Lecture 8: Divide and Conquer
Algorithm Design Techniques

Divide & Conquer

• Divide instance into subparts.
• Solve the parts recursively.
• Conquer by combining the answers

To truly fit Divide & Conquer

• each sub-part should be at most a constant fraction of the size of the original input instance
  • e.g. Mergesort, Binary Search, Quicksort (sort of), etc.
Binary search for roots (bisection method)

Given:
• Continuous function \( f \) and two points \( a < b \) with \( f(a) \leq 0 \) and \( f(b) > 0 \)

Find:
• Approximation within \( \epsilon \) of \( c \) s.t. \( f(c) = 0 \) and \( a < c < b \)
Bisection method

Bisection($a$, $b$, $\varepsilon$)

if $(b - a) \leq \varepsilon$ then
    return($a$)
else {
    $c \leftarrow (a + b)/2$
    if $f(c) \leq 0$ then
        return(Bisection($c$, $b$, $\varepsilon$))
    else
        return(Bisection($a$, $c$, $\varepsilon$))
}
Time Analysis

At each step we halved the size of the interval

- It started at size $b - a$
- It ended at size $\varepsilon$

So # of calls to $f$ is $\log_2 \left( \frac{(b - a)}{\varepsilon} \right)$

# of bits of precision
Old Favorites

Binary search:

- One subproblem of half size plus one comparison
- Recurrence* for time in terms of # of comparisons
  - $T(n) = T(n/2) + 1$ for $n \geq 2$
  - $T(1) = 0$
- Solving shows that $T(n) = \lceil \log_2 n \rceil + 1$

Mergesort:

- Two subproblems of half size plus merge cost of $n - 1$ comparisons
- Recurrence* for time in terms of # of comparisons
  - $T(n) \leq 2T(n/2) + n - 1$ for $n \geq 2$
  - $T(1) = 0$
- Roughly $n$ comparisons at each of $\log_2 n$ levels of recursion so $T(n)$ is roughly $n \log_2 n$

*We will implicitly assume that every input to $T(\cdot)$ is rounded up to the nearest integer.
Euclidean Closest Pair

Given:
- A sequence of \( n \) points \( p_1, \ldots, p_n \) with real coordinates in \( d \) dimensions (\( \mathbb{R}^d \))

Find:
- A pair of points \( p_i, p_j \) s.t. the Euclidean distance \( d(p_i, p_j) \) is minimized

What is the first algorithm you can think of?
- Try all \( \Theta(n^2) \) possible pairs

Can we do better if dimension \( d = 1 \)?
Closest Pair in 1 Dimension

Algorithm:
- Sort points so $p_1 \leq p_2 \leq \cdots \leq p_n$
- Find closest adjacent pair $p_i, p_{i+1}$.

Running time: $O(n \log n)$

What about $d = 2$?
Closest Pair in 2 Dimensions

No single direction to sort points to guarantee success!

Let’s try divide & conquer...

How might we divide the points so that each subpart is a constant factor smaller?

Sorting on 1\textsuperscript{st} coordinate doesn’t work
Closest Pair in 2 Dimensions: Divide and Conquer

How might we divide the points so that each subpart is a constant factor smaller?

Split using median $x$-coordinate!
- each subpart has size $n/2$.

Conquer:
- Solve both size $n/2$ subproblems recursively
Recombine to get overall answer?

Take the closer of the two answers?
- works here but....
Closest Pair in 2 Dimensions: Divide and Conquer

How might we divide the points so that each subpart is a constant factor smaller?

Split using median \( x \)-coordinate!
- each subpart has size \( n/2 \).

Conquer:
- Solve both size \( n/2 \) subproblems recursively
Recombine to get overall answer?

Take the closer of the two answers?
- ...but not always!
Closest Pair in 2 Dimensions: Divide and Conquer

Need to worry about pairs across the split!

New idea to handle them

• Let $\delta$ be the distance of the closest pair in the 2 subparts
• This pair is a candidate
• Only need to check width $\delta$ band either side of the median

Within that band ...

• only need to compare each point with the other points in the rectangle of height $\delta$ above it.

How many points can that be?
How many points can there be in that $\delta$ by $2\delta$ rectangle?

**Key idea:** We know that no pair on either side is closer than $\delta$ apart so there can’t be too many!

- Each of the 8 squares of side $\delta/2$ can contain at most 1 point!
  - Because diagonal has length $< \delta$
- So....only need to compare each point with the next 7 points above it to guarantee you’ll find a partner closer than $\delta$ in the rectangle if there is one!
Closest Pair in 2 Dimensions: Divide and Conquer

Fleshing out the algorithm:

Divide:
• At top level we need median $x$ coordinate to split points
• At next level down we’ll need median $x$ coordinate for each side
• Might as well sort all points by $x$ coordinate up front to get all medians at once!

Conquer: Solve the two sub-problems to get two candidate pairs

Recombine:
• Choose closer candidate pair and let its distance be $\delta$
• Select $B = \text{all points in band with } x \text{ coordinates within } \delta \text{ of median}$
• Sort $B$ by $y$ coordinate \( O(n \log n) \text{ total over all calls} \)
• Compare each point in $B$ with next 7 points and update if closer pair found. \( O(n) \)
Closest Pair in 2 Dimensions: Divide and Conquer

Fleshing out the algorithm: A better version:

Preprocess: Compute sorted list $X$ of points by $x$ coordinate
  • Subparts will be defined by two indices into this list
    Compute sorted list $Y$ of points by $y$ coordinate

Divide: Use median in $X$ to get $X_L$ and $X_R$ and filter points of $Y$ to produce
  sorted sublists $Y_L$ and $Y_R$

Conquer: Solve the two sub-problems to get two candidate pairs

Recombine:
  • Choose closer candidate pair and let its distance be $\delta$
  • Filter $Y$ to get $B =$ points in band w/ $x$ coordinates within $\delta$ of median
  • Compare each point in $B$ with next 7 points and update if closer pair found.
Closest Pair in 2 Dimensions: Divide and Conquer

Total runtime = Preprocessing time + Divide and Conquer time

Let \( T(n) \) be Divide and Conquer time:

Recurrence:
- \( T(n) \leq 2T(n/2) + O(n) \) for \( n \geq 3 \)
- \( T(2) = 1 \)

Solution: \( T(n) \) is \( O(n \log n) \).

With preprocessing, total runtime is \( O(n \log n) \).
Sometimes two sub-problems aren’t enough

More general divide and conquer

• You’ve broken the problem into \textit{a} different sub-problems
• Each has size at most \( n/b \)
• The cost of break-up and recombining sub-problem solutions is \( O(n^k) \)
  • “cost at the top level”

Recurrence

• \( T(n) = a \cdot T(n/b) + O(n^k) \) for \( n \geq b \)
• \( T \) is constant for inputs \( < b \).
  • For solutions correct up to constant factors no need for exact base case
Solving Divide and Conquer Recurrence

Master Theorem: Suppose that $T(n) = a \cdot T(n/b) + O(n^k)$ for $n > b$.

- If $a < b^k$ then $T(n)$ is $O(n^k)$
  - Cost is dominated by work at top level of recursion
- If $a = b^k$ then $T(n)$ is $O(n^k \log n)$
  - Total cost is the same for all $\log_b n$ levels of recursion
- If $a > b^k$ then $T(n)$ is $O(n^{\log_b a})$
  - Note that $\log_b a > k$ in this case
  - Cost is dominated by total work at lowest level of recursion

Binary search: $a = 1, b = 2, k = 0$ so $a = b^k$: Solution: $O(n^0 \log n) = O(\log n)$

Mergesort: $a = 2, b = 2, k = 1$ so $a = b^k$: Solution: $O(n^1 \log n) = O(n \log n)$
Proving Master Theorem for $T(n) = a \cdot T(n/b) + c \cdot n^k$

Write $d = \lceil \log_b n \rceil$ so $n \leq b^d$

<table>
<thead>
<tr>
<th>problem size</th>
<th># of problems</th>
<th>level work/problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n \leq b^d$</td>
<td>$1$</td>
<td>$n^k$</td>
</tr>
<tr>
<td>$n/b \leq b^{d-1}$</td>
<td>$a$</td>
<td>$n^k/b^k$</td>
</tr>
<tr>
<td>$n/b^2 \leq b^{d-2}$</td>
<td>$a^2$</td>
<td>$n^k/b^{2k}$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$n/b^{d-1} \leq b$</td>
<td>$a^{d-1}$</td>
<td>$b^k$</td>
</tr>
<tr>
<td>$n/b^d \leq 1$</td>
<td>$a^d$</td>
<td>$1$</td>
</tr>
</tbody>
</table>
Proving Master Theorem for $T(n) = a \cdot T(n/b) + c \cdot n^k$

Write $d = \lfloor \log_b n \rfloor$ so $n \leq b^d$

<table>
<thead>
<tr>
<th># of problems</th>
<th>level work/problem</th>
<th>total work/level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$n^k$</td>
<td>$n^k$</td>
</tr>
<tr>
<td>$a$</td>
<td>$n^k/b^k$</td>
<td>$(a/b^k) \cdot n^k$</td>
</tr>
<tr>
<td>$a^2$</td>
<td>$n^k/b^{2k}$</td>
<td>$(a/b^k)^2 \cdot n^k$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$a^{d-1}$</td>
<td>$b^k$</td>
<td>...</td>
</tr>
<tr>
<td>$a^d$</td>
<td>1</td>
<td>$a^{\log_b n}$</td>
</tr>
</tbody>
</table>

- **total work**
  - If $a < b^k$ sum of geometric series with biggest term $O(n^k)$
  - If $a = b^k$ sum of $O(\log n)$ terms each $O(n^k)$
  - If $a > b^k$ sum of geometric series with biggest term $O(a^{\log_b n})$

**Claim:** $a^{\log_b n} = n^{\log_b a}$

**Proof:** Take $\log_b$ of both sides