CSE 421
Algorithms
Richard Anderson
Lecture 14
Divide and Conquer

Announcements
• Review session, 3:30 pm. CSE 403.
• Midterm. Monday.

What you really need to know about recurrences
• Work per level changes geometrically with the level
  • Geometrically increasing \((x > 1)\) — The bottom level wins
  • Geometrically decreasing \((x < 1)\) — The top level wins
• Balanced \((x = 1)\) — Equal contribution

\[
T(n) = aT(n/b) + n^c
\]
• Balanced: \(a = b^c\)
• Increasing: \(a > b^c\)
• Decreasing: \(a < b^c\)

Divide and Conquer Algorithms
• Split into sub problems
• Recursively solve the problem
• Combine solutions
• Make progress in the split and combine stages
  – Quicksort — progress made at the split step
  – Mergesort — progress made at the combine step
• D&C Algorithms
  – Strassen’s Algorithm — Matrix Multiplication
  – Inversions
  – Median
  – Closest Pair
  – Integer Multiplication
  – FFT

Inversion Problem
• Let \(a_1, \ldots, a_n\) be a permutation of \(1 \ldots n\)
• \((a_i, a_j)\) is an inversion if \(i < j\) and \(a_i > a_j\)

4, 6, 1, 7, 3, 2, 5
• Problem: given a permutation, count the number of inversions
• This can be done easily in \(O(n^2)\) time
  – Can we do better?
Application

- Counting inversions can be used to measure how close ranked preferences are
  - People rank 20 movies, based on their rankings you cluster people who like the same type of movie

Counting Inversions

Count inversions on lower half
Count inversions on upper half
Count the inversions between the halves

Count the Inversions

Problem – how do we count inversions between sub problems in O(n) time?

- Solution – Count inversions while merging

Use the merge algorithm to count inversions

Indicate the number of inversions for each element detected when merging

Inversions

- Counting inversions between two sorted lists
  - O(1) per element to count inversions

  - Satisfies the “Standard recurrence”
  - $T(n) = 2 \cdot T(n/2) + cn$
Computing the Median

- Given n numbers, find the number of rank n/2
- Selection, given n numbers and an integer k, find the k-th largest

Select(A, k)

```
Select(A, k)

Choose element x from A
S₁ = {y in A | y < x}
S₂ = {y in A | y = x}
S₃ = {y in A | y > x}

if (|S₂| >= k)
    return Select(S₂, k)
else if (|S₂| + |S₃| >= k)
    return x
else
    return Select(S₁, k - |S₂| - |S₃|)
```

Randomized Selection

- Choose the element at random
- Analysis can show that the algorithm has expected run time O(n)

Deterministic Selection

- What is the run time of select if we can guarantee that choose finds an x such that |S₁| < 3n/4 and |S₂| < 3n/4

BFPRT Algorithm

- A very clever choose algorithm . . .

Split into n/5 sets of size 5
M be the set of medians of these sets
Let x be the median of M

BFPRT runtime

|S₁| < 3n/4, |S₂| < 3n/4

Split into n/5 sets of size 5
M be the set of medians of these sets
x be the median of M
Construct S₁ and S₂
Recursive call in S₁ or S₂
BFPRT Recurrence

- $T(n) \leq T(3n/4) + T(n/5) + cn$

Prove that $T(n) \leq 20cn$