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: \(a = b^c\)
  – \(T(n) = 4T(n/2) + n^2\)
• Increasing: \(a > b^c\)
  – \(T(n) = 9T(n/8) + n\)
  – \(T(n) = 3T(n/4) + n^{1/2}\)
• Decreasing: \(a < b^c\)
  – \(T(n) = 5T(n/8) + n\)
  – \(T(n) = 7T(n/2) + n^3\)

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

How to multiply 2 x 2 matrices with 7 multiplications
Multiply 2 x 2 Matrices:
<table>
<thead>
<tr>
<th>r</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>u</td>
</tr>
</tbody>
</table>

\[ r = p_1 + p_2 - p_4 + p_6 \]
\[ s = p_4 + p_5 \]
\[ t = p_6 + p_7 \]
\[ u = p_2 - p_3 + p_5 - p_7 \]

Where:
\[ p_1 = (b - d)(f + h) \]
\[ p_2 = (a + d)(e + h) \]
\[ p_3 = (a - c)(e + g) \]
\[ p_4 = (a + b)h \]
\[ p_5 = a(g - h) \]
\[ p_6 = d(f - e) \]
\[ p_7 = (c + d)e \]

Corrected version from AHU 1974
Strassen’s Algorithms
• Treat $n \times n$ matrices as $2 \times 2$ matrices of $n/2 \times n/2$ submatrices
• Use Strassen’s trick to multiply $2 \times 2$ matrices with 7 multiplies
• Base case standard multiplication for single entries
• Recurrence: $T(n) = 7T(n/2) + cn^2$
• Solution is $O(7^{\log_2 n}) = O(n^{\log_7 7})$ which is about $O(n^{2.807})$

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 that same type of movie

Counting Inversions

Count the Inversions

Problem – how do we count inversions between sub problems in $O(n)$ time?
• Solution – Count inversions while merging

Standard merge algorithm – add to inversion count when an element is moved from the upper array to the solution
Use the merge algorithm to count inversions

<table>
<thead>
<tr>
<th>1</th>
<th>4</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>7</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5</th>
<th>8</th>
<th>9</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>10</td>
<td>13</td>
<td>14</td>
</tr>
</tbody>
</table>

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

### Algorithm summary
- Satisfies the “Standard recurrence”
- $T(n) = 2T(n/2) + cn$

Computing the Median

- Given $n$ numbers, find the number of rank $n/2$
- One approach is sorting
  - Sort the elements, and choose the middle one
  - Can you do better?

Problem generalization

- Selection, given $n$ numbers and an integer $k$, find the $k$-th largest

Select(A, k)

```java
Select(A, k) {
    Choose element $x$ from A
    $S_1 \leftarrow \{y \in A \mid y < x\}$
    $S_2 \leftarrow \{y \in A \mid y > x\}$
    $S_3 \leftarrow \{y \in A \mid y = x\}$
    if ($|S_2| \geq k$)
        return Select($S_2$, k)
    else if ($|S_1| + |S_3| \geq k$)
        return x
    else
        return Select($S_1$, $k - |S_2| - |S_3|$)
}
```

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 |S_1| < 3n/4 and |S_2| < 3n/4 in O(n) time

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

|S_1| < 3n/4, |S_2| < 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 S_1 and S_2
Recursive call in S_1 or S_2

BFPRT Recurrence

- T(n) <= T(3n/4) + T(n/5) + c n

Prove that T(n) <= 20 c n