403: Algorithms and Data Structures

Lower Bound for Sorting

&

The closest pair in 2D

Fall 2016

UAlbany

Computer Science
So far: Sorting

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion</td>
<td>$O(n^2)$</td>
<td>in-place</td>
</tr>
<tr>
<td>Merge</td>
<td>$O(n \log n)$</td>
<td>2\textsuperscript{nd} array to merge</td>
</tr>
<tr>
<td>Heapsort</td>
<td>$O(n \log n)$</td>
<td>in-place</td>
</tr>
<tr>
<td>Quicksort</td>
<td>from $O(n \log n)$</td>
<td>in-place</td>
</tr>
<tr>
<td></td>
<td>to $O(n^2)$</td>
<td></td>
</tr>
</tbody>
</table>

Can we do better than $O(n \log n)$?

Spoiler: Not if we do comparisons only
Lower bound on comparison sorts

• Any algorithm performing only comparisons runs in $n \log n$.
• We will prove this using the concept of decision trees.
Closest pair in 2D

• Given n points in 2-dimensions, find two whose mutual distance is smallest.

• Euclidean distance

\[ d(p, q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2} \]
Closest pair in 2D

- Brute force?
  - Consider all pairs
- Complexity?
  - $O(n^2)$
Divide-And-Conquer (1D)

- We can simply sort and consider consecutive pairs $O(n\log n)$
  - Does not generalize to 2D
Divide-And-Conquer (1D)

- **DIVIDE**: split array in two equal parts
- **CONQUER**: recursively find closest pair in parts
- **COMBINE**:
  - Let $d$ be the smallest separation in $S1$ and $S2$
  - If $\text{dist}(p3,q3)<d$ return $\text{dist}(p3,q3)$ else $d$
Divide-And-Conquer (1D) Pseudo code

Closest-Pair-1D(S)

If \(|S| = 1\), output \(d = \infty\)
If \(|S| = 2\), output \(d = |p_2 - p_1|\)
Otherwise, do the following steps:

1. Let \(m = \text{median}(S)\)
2. Divide \(S\) into \(S_1, S_2\) at \(m\).
3. \(d_1 = \text{Closest-Pair-1D}(S_1)\).
4. \(d_2 = \text{Closest-Pair-1D}(S_2)\).
5. \(d_{12}\) is minimum distance across the cut.
6. Return \(d = \min(d_1; d_2; d_{12})\)
Divide-And-Conquer (1D)

- Key observation: If \( m \) is the dividing coordinate, then \( p_3, q_3 \) must be within \( d \) of \( m \).
  - \( p_3 \) must be the rightmost point in \( S_1 \)
  - \( q_3 \) must be the leftmost point in \( S_2 \)
  - Hard to generalize to 2D

- How many points of \( S_1 \) can be in \( (m-d, m] \)?
Divide-And-Conquer (2D)

- **DIVIDE**: split points in two equal parts with line $L$
- **CONQUER**: recursively find closest pair in parts
- **COMBINE**:
  - $d = \min(d_{\text{left}}, d_{\text{right}})$
  - $d$ is the answer unless $L$ split points that are close
Region near L

• If there is a pair \((p,q)\) within distance \(d\) split by \(L\), then both \(p\) and \(q\) must be within \(d\) from \(L\)
• Let \(S_y\) be an array of points in the region sorted by \(y\) coordinate
• size of \(S_y\) might be \(O(|S|)\)
  – Cannot check all pairs
Special structure in Sy

• Let: $\text{Sy} = p_1, p_2, \ldots, p_m$, then if $\text{dist}(p_i, p_j) < d$ then $j - i \leq 15$

• close-by points are closeby in the array
Proof: close points within 15 positions

- Divide the region in squares of side \( \frac{d}{2} \)
- How many points in each box?
- At most 1
  - Each box in contained in one half
  - No 2 points in a half are closer than \( d \)
Proof: close points within 15 positions

- Suppose 2 points separated by 15 indices
- At least 3 full rows separate them
- Height of 3 rows >3d/2 > d
- Points are farther than d from each other
Divide and Conquer(2D)

ClosestPair(ptsX, ptsY)

1. if (size(ptsX)<2) return null
2. if (size(ptsX)==2) return ptsX
3. \( m \) = median of x coordinates
4. Prepare subsets to the left of \( m \): ptsX-left, ptsY-left and to the right of \( m \): ptsX-right, ptsY-right // They should be sorted but you should not use sorting (see book chapter)
5. pair-left = ClosestPair(ptsX-left, ptsY-left)
6. pair-right = ClosestPair(ptsX-right, ptsY-right)
7. \( d \) = min of distances between pair-left and pair-right
8. res = pair among pair-left and pair-right of the smaller distance
9. ptsWithinD: an array of points within distance \( d \) from \( m \), sorted by y coordinates
10. for \( i=1 \ldots ptsWithinD.length \)
11. for \( j=i+1 \ldots \min(ptsWithinD.length, i+15) \)
12. if dist(ptsWithinD[i], ptsWithinD[j])<d
13. res = (ptsWithinD[i], ptsWithinD[j])
14. b = dist(ptsWithinD[i], ptsWithinD[j])
15. return res
Analysis

• Divide set of points in half each time:
  – depth of recursion is $O(\log n)$
• Merge takes $O(n)$ time.
• Recurrence: $T(n) = 2T(n/2) + cn$
• Same as MergeSort: $O(n \log n)$ time.