Divide and Conquer Recap

- Given a problem of an input of size \( n \),
- We generate (multiple) smaller instances of the problem
- We solve each of these smaller instances
- We use the solutions of the small instances to solve the original problem.

- Suppose that the first and third steps can be performed in \( O(n^\alpha) \) time. If there are \( q \) smaller instances generated, each of size \( n/k \), then the running time \( T(n) \) of the algorithm satisfies the recurrence.

\[
T(n) \leq qT(n/k) + cn^\alpha
\]

Divide and Conquer Algorithms

- Mergesort, Maximum Subsequence Sum
- Integer Multiplication
- Minimum distance
- Today: Counting inversions

Minimum Distance Recap

- Problem: Given \( n \) distinct points \( p_1, \ldots, p_n \in \mathbb{R}^2 \), find the minimum distance between any two points:

\[
\min_{i \neq j} d(p_i, p_j)
\]

\[
d(p, q) = \sqrt{(p[1] - q[1])^2 + (p[2] - q[2])^2}
\]

Naive algorithm takes \( O(n^2) \)
But we can do \( O(n \log_2 n) \).
Minimum Distance Algorithm

- Divide points \( P \) with a vertical line into \( P_L \) and \( P_R \) where \(|P_L| = |P_R| = n/2\)
- Recursively find minimum distance within \( P_L \) and \( P_R \):
  \[ \delta_L = \min_{p,q \in P_L \atop p \neq q} d(p,q) \quad \delta_R = \min_{p,q \in P_R \atop p \neq q} d(p,q) \]
  \[ \delta_M = \min_{p \in P_L, q \in P_R} d(p,q) \]
- Key idea: Can make step 3 \( O(n) \)-time.
- Proof requires “packing” argument.

Counting Inversions

- Consider a music recommendation system that works as follows
  - When you join the service, they ask you to rank \( n \) songs
  - Based on this ranking, they identify people with similar music preferences
  - How to measure “similar” in a large database? Count inversions
    - My ranking is: 1, 2, …, \( n \)
    - Your ranking is: \( a_1, a_2, \ldots, a_n \) (\( a_i \in \{1, \ldots, n\} \))
    - An inversion is a pair \((i, j)\) where \( i < j \) but \( a_i > a_j \).

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
</table>
  me| 1 | 2 | 3 | 4 | 5 |
  you| 5 | 4 | 3 | 2 | 1 |

  - Inversions at \( B - D \) and \( C - D \).

Counting Inversions: Divide and Conquer

- Divide: Split list in two halves \( L, R \)
- Recurse: Count inversions in each half
- Combine: Count inversions \((\ell, r)\) with \( \ell \in L, r \in R \)
- Return the sum

  \[
  \begin{array}{cccccccc}
  1 & 5 & 4 & 8 & 10 & 2 & 6 & 9 & 3 & 7 \\
  \end{array}
  \]

  - Challenge: How to do combine step quickly?

The combine step?

- Challenge: How to do combine step quickly?
- What if \( L \) and \( R \) were sorted?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>5</th>
<th>4</th>
<th>8</th>
<th>10</th>
<th>2</th>
<th>6</th>
<th>9</th>
<th>3</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>me</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>8</td>
<td>10</td>
<td>2</td>
<td>6</td>
<td>9</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>you</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>10</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>

  - Combine step: \( 4 + 4 + 2 + 2 + 1 = 13 \).
  - Can be done in \( O(n) \) time!
Inversions Divide and Conquer

▶ Divide: Split list in two halves \(L, R\)
▶ Recurse: Count inversions in each half and sort each half!
▶ Combine: Count inversions \((\ell, r)\) with \(\ell \in L, r \in R\).
▶ Return the sum and sorted list.

Notes
▶ Solve “harder” problem to make your life easier later.
▶ Important: Count inversions before sorting!

Pseudocode

```java
if length(Arr) ≤ 2 then ⊲ Base case
  run brute force algorithm return inversions and sorted list.
else
  middle = length(Arr)/2 ⊲ Recursive Steps
  \((c_L, L) = \text{CountAndSort}(\text{Arr}[0:middle])\)
  \((c_R, R) = \text{CountAndSort}(\text{Arr}[middle:length(\text{Arr})])\)
  \(\ell = 1, r = 1, C = [], \ c_m = 0\) ⊲ Combine step
  while \(\ell \leq n/2, r \leq n/2\) do
    if \(L[\ell] < R[r]\) then
      C.append\((L[\ell])\), \(\ell = \ell + 1\)
    else
      \(c_m = c_m + (n/2 - \ell + 1)\), C.append\((R[r])\), \(r = r + 1\)
    end if
  end while
  Return \((c_L + c_R + c_m, C)\)
end if
```

Runtime

▶ Two recursive calls of size \(n/2\)
▶ Combine step takes \(O(n)\) times
▶ Recurrence:
\[
T(n) \leq 2T(n/2) + cn
\]
▶ Runtime: \(O(n \log n)\) – same as merge sort.

Divide and Conquer Wrap-up

▶ Intuition: Solve subproblems and combine together
▶ Combine step can be tricky!
▶ Runtime analysis: Solving recurrence relations
▶ Other problems: Convolutions and FFT, Quicksort, Median find

Algorithm Design Techniques

▶ Greedy
▶ Divide and Conquer
▶ Dynamic Programming
▶ Network Flows

Divide and Conquer Recipe

▶ Devise recursive form for solution
▶ Implement recursion

Example. Compute sum of leaf weights for each internal node in \(k\)-ary tree. (From practice exam)
▶ Recursive form \(w(v) = \sum_{u \text{ child of } v} w(u)\).
Dynamic Programming Recipe

- Devise recursive form for solution
- Observe that recursive implementation involves redundant computation. (Often exponential time)
- Design iterative algorithm that solves all subproblems without redundancy.

Example (From HW1)


```
for i = 1, 2, \ldots, n do
    for j = i + 1, \ldots, n do
    end for
end for
```

Running time: $\Theta(n^3)$.

Example (From HW1)


```
B[i,j] =
\begin{cases}
    B[i, j-1] + A[j] & \text{if } j > i \\
    0 & \text{if } j \leq i
\end{cases}
```

```
for i = 1, 2, \ldots, n do
    B[i, i] = 0
    for j = i + 1, \ldots, n do
    end for
end for
```

Running time: $O(n^2)$