Basic Algorithms - Sec 7

More D&C.

Closest pair of points in $\mathbb{R}^2$

given $(x_i, y_i)_{i=1}^n$, find the pair of points closest to each other.

Naive: Check all $\binom{n}{2}$ pairs & keep smallest.

Better: Use D&C.

Take the $n$ points & split it into two parts of size $n/2$ each based on the $x$-coordinate.
Recursively solve the problem for sizes \( \frac{\text{size}}{2} \) to get two minimum distances \( d_2, d_r \). Def \( d = \min(d_2, d_r) \).

Now we know the smallest distances in each part, but there could be two closer points across the split.

Notice that if the closest pair with distance less than \( d \) is across the split, then they are in a strip of \( 2d \) along the split. So for the conquering step we only need to care about the points in the strip.
This boils the problem down to finding the closest pair in the strip of size $2d$. Could potentially have every point in the strip to get $O(n^2)$ again.

$\Rightarrow$ In a square of size $2d \times 2d$, there can be at most $7$ points before two of them have distance smaller than $d$.

Proof: Cover with $7$ circles of diameter $d$ as follows.

by pigeon-hole principle if there are more than $7$ points in this rectangle $\Rightarrow$ two of them are in the same circle $\Rightarrow$ their distance is less than $d$. 
Hence for each point in the strip you only have to compare it to the next 7 points, \( \Rightarrow \) conquering is \( O(n) \).

\[
T(n) = 2T(n/2) + O(n).
\]

We can find the closest pair of points in \( O(n \log n) \).

Proof that we can't do faster.

"Reduction" from the element uniqueness problem: given \( a_1, \ldots, a_n \) are all of them distinct? What is a "reduction"? (what is hardness?)

Say you have two problems \( A \) & \( B \) and you want to claim \( A \) is harder than \( B \). That means if you can solve \( A \) \( \Rightarrow \) you can solve \( B \). So just focus on solving \( A \).
E.g.

Reductions are used to give lower bounds. We will show how to use the closest pair problem to solve element uniqueness.

1) given a, ..., g an construct n points.
   \((a_1, 0), (a_2, 0), \ldots, (a_n, 0)\)

2) Find the smallest distance between the points. \(D\).

3) \(\text{if } D = 0, \text{ then } \text{NO else } \text{YES.}\)

\[ T_{EU}(n) \leq O(n) + T_{CP}(n). \]

(There could be other techniques...).

Fact: lower bound for element uniqueness \( = O(n\log n)\)

\[ T_{EU}(n) \not\leq O(n\log n). \]

\[ = T_{CP}(n) \not\leq O(n\log n). \]
Proof for element uniqueness

Visit the decision tree model for computation.
Input → \( a_1, \ldots, a_n \) ≤ [\( b_{\pi(1)}, b_{\pi(2)}, \ldots, b_{\pi(n)} \)]
(sorted order = \( b_1, b_2, \ldots, b_n \))

Claim 1. on the path to a YES leaf you must have compared \( b_i, b_{i+1} \) ∀ \( i < n \)

Proof. If not replace \( b_{i+1} \) with \( b_i \) & give the input, which will follow the same path & reach a YES leaf. \( \Rightarrow \)

Claim 2. each permutation has its own leaf (YES)

Proof. As you compared, each \( b_i, b_{i+1} \)
you get \( b_1 < b_2 < b_3 < \ldots < b_n \)
so you get an ordering of the input which determines \( \pi \).

\( \Rightarrow \) each leaf (YES) has a unique permutation

\( \Rightarrow \) \( \# \) of leaves.

\( \Rightarrow \) \( \log n \), \( O(n \log n) \)
**Convex Hull Problem**

**Def:** given $P_i = (x_i, y_i)$, $i = 1 \ldots n$ as points, the convex hull of $\{P_i\}_{i=1}^n$ is a subset of points $P_1', P_2', \ldots, P_K'$ such that $P_1'$ form a convex polygon & the polygon contains all $P_i$'s.

**E.g.**

![Diagram of convex hull](image)

**Note:** you need to output them in order (either cyclic directly).

**Seems like a hard problem without naive sol', but first find one vertex → lowermost. now find its right neighbor. → $O(n)$ continue → $O(n^2)$.**
Can do better using D&C.

1) Divide the points into two halves, & recursively get two convex hulls.

2) To merge we need to find the two tangents: bottom & top.

Def. tangent: a line joining two vertices such that both the hulls are on the same side.

E.g.

Once we find the tangents we can merge the two by ignoring the inner sectors.
For the merge step if it takes \( f(n) \) we get
\[
T(n) = 2T(n/2) + f(n).
\]
to get \( T(n) = \theta(n \log n) \)
we need \( f(n) = \theta(n) \).
Hence to find the two tangents we can't check all pairs.

To find tangents:

1) first join the extreme points to the left & right of 400 polygons.

2) "Wiggle" the line up/down till the full polygon is on one side.

E.g.

How to check if the whole polygon is on one side?
Naive takes $O(n)$.

Better: Don't need to check all points.  
only check left & right neighbour.

(*) Because polygon is convex.

Wiggling takes $O(n)$ to find one tangent  
Hence we can merge two polygons in $O(n)$. 

$\Rightarrow$ Convex Hull can be found in $O(n \log n)$.  

$T(n) = 2 \cdot T(n/2) + O(n)$.  

(*) Master theorem.
We cannot do better:
reduce sorting to convex hull.

1) given $a_1, \ldots, a_n$ construct points.
   \[ P_i = (a_i, a_i^2) \]

2) the points lie on the parabola \( y = x^2 \)

3) Hence the output of the convex hull of these points is the sorted order of $x$-coordinate.

\[ \Rightarrow T_{\text{sort}}(n) \leq O(n) + T_{\text{ch}}(n). \]

\[ \Rightarrow T_{\text{ch}}(n) \geq \Omega(n \log n). \]
Polynomial Multiplication

$f, g \in \mathbb{R}[x] \rightarrow \text{"set" of all polynomials.}

f = \sum_{i=0}^{n} a_i x^i \quad g = \sum_{i=0}^{n} b_i x^i

(assume of same degree).

\[ f \times g = \sum_{j=0}^{2n} \left( \sum_{i=0}^{j} a_i b_{j-i} \right) x^j \]

Naive algorithm: \( \text{do } O(n^2) \text{ multiplications.} \)

Can do better: \( O(n \log_2 n) \) - Karatsuba

\( O(n \log n \log \log n) \) - FFT

Both are \( \text{D&C.} \)

\[ f = \sum_{i=2^l}^{2^{l+1}-1} f_{hi} x_{hi} + f_{ilo} \]

\[ g = \sum_{i=2^l}^{2^{l+1}-1} g_{hi} x_{hi} + g_{ilo}. \]
\[ f \cdot g = \left( \frac{f}{\text{hi}} \cdot g_{\text{hi}} \right) x + \left( \frac{f}{\text{hi}} \cdot g_{\text{lo}} + \frac{f}{\text{lo}} \cdot g_{\text{hi}} \right) x^{2^{\frac{1}{2}}} \]

+ \frac{f}{\text{lo}} \cdot g_{\text{lo}}.

\[ T(n) = 4 \cdot T(n/2) + O(n) \]

Master theorem \rightarrow O(n^2)

We can improve a bit.

Define \[ f_{\text{mid}} = \frac{f}{\text{hi}} + \frac{f}{\text{lo}} \]

\[ g_{\text{mid}} = g_{\text{hi}} + g_{\text{lo}}. \]

What is:

\[ f_{\text{mid}} \cdot g_{\text{mid}} = \frac{f}{\text{hi}} \cdot g_{\text{hi}} + \frac{f}{\text{lo}} \cdot g_{\text{lo}}. \]

\[ = \left( \frac{f}{\text{hi}} \cdot g_{\text{lo}} + \frac{f}{\text{lo}} \cdot g_{\text{hi}} \right) \]

\[ \rightarrow \text{middle coefficient}. \]
So algorithm is:

1) Compute $f_{hi} g_{hi}, f_{lo} g_{lo}, f_{mid} g_{mid}$ recursively in $3T(n/2)$

2) Do polynomial additions/subtractions to compute the middle polynomial

3) Put them together to get the final answer.

$$T(n) = 3T(n/2) + O(n)$$

$$\Rightarrow T(n) = O(n \log^3 2)$$

Master Theorem.