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:

$$\begin{vmatrix} r & s \\ t & u \end{vmatrix} = \begin{vmatrix} a & b \\ c & d \end{vmatrix} \begin{vmatrix} e & g \\ f & h \end{vmatrix}$$

$r = p_1 + p_2 – p_4 + p_6$

$s = p_4 + p_5$

$t = p_6 + p_7$

$u = p_2 - p_3 + p_5 - p_7$

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 x n matrices as 2 x 2 matrices of n/2 x n/2 submatrices
- Use Strassen's trick to multiply 2 x 2 matrices with 7 multiplies
- Base case standard multiplication for single entries
- Recurrence: $T(n) = 7\ T(n/2) + cn^2$
- Solution is $O(7^{\log n}) = O(n^{\log 7})$ which is about $O(n^{2.807})$

Inversion Problem

- Let $a_1, \ldots, a_n$ be a permutation of 1 . . 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 use 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

11	12	4	1	7	2	3	15	9	5	16	8	6	13	10	14

Count inversions on lower half

Count inversions on upper half

Count the inversions between the halves

Count the Inversions

5

11	12	4	1

2

7	2	3	15

8

3

9	5	16	8

1

6	13	10	14

6

15

11	12	4	1	7	2	3	15

10

9	5	16	8	6	13	10	14

19

43

11	12	4	1	7	2	3	15	9	5	16	8	6	13	10	14

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

- Solution – Count inversions while merging

1	2	3	4	7	11	12	15

5	6	8	9	10	13	14	16

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

1	4	11	12

2	3	7	15

5	8	9	16

6	10	13	14

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

x	x	x	x	x	x	x	x

y	y	y	y	y	y	y	y

z	z	z	z	z	z	z	z	z	z	z	z	z	z	z	z

- Algorithm summary
  - Satisfies the "Standard recurrence"
  - T(n) = 2 T(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)

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

S_1	S_3	S_2

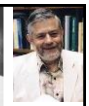
Randomized Selection

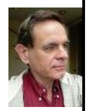
- Choose the element at random
- Analysis can show that the algorithm has expected run time O(n)

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 $T(n) <= 20\ c\ n$