Divide and Conquer Examples

Counting Inversions

Some web sites try to match your preference (for music, movies, books) with others. How do you compare two rankings?

For example:

I like (best) B D C A
You like (worst) A 0 B C

Count: how many pairs do we rank differently? Answer: 4
That is, how many pairs of lines cross (out of 6 pairs). (Note: $\binom{4}{2} = \frac{4 \cdot 3}{2} = 6$.)

BD BA DA CA (and the two pairs that don’t cross are BC DC)

Equivalently: my ranking 1 2 3 4
your ranking 4 2 1 3

and we count # inversions = # pairs out of order in 2nd list.

Brute Force: check all $\binom{n}{2}$ pairs: $O(n^2)$
Does sorting help? Doesn’t seem to.
Counting inversions: naive divide and conquer approach

**Input:** \( L = [4 \ 6 \ 1 \ 5 \ 2 \ 3] \) of length \( n = 6 \)

**Goal:** Count the number of inversions.

1. **Divide:** Split list into two parts \( A \) (first half) and \( B \) (second half).

\[
L = [4 \ 6 \ 1 \ | \ 5 \ 2 \ 3]
\]

2. **Recursively count # of inversions** \( r_A \) in \( A \) and \( r_B \) in \( B \). We get

\[
r_A = 2 \quad \text{and} \quad r_B = 2
\]

3. How to combine? That is, how to count additional inversions?

   **Answer:** For each element in \( B \) count how many elements in \( A \) are greater.

   - \# elements in \( A \) greater than 5: 1
   - \# elements in \( A \) greater than 2: 2
   - \# elements in \( A \) greater than 3: 2

   Number of inversions between \( A \) and \( B \) is \( r = 1 + 2 + 2 = 5 \).

   Return \( r_A + r_B + r = 9 \).

   Using brute force approach, combine step has cost \( O(n^2) \). No gain!
Counting inversions: divide and conquer with improved combine phase

Given list $a_1, \ldots, a_n$, count # inversions.

- Divide list in two: $m = \lceil \frac{n}{2} \rceil$
- Recursively count # inversions in each half, $r_A, r_B$
- Combine: answer $\leftarrow r_A + r_B + r$

$$r = \# \text{ inversions with one element in } A, \text{ one in } B$$
$$= \# \text{ pairs } a_i, a_j \text{ with } a_i \in A, \ a_j \in B, \ a_i > a_j$$

How do we find $r$?

Can we count, for each $a_j \in B$, how many larger elements are there in $A$? — $r_j$

Then $r = \sum_{a_j \in B} r_j$

Think about mergesort: sort $A$, sort $B$, merge

when $a_j$ is output to merged list: $r_j \leftarrow k$
Whole algorithm:

sort-and-count(L) – returns sorted L, # inversions

- divide L into A, B (first half, second half)
- $(r_A, A) \leftarrow$ sort-and-count(A)
- $(r_B, B) \leftarrow$ sort-and-count(B)
- $r \leftarrow 0$

- Do merge of A and B
  - When elements is moved from B to output
    - $r \leftarrow r + \# \text{ elements remaining in } A$
  - return $(r_A + r_B + r, \text{merged list})$

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

solution: $T(n) = O(n \log n)$ (as in mergesort)

Question: Is there a better algorithm?

- $O(n \log n / (\log \log n))$ ’89
- $O(n \sqrt{\log n})$ 2010 Timothy Chan et al. using techniques/model
  where sorting is $o(n \log n)$
Finding the closest pair of points: an example with the “conquer” step more complicated

**Problem:** Given \( n \) points in the plane, find the pair that is closest together.

\[
d(p, q) = \text{distance between } p \text{ and } q
= \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}
\]

Note: we can compare \( d(p, q) \) to \( d(r, s) \) without \( \sqrt{} \).

**Brute force:** try all pairs: \( O(n^2) \).

In 1D (points on a line): sort and check consecutive pairs: \( O(n \log n) \).

Note that this does not work in 2D.

**Divide and Conquer** (after sorting by \( x \)-coord.)

- divide points in left half, \( Q \), right half, \( R \), dividing line \( L \)
- recursively find closest pair in \( Q \), closest pair in \( R \)
- combine

To combine, we need to find close pairs crossing \( L \).
This is the tricky part.
Let $\delta = \min$ distance of $\{\text{closest pair in } Q, \text{closest pair in } R\}$

Must check pairs $q \in Q$, $r \in R$ with $d(q, r) < \delta$.

**Claim:** Such points satisfy $d(q, L) < \delta$, $d(r, L) < \delta$

**Proof:** Otherwise horizontal distance $\geq \delta$, so distance $\geq \delta$.

Let $S =$ points in this vertical strip of width $2\delta$.
We can restrict our search to $S$ (might be all $n$ points!)
This problem seems more 1-dimensional.
Sort by $y$-coordinate.

Note: Do not do this in each recursive call. Do it once for all points $O(n \log n)$
and extract the sorted sublist for $S$ in linear time.
This is like “unmerging”. For example, points sorted by $y$-coordinate

$P_1, P_2, P_3, P_4, P_5, \ldots, P_r, P_{r+1}, \ldots, P_n$

points of $S$ circled: $P_2, P_3, P_c$

Note: Each recursive call needs to know its points sorted by $y$-coordinate.
\[ s = \min \left( \delta_Q, \delta_R \right) \]
Overall structure of the algorithm:

\[ X \leftarrow \text{sort points by } x\text{-coord} \]
\[ Y \leftarrow \text{sort points by } y\text{-coord} \]
\[ \text{closest}(X, Y) \text{ — returns distance between closest pair of points} \]
\[ \text{closest}(X, Y) \]
\[ L \leftarrow \text{dividing line (middle of } X) \]

"unmerge" to get \( X_Q, X_R, Y_Q, Y_R \)

sorted lists for leftside \( (Q) \), rightside \( (R) \)
\[ \delta_Q \leftarrow \text{closest}(X_Q, Y_Q) \]
\[ \delta_R \leftarrow \text{closest}(X_R, Y_R) \]
\[ \delta \leftarrow \min\{\delta_Q, \delta_R\} \]

find set \( S \) as above
\[ Y_S \leftarrow S \text{ sorted by } y\text{-coord (extract from } Y) \]

Finally, what do we do with \( S, Y_S \)?

Our hope: if \( q, r \in S, q \in Q, r \in R \) and \( d(q, r) < \delta \)
then \( q \) and \( r \) are near each other in the sorted \( S \).
Why? We have 8 squares $\delta^2 \times \delta^2$ and each square has $\leq 1$ point.

Thus if $q, r \in S$, $q \in Q, r \in R$, $d(q, r) < \delta$
then $q$ and $r$ are at most 8 positions apart in sorted $S'$.

Wrapping up: preliminary sort by $x$ and $y$
then $T(n) = 2T(n/2) + cn$ so $T(n) \in O(n \log n)$

This algorithm was due to Preparata & Shamos in the early days of computational geometry (70’s and 80’s).

In fact, one can do much more in $O(n \log n)$ — find closest neighbours of all points