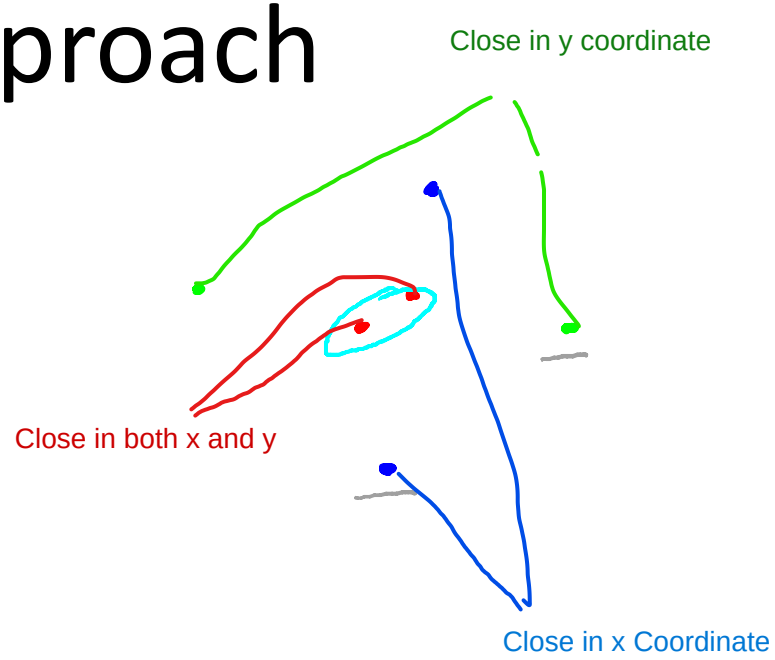
Goal : O(nlog(n)) time algorithm for 2-D version.

High-Level Approach

1. Make copies of points sorted by x-coordinate (P_x) and by y-coordinate (P_y)
   [O(nlog(n)) time]

   (but this is not enough!)

2. Use Divide+Conquer



The Divide and Conquer Paradigm

1. DIVIDE into smaller subproblems.

2. CONQUER subproblems recursively.

3. COMBINE solutions of subproblems into one for the original problem.

ClosestPair(P_x, P_y)

BASE CASE OMITTED

1. Let Q = left half of P, R = right half of P. Form

   Q_x, Q_y, R_x, R_y [takes O(n) time]

2. (p_1,q_1) = ClosestPair(Q_x,Q_y)
3. (p_2,q_2) = ClosestPair(R_x,R_y)
4. (p_3,q_3) = ClosestSplitPair(P_x,P_y)
5. Return best of (p_1,q_1), (p_2,q_2), (p_3,q_3)

Suppose we can correctly implement the ClosestSplitPair subroutine in O(n) time. What will be the overall running time of the Closest Pair algorithm ? (Choose the smallest upper bound that applies.)

- $O(n)$
- $O(n \log n)$
- $O(n (\log n)^2)$
- $O(n^2)$

Key Idea : only need to bother computing the closest split pair in "unlucky case" where its distance is less than $d(p_1,q_1)$ and $d(p_2,q_2)$.

Result of 1st recursive call

Result of 2nd recursive call

ClosestPair(P_x, P_y)

1. Let Q = left half of P, R = right half of P. Form

   Q_x, Q_y, R_x, R_y [takes O(n) time]

2. (p_1,q_1) = ClosestPair(Q_x,Q_y)
3. (p_2,q_2) = ClosestPair(R_x,R_y)
4. Let $\delta = min\{d(p_1, q_1), d(p_2, q_2)\}$
5. (p_3,q_3) = ClosestSplitPair(P_x,P_y,δ)
6. Return best of (p_1,q_1), (p_2,q_2), (p_3,q_3)

BASE CASE OMITTED

WILL DESCRIBE NEXT

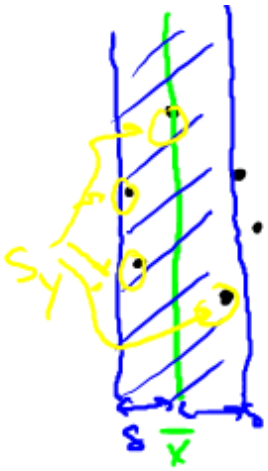
Requirements

1. O(n) time
2. Correct whenever closest pair of P is a split pair

ClosestSplitPair(P_x, P_y, δ)

Let $\bar{x}$ = biggest x-coordinate in left of P. (O(1) time)

Let S_y = points of P with x-coordinate in
Sorted by y-coordinate (O(n) time)



Initialize best = δ , best pair = NULL
For i = 1 to $|S_y| - 7$
  For j = 1 to 7
    Let p,q = i^{th} , $(i+j)^{th}$ points of S_y
    If d(p,q) < best
      best pair = (p,q), best = d(p,q)

O(1) time

O(n) time

At end return
best pair

Tim Roughgarden

Correctness Claim

$min\{d(p_1, q_1), d(p_2, q_2)\}$

Claim : Let $p \in Q, q \in R$ be a split pair with d(p,q) < δ

Then: (A) p and q are members of S_y

(B) p and q are at most 7 positions apart in S_y.

Corollary1 : If the closest pair of P is a split pair, then the ClosestSplitPair finds it.

Corollary2 ClosestPair is correct, and runs in O(nlog(n)) time.

Assuming claim is true!

Divide and Conquer

Closest Pair II

Design and Analysis of Algorithms I

Correctness Claim

$min\{d(p_1, q_1), d(p_2, q_2)\}$

Claim : Let $p \in Q, q \in R$ be a split pair with d(p,q) < δ

Then: (A) p and q are members of S_y

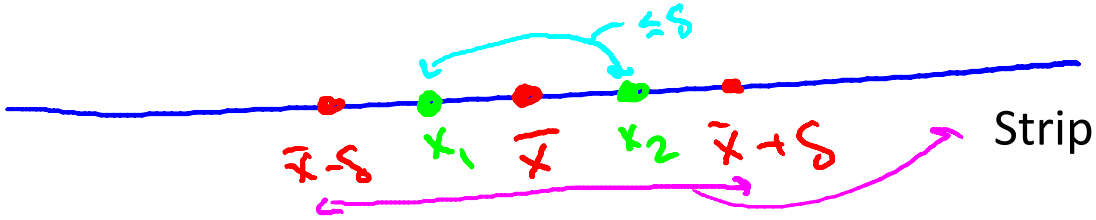
(B) p and q are at most 7 positions apart in S_y.

Proof of Correctness Claim (A)

Let $p = (x_1, y_1) \in Q,\ q = (x_2, y_2) \in R,\ d(p,q) \leq \delta$

Note : Since $d(p,q) \leq \delta,\ |x_1 - x_2| \leq \delta\ and\ |y_1 - y_2| \leq \delta$

Proof of (A) [p and q are members of S_y i.e. $x_1, x_2 \in [\bar{x} - \delta, \bar{x} + \delta]$]



Strip that defines S_y

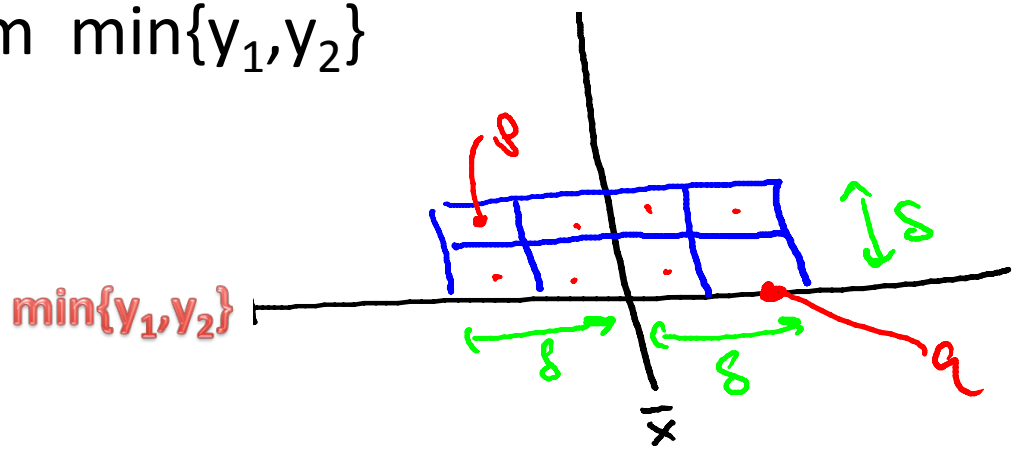
Note : $p \in Q => x_1 \leq \bar{x}\ and\ q \in R => x_2 \geq \bar{x}.$

=> $x_1, x_2 \in [\bar{x} - \delta, \bar{x} + \delta]$

Proof of Correctness Claim (B)

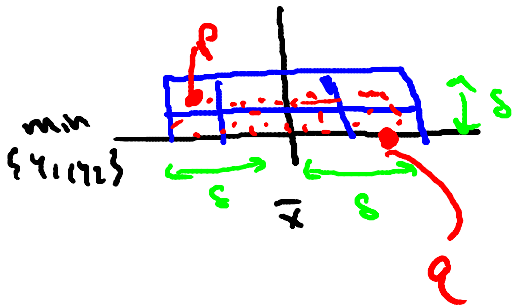
(B) : p = (x_1,y_1) and q = (x_2,y_2) are at most 7 positions apart in S_y

Key Picture : draw $\delta/2 \times \delta/2$ boxes with center $\bar{x}$ and bottom $\min\{y_1,y_2\}$



Proof of Correctness Claim (B)

Lemma 1 : all points of S_y with y-coordinate between those of p and q, inclusive, lie in one of these 8 boxes.

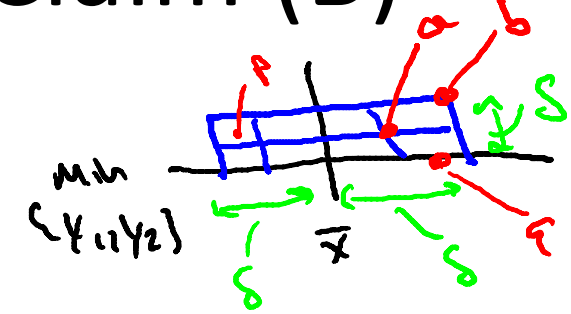


Proof : First, recall y-coordinates of p,q differ by $< \delta$

Second, by definition of S_y, all have x-coordinates between $\bar{x} - \delta \ \ and \ \ \bar{x} + \delta$

Q.E.D

Proof of Correctness Claim (B)



Lemma 2 : At most one point of P in each box.

Proof : by contradiction

Suppose a,b lie in the same box. Then :

I. a,b are either both in Q or both in R

II. $d(a,b) \leq \frac{\delta}{2} \cdot \sqrt{2} \leq \delta$

But (i) and (ii) contradict the definition of δ (as smallest distance between pairs of points in Q or in R)

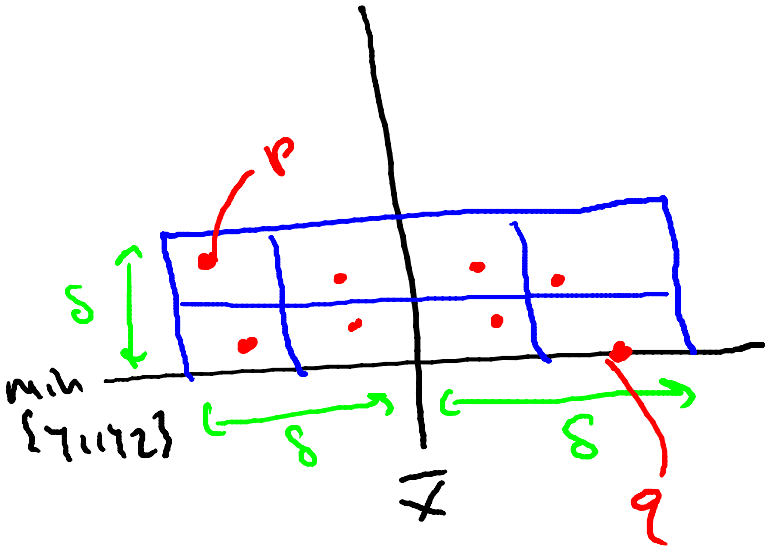
Q.E.D

Tim Roughgarden

Final Wrap-Up

Lemmas 1 and 2 => at most 8 points in this picture (including p and q)

=> Positions of p,q in S_y differ by at most 7



Q.E.D