# COMPSCI 311: Introduction to Algorithms

## Lecture 12: Counting Inversions. Closest Pair of Points

Marius Minea

University of Massachusetts Amherst

slides credit: Dan Sheldon

4 March 2019

## Master Theorem

Consider the general recurrence:

$$T(n) \le aT\left(\frac{n}{b}\right) + cn^d$$

This solves to:

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } \log_b a < d \\ \Theta(n^d \log n) & \text{if } \log_b a = d \\ \Theta(n^{\log_b a}) & \text{if } \log_b a > d \end{cases}$$

Intuition: work at each level of the recursion tree is (1) decreasing exponentially, (2) staying the same, (3) increasing exponentially.

## Clicker Question 1: Master Theorem

Consider the general recurrence:

$$T(n) \le aT\left(\frac{n}{b}\right) + cn^d$$

**Clicker**. How much work is done outside recursion at the root of the recursion tree?

A. $\Theta(n^a)$

B. $\Theta(n^b)$

C. $\Theta(n^d)$

D. None of the above

The work done at the root is $\Theta(n^d)$

## Clicker Question 2: Master Theorem

Consider the general recurrence:

$$T(n) \le aT\left(\frac{n}{b}\right) + cn^d$$

**Clicker**. How many leaves are in the recursion tree?

A. $\Theta(a^{\log_b n})$

B. $\Theta(b^{\log_a n})$

C. $\Theta(n^{\log_b a})$

D. Both a and c

The work done at the leaves (base cases) is $\Theta(n^{\log_b a})$

## Counting Inversions: Motivation

$n$ objects, ranked in linear order by different sources

| | A | B | C | D | E |
|---|---|---|---|---|---|
| RankList1 | 3 | 4 | 2 | 1 | 5 |
| RankList2 | 4 | 2 | 1 | 3 | 5 |

How similar are these rankings?

Applications:

- Recommendation systems (collaborative filtering)
- Stability / sensitivity of web ranking functions
- Meta-search tools: compare & aggregate search engines
- Measure "sortedness" of an array

## Similarity Metric: Number of Inversions

| | A | B | C | D | E |
|---|---|---|---|---|---|
| RankList1 | 3 | 4 | 2 | 1 | 5 |
| RankList2 | 4 | 2 | 1 | 3 | 5 |

A pair $\{X, Y\} \subseteq \{A, B, C, D, E\}$ has an **inversion** between the two rankings if $rank_1(X) < rank_1(Y)$ but $rank_2(X) > rank_2(Y)$ or vice-versa.

Alternate view: Take Rank1 as reference point and renumber objects based on that rank: D = 1, C = 2, A = 3, B = 4, E = 5.

Rewrite Rank2 as $R'$, ranking each of the new IDs

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| R' | 3 | 1 | 4 | 2 | 5 |

Say $i$ and $j$ are **inverted** if $i < j$ but $R'(i) > R'(j)$

Is this the same definition? (Has the number of inversions changed?)

## Clicker Question 3: Inversions

| | A | B | C | D | E |
|---|---|---|---|---|---|
| RankList1 | 3 | 4 | 2 | 1 | 5 |
| RankList2 | 4 | 2 | 1 | 3 | 5 |

Rename D = 1, C = 2, A = 3, B = 4, E = 5, rewrite:

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| R' | 3 | 1 | 4 | 2 | 5 |

Does the number of inversions change?

A) Yes, it has changed

B) No, it never changes

C) It has not changed here, but changes in other cases

## Trying Divide-and-Conquer

- Divide: into two array halves, A and B
- Conquer: recursively count inversions in each list
- Combine: add the two counts
  plus inversions between halves: $(a, b)$ with $a \in A, b \in B$

Example:

| 2 | 4 | 7 | 9 | 3 | 10 | 1 | 5 | 8 | 6 |

| 2 | 4 | 7 | 9 | 3 | | 10 | 1 | 5 | 8 | 6 |

3 (4-3, 7-3, 9-3) 5 (10 - all, 8-6)

Inversions between halves: ???

## Clicker Question 4

What do we need to combine the subproblems?

To count inversions between the array halves, we'd need to:

A) Know min and max values in both halves

B) Know min and max values in both halves and their positions

C) Neither of the above is enough

## Counting inversions between halves

- Need to compare *each* element on left with *each* on right
  brute force: $O(n^2)$
- Avoid redundant work: know something about relative order
- What if elements in each half were **sorted** ?

| 2 | 3 | 4 | 7 | 9 | | 1 | 5 | 6 | 8 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | ↑ | | | | |
| | | | | | | 7 : 3 inversions | | | | |

| 2 | 3 | 4 | 7 | 9 | | 1 | 5 | 6 | 8 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 3 | 4 | inversions with $B$ per element of $A$ | | | | | |

- How to count these inversions? **binary search**
  Find smallest index $i$ so $a < b_i$. Then $a$ has $i$ inversions with $B$.
  If no such index ($a > b_i$ for all $i$), $|B|$ inversions.

## Combining Counting and Sorting

- Inductive assumption: halves are sorted
- Must reestablish when combining: MERGE-AND-COUNT
- Merge as usual (left-to-right), compare $a_i$ to $b_j$
  - all inversions so far accounted for
  - if $a_i < b_j$, pick $a_i$, no extra inversions

    | | | 4 | 7 | 9 | | 5 | 6 | 8 | 10 |
    |---|---|---|---|---|---|---|---|---|---|

  - if $a_i > b_j$, pick $b_j$, inverted with remaining elements in $A$

    | | | | 7 | 9 | | 5 | 6 | 8 | 10 |
    |---|---|---|---|---|---|---|---|---|---|

## Counting Inversions: Algorithm

SORT-AND-COUNT($L$)

**if** $L$ has one element **then return** $(0, L)$
**end if**
Divide list $L$ into halves $A$ and $B$ ▷ $T(n/2)$
$(c_A, A) =$ SORT-AND-COUNT($A$) ▷ $T(n/2)$
$(c_B, B) =$ SORT-AND-COUNT($B$) ▷ $\Theta(n)$
$(c_{AB}, L) =$ MERGE-AND-COUNT($A, B$)
**return** $(c_A + c_B + c_{AB}, L)$

- Complexity? $\Theta(n \log n)$
- Take-away:
  - have solved **more** than asked for (also sorted)
  - extra work means $\Omega(n \log n)$ (mergesort)
  - but helps do required work efficiently (also $O(n \log n)$)

## Finding Minimum Distance between Points

- **Problem 1**: Given $n$ points on a line $p_1, p_2, \ldots, p_n \in \mathbb{R}$, find the closest pair: $\min_{i \neq j} |p_i - p_j|$.
  - Compare all pairs $O(n^2)$
  - Better algorithm? Sort and compare adjacent pairs. $O(n \log n)$
- **Problem 2:** Now what if the points are in $\mathbb{R}^2$?
  - Compare all pairs $O(n^2)$
  - Sort? Points can be close in one coordinate and far in other
  - We'll do it in $O(n \log n)$ steps using divide-and-conquer.

## Problem Formulation

- **Input**: set of points $P = \{p_1, \ldots, p_n\}$ where $p_i = (x_i, y_i)$
- **Assumption**: we can iterate over points in order of $x$- or $y$-coordinate in $O(n)$ time.
  Pre-generate data structures to support iteration cost: $O(n \log n)$ time.

## Minimum Distance: Recursive Algorithm

1. Find vertical line $L$ to split points into sets $P_L$, $P_R$ of size $n/2$. $O(n)$
2. Recursively find minimum distance in $P_L$ and $P_R$.
   - $\delta_L$ = minimum distance between $p, q \in P_L, p \neq q$. $T(n/2)$
   - $\delta_R$ = same for $P_R$. $T(n/2)$
3. $\delta_M$ = minimum distance between $p \in P_L, q \in P_R$. ??
4. Return $\min(\delta_L, \delta_R, \delta_M)$.

Naive Step 3 takes $\Omega(n^2)$ time. But if we do it in $O(n)$ time we get

$$T(n) = 2T(n/2) + O(n) \Longrightarrow T(n) = O(n \log n)$$

## Making Step 3 Efficient

- **Goal**: given $\delta_L$, $\delta_R$, compute $\min(\delta_L, \delta_R, \delta_M)$
- Let $\delta = \min(\delta_L, \delta_R)$. If $p \in P_L, q \in P_R$ are at least $\delta$ apart, they cannot be a closer pair, so we can ignore pair $(p, q)$.
- Let $S$ be the set of points within distance $\delta$ from $L$. We only need to consider pairs that are both in $S$.
- For a given point $p \in S$, how many points $q$ are within $\delta$ units of $p$ in the $y$ coordinate?

## How to find closest pair with one point in each side?

Def. Let $s_i$ be the point in the $2\delta$-strip, with the $i^{th}$ smallest $y$-coordinate.

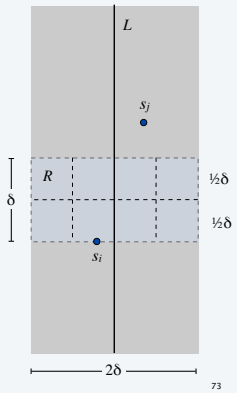
Claim. If $|j - i| > 7$, then the distance between $s_i$ and $s_j$ is at least $\delta$.

Pf.

- Consider the $2\delta$-by-$\delta$ rectangle $R$ in strip whose min $y$-coordinate is $y$-coordinate of $s_i$.
- Distance between $s_i$ and any point $s_j$ above $R$ is $\geq \delta$.
- Subdivide $R$ into 8 squares.
- At most 1 point per square.
- At most 7 other points can be in $R$. ▪

diameter is $\delta / \sqrt{2} < \delta$

constant can be improved with more refined geometric packing argument



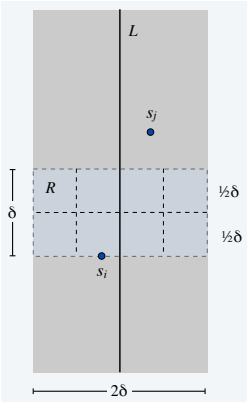



slide credit: Kevin Wayne / Pearson

## Clicker Question 5

What is the maximum number of points with larger $y$ coordinate that we need to compare to $s_i$?

A. 3 points
B. 4 points
C. 7 points
D. 8 points



## Wrap-Up

- Step 3 is $O(n)$: iterate in order of $y$ coordinate and compare each point to constant number of neighbors.
- $\implies O(n \log n)$ overall.
- **Intuition**: we reduced Step 3 (almost) to 1D closest-pair
  - Iterate, compare each point to next $k$ points (instead of 1)
  - The set $S$ is "nearly one-dimensional". Points cannot be packed too tightly, because pairs on each side have to be at least $\delta$ apart.
- For $d > 2$ dimensions, there is a divide and conquer algorithm where the "combine" step (i.e., Step 3) solves a closest pair problem in $d - 1$ dimensions