CSE 417 Algorithms

Lecture 16, Winter 2020
Inversions and 2-d Closest Pair

Announcements

- Exams will be returned at end of class

Divide and Conquer Algorithms

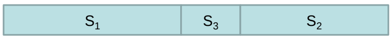
- Mergesort, Quicksort
- Strassen's Algorithm
- Median
- Inversion counting
- Closest Pair Algorithm (2d)
- Integer Multiplication (Karatsuba's Algorithm)

Select the k-th largest from an array

- *Selection*, given n numbers and an integer k, find the k-th largest
- Median is a special case
- The standard approach is to use a quicksort like algorithm
  - But with one recursive problem
- The difficulty is ensuring a good split
  - Worst case $O(n^2)$ time

Select(A, k)

```
Select(A, k){
        Choose a pivot 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 $|S_1| < 3n/4$ and $|S_2| < 3n/4$ in O(n) time

What to know about median finding

- The key to the algorithm is pivot selection
- Choosing a random pivot works well
- Improved random pivot selection: median of three
- Randomized algorithms can find median with 3/2 n comparisons
- Deterministic median finding is harder
  - BFPRT Algorithm guarantees a 3n/4-n/4 split


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

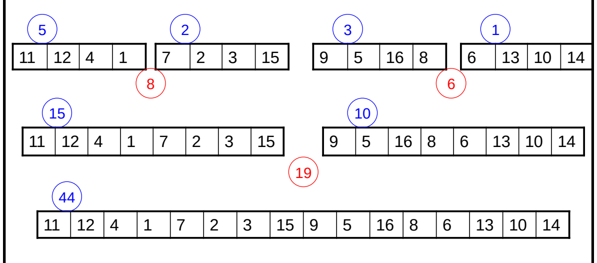
11	12	4	1	7	2	3	15	9	5	16	8	6	13	10	14

Count inversions on lower half

Count inversions on upper half

Count the inversions between the halves

Count the Inversions



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

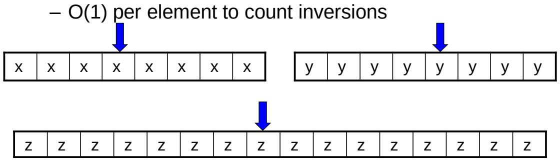
5	8	9	16

6	10	13	14

Indicate the number of inversions for each element detected when merging

Inversions

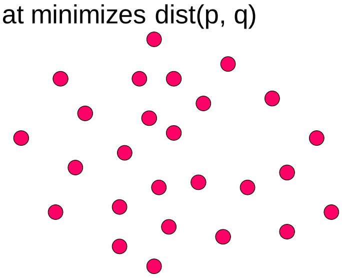
- Counting inversions between two sorted lists
  - O(1) per element to count inversions



- Algorithm summary
  - Satisfies the "Standard recurrence"
  - $T(n) = 2\,T(n/2) + cn$

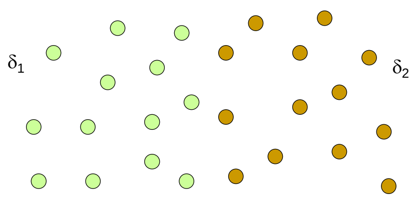
Closest Pair Problem (2D)

- Given a set of points find the pair of points p, q that minimizes dist(p, q)



Divide and conquer

- If we solve the problem on two subsets, does it help? (Separate by median x coordinate)



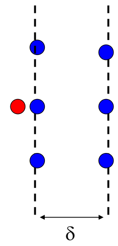
Packing Lemma

Suppose that the minimum distance between points is at least δ, what is the maximum number of points that can be packed in a ball of radius δ?

Combining Solutions

- Suppose the minimum separation from the sub problems is δ
- In looking for cross set closest pairs, we only need to consider points with δ of the boundary
- How many cross border interactions do we need to test?

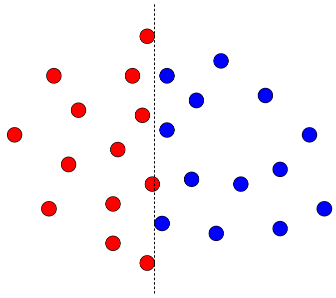
A packing lemma bounds the number of distances to check



Details

- Preprocessing: sort points by y
- Merge step
  - Select points in boundary zone
  - For each point in the boundary
    - Find highest point on the other side that is at most δ above
    - Find lowest point on the other side that is at most δ below
    - Compare with the points in this interval (there are at most 6)

Identify the pairs of points that are compared in the merge step following the recursive calls



Algorithm run time

- After preprocessing:
  - $T(n) = cn + 2\,T(n/2)$

Integer Arithmetic

```
    9715480283945084383094856701043643845790217965702956767
+   1242431098234099057329075097179898430928779579277597977
```

Runtime for standard algorithm to add two n digit numbers:

```
    2095067093034680994318596846868779409766717133476767930
X   5920175091777634709677679342929097012308956679993010921
```

Runtime for standard algorithm to multiply two n digit numbers:

Recursive Multiplication Algorithm (First attempt)

$x = x_1 2^{n/2} + x_0$

$y = y_1 2^{n/2} + y_0$

$xy = (x_1 2^{n/2} + x_0) (y_1 2^{n/2} + y_0)$

$\quad = x_1y_1 2^n + (x_1y_0 + x_0y_1)2^{n/2} + x_0y_0$

Recurrence:

Run time:

Simple algebra

$x = x_1 2^{n/2} + x_0$

$y = y_1 2^{n/2} + y_0$

$xy = x_1y_1 2^n + (x_1y_0 + x_0y_1) 2^{n/2} + x_0y_0$

$p = (x_1 + x_0)(y_1 + y_0) = x_1y_1 + x_1y_0 + x_0y_1 + x_0y_0$

Karatsuba's Algorithm

Multiply n-digit integers x and y

Let $x = x_1 2^{n/2} + x_0$ and $y = y_1 2^{n/2} + y_0$
Recursively compute
$a = x_1y_1$
$b = x_0y_0$
$p = (x_1 + x_0)(y_1 + y_0)$
Return $a2^n + (p - a - b)2^{n/2} + b$

Recurrence: $T(n) = 3T(n/2) + cn$

$\log_2 3 = 1.58496250073\ldots$

Next week

- Dynamic Programming!