CSE 421

Divide and Conquer

Shayan Oveis Gharan
Boiling Water Example

Q: Given an empty bowl, how do you make boiling water?

A: Well, I fill it with water, turn on the stove, leave the bowl on the stove for 20 minutes. I have my boiling water.

Q: Now, suppose you have a bowl of water, how do you make boiling water?

A: First, I pour water away, now I have an empty bowl and I have already solved this!
Lesson: Never solve a problem twice!
Divide and Conquer Approach
Finding the Root of a Function
Finding the Root of a Function

Given a continuous function $f$ and two points $a < b$ such that
\[
    f(a) \leq 0 \quad f(b) \geq 0
\]
Find an approximate root of $f$ (a point $c$ where there is $r$ s.t.,
$|r - c| \leq \epsilon$ and $f(r) = 0$).

Note that roots of $f$ may be irrational,
So, we want to approximate the root with an arbitrary precision!

\[
f(x) = \sin(x) - \frac{100}{\sqrt{x}} + x^4
\]
A Naive Approach

Suppose we want $\epsilon$ approximation to a root.

Divide $[a, b]$ into $n = \frac{b-a}{\epsilon}$ intervals. For each interval check

$$f(x) \leq 0, f(x + \epsilon) \geq 0$$

This runs in time $O(n) = O\left(\frac{b-a}{\epsilon}\right)$

Can we do faster?
D&C Approach (Based on Binary Search)

\textbf{Bisection}(a,b, \varepsilon)

\textbf{if} (b - a) < \varepsilon \textbf{ then}
    \textbf{return} (a)

\textbf{else}

    m \leftarrow (a + b)/2

\textbf{if} \ f(m) \leq 0 \textbf{ then}
    \textbf{return}(\text{Bisection}(c, b, \varepsilon))

\textbf{else}

    \textbf{return}(\text{Bisection}(a, c, \varepsilon))
Time Analysis

Let \( n = \frac{a-b}{\epsilon} \)

And \( c = (a + b)/2 \)

Always half of the intervals lie to the left and half lie to the right of \( c \)

So,

\[
T(n) = T\left(\frac{n}{2}\right) + O(1)
\]

i.e., \( T(n) = O(\log n) = O(\log \frac{a-b}{\epsilon}) \)
Correctness Proof

\[ P(k) = \text{"For any } a, b \text{ such that } k\varepsilon \leq |a - b| \leq (k + 1)\varepsilon \text{ if } f(a)f(b) \leq 0, \text{ then we find an } \varepsilon \text{ approx to a root using } \log k \text{ queries to } f." \]

**Base Case:** \( P(1) \): Output \( a + \varepsilon \)

**IH:** Assume \( P(k) \).

**IS:** Show \( P(2k) \). Consider an arbitrary \( a, b \) s.t.,
\[ 2k\varepsilon \leq |a - b| < (2k + 1)\varepsilon \]

If \( f(a + k\varepsilon) = 0 \) output \( a + k\varepsilon \).

If \( f(a)f(a + k\varepsilon) < 0 \), solve for interval \( a, a + k\varepsilon \) using \( \log(k) \) queries to \( f \).

Otherwise, we must have \( f(b)f(a + k\varepsilon) < 0 \) since \( f(a)f(b) < 0 \) and \( f(a)f(a + k\varepsilon) \geq 0 \). Solve for interval \( a + k\varepsilon, b \).

Overall we use at most \( \log(k) + 1 = \log(2k) \) queries to \( f \).
Master Theorem

Suppose $T(n) = a \, T \left( \frac{n}{b} \right) + cn^k$ for all $n > b$. Then,

- If $a > b^k$ then $T(n) = \Theta(n^{\log_b a})$

- If $a < b^k$ then $T(n) = \Theta(n^k)$

- If $a = b^k$ then $T(n) = \Theta(n^k \log n)$

Works even if it is $\left\lfloor \frac{n}{b} \right\rfloor$ instead of $\frac{n}{b}$.

We also need $a \geq 1, b > 1, k \geq 0$ and $T(n) = O(1)$ for $n \leq b$. 
Master Theorem

Suppose $T(n) = a T\left(\frac{n}{b}\right) + cn^k$ for all $n > b$. Then,

- If $a > b^k$ then $T(n) = \Theta(n^{\log_b a})$
- If $a < b^k$ then $T(n) = \Theta(n^k)$
- If $a = b^k$ then $T(n) = \Theta(n^k \log n)$

**Example:** For mergesort algorithm we have

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

So, $k = 1, a = b^k$ and $T(n) = \Theta(n \log n)$
Finding the Closest Pair of Points
Closest Pair of Points (non geometric)

Given n points and arbitrary distances between them, find the closest pair. (E.g., think of distance as airfare – definitely not Euclidean distance!)

Must look at all n choose 2 pairwise distances, else any one you didn’t check might be the shortest. i.e., you have to read the whole input
Closest Pair of Points (1-dimension)

Given n points on the real line, find the closest pair, e.g., given 11, 2, 4, 19, 4.8, 7, 8.2, 16, 11.5, 13, 1 find the closest pair

Fact: Closest pair is adjacent in ordered list
So, first sort, then scan adjacent pairs.
Time $O(n \log n)$ to sort, if needed, Plus $O(n)$ to scan adjacent pairs

Key point: do not need to calc distances between all pairs: exploit geometry + ordering
Closest Pair of Points (2-dimensions)

Given \( n \) points in the plane, find a pair with smallest Euclidean distance between them.

Fundamental geometric primitive.

- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST, Voronoi.

**Brute force:** Check all pairs of points \( p \) and \( q \) with \( \Theta(n^2) \) time.

**Assumption:** No two points have same \( x \) or \( y \) coordinates.
A Divide and Conquer Alg

Divide: draw vertical line $L$ with $\approx n/2$ points on each side.
Conquer: find closest pair on each side, recursively.
Combine to find closest pair overall
Return best solutions

seems like $\Theta(n^2)$ ?
Key Observation

Suppose $\delta$ is the minimum distance of all pairs in left/right of L.

$$\delta = \min(12, 21) = 12.$$  

**Key Observation**: suffices to consider points within $\delta$ of line L.

Almost the one-D problem again: Sort points in $2\delta$-strip by their y coordinate.

Only check pts within 11 in sorted list!
Almost 1D Problem

Partition each side of L into $\frac{\delta}{2} \times \frac{\delta}{2}$ squares

Claim: No two points lie in the same $\frac{\delta}{2} \times \frac{\delta}{2}$ box.

Pf: Such points would be within

$$\sqrt{\left(\frac{\delta}{2}\right)^2 + \left(\frac{\delta}{2}\right)^2} = \delta \sqrt{\frac{1}{2}} \approx 0.7\delta < \delta$$

Let $s_i$ have the $i^{th}$ smallest y-coordinate among points in the $2\delta$-width-strip.

Claim: If $|i - j| > 11$, then the distance between $s_i$ and $s_j$ is $> \delta$.

Pf: only 11 boxes within $\delta$ of $y(s_i)$. 
Closest Pair (2Dim Algorithm)

Closest-Pair(p_1, ..., p_n) {
    if(n <= ??) return ??

    Compute separation line L such that half the points are on one side and half on the other side.

    δ_1 = Closest-Pair(left half)
    δ_2 = Closest-Pair(right half)
    δ = min(δ_1, δ_2)

    Delete all points further than δ from separation line L

    Sort remaining points p[1]...p[m] by y-coordinate.

    for i = 1..m
        for k = 1...11
            if i+k <= m
                δ = min(δ, distance(p[i], p[i+k]));

    return δ.
}
Closest Pair Analysis I

Let $D(n)$ be the number of pairwise distance calculations in the Closest-Pair Algorithm when run on $n \geq 1$ points.

$$D(n) \leq \begin{cases} 1 & \text{if } n = 1 \\ 2D \left( \frac{n}{2} \right) + 11 n & \text{otherwise} \end{cases}$$

$\Rightarrow D(n) = O(n \log n)$

BUT, that’s only the number of distance calculations.

What if we counted running time?

$$T(n) \leq \begin{cases} 1 & \text{if } n = 1 \\ 2T \left( \frac{n}{2} \right) + O(n \log n) & \text{otherwise} \end{cases}$$

$\Rightarrow D(n) = O(n \log^2 n)$
Can we do better? (Analysis II)

Yes!!

Don’t sort by y-coordinates each time.
Sort by x at top level only.

This is enough to divide into two equal subproblems in O(n)
Each recursive call returns δ and list of all points sorted by y
Sort points by y-coordinate by merging two pre-sorted lists.

\[
T(n) \leq \begin{cases} 
1 & \text{if } n = 1 \\
2T\left(\frac{n}{2}\right) + O(n) & \text{o.w.} \end{cases} \Rightarrow D(n) = O(n \log n)
\]