MA/CSSE 473 Day 17

Divide-and-conquer Convex Hull

Strassen's Algorithm: Matrix Multiplication

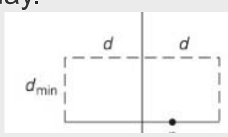
(if time, Shell's Sort)

MA/CSSE 473 Day 17

- **Student Questions**
- Exam 2 specification
- Levitin 3rd Edition Closest Pairs algorithm
- Convex Hull (Divide and Conquer)
- Matrix Multiplication (Strassen)
- Shell's Sort (a.k.a. shellsort)

Levitin 3rd edition Closest Pair Algorithm

- Sorting by both X and Y coordinates happens *once*, before the recursive calls are made.
- When doing the comparisons in the inner loop, we compare all points that are in "y within d" range, not just those on opposite sides of the median line.
- Simpler but more distances to calculate than in what I presented on Friday.



ALGORITHM *EfficientClosestPair*(P, Q)

//Solves the closest-pair problem by divide-and-conquer
//Input: An array P of $n \geq 2$ points in the Cartesian plane sorted in
// nondecreasing order of their x coordinates and an array Q of the
// same points sorted in nondecreasing order of the y coordinates
//Output: Euclidean distance between the closest pair of points
if $n \leq 3$
  return the minimal distance found by the brute-force algorithm
else
  copy the first $\lceil n/2 \rceil$ points of P to array P_l
  copy the same $\lceil n/2 \rceil$ points from Q to array Q_l
  copy the remaining $\lfloor n/2 \rfloor$ points of P to array P_r
  copy the same $\lfloor n/2 \rfloor$ points from Q to array Q_r
  $d_l \leftarrow$ *EfficientClosestPair*(P_l, Q_l)
  $d_r \leftarrow$ *EfficientClosestPair*(P_r, Q_r)
  $d \leftarrow \min\{d_l, d_r\}$
  $m \leftarrow P[\lceil n/2 \rceil - 1].x$
  copy all the points of Q for which $|x - m| < d$ into array $S[0..num - 1]$
  $dminsq \leftarrow d^2$
  for $i \leftarrow 0$ **to** $num - 2$ **do**
    $k \leftarrow i + 1$
    while $k \leq num - 1$ **and** $(S[k].y - S[i].y)^2 < dminsq$
      $dminsq \leftarrow \min((S[k].x - S[i].x)^2 + (S[k].y - S[i].y)^2, dminsq)$
      $k \leftarrow k + 1$
return $sqrt(dminsq)$

A fast algorithm for solving the Convex Hull problem

QUICKHULL

Convex Hull Problem

- Again, sort by x-coordinate, with tie going to larger y-coordinate.

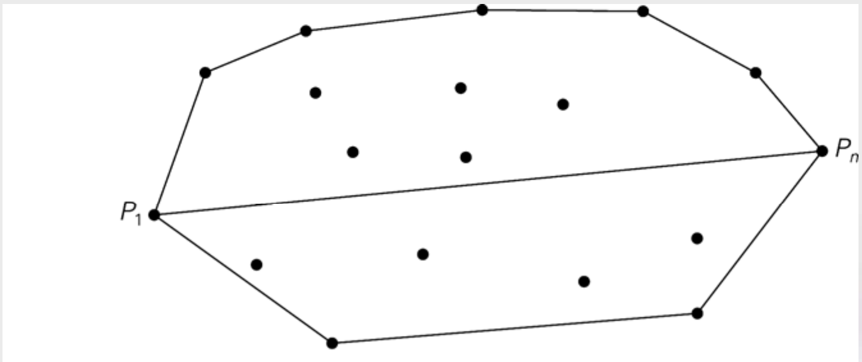


FIGURE 4.8 Upper and lower hulls of a set of points

Recursive calculation of Upper Hull

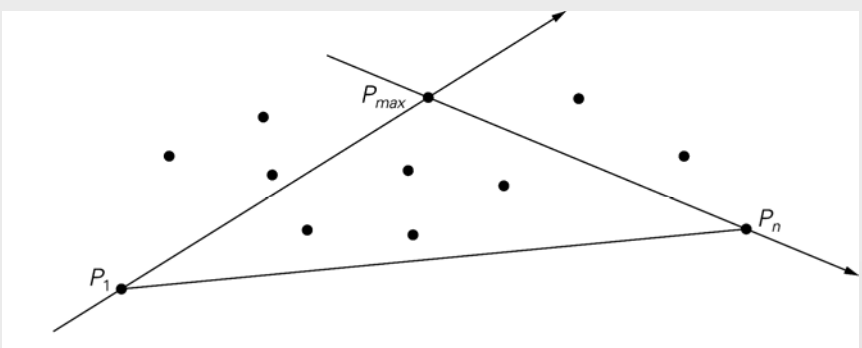


FIGURE 4.9 The idea of quickhull

Simplifying the Calculations

We can simplify two things at once:

- Finding the distance of P from line $P_1P_{2, \text{and}}$
- Determining whether P is "to the left" of P_1P_2
  - The area of the triangle through $P_1=(x_1,y_1)$, $P_2=(x_2,y_2)$, and $P_3=(x_3,y_e)$ is ½ of the absolute value of the determinant

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = x_1y_2 + x_3y_1 + x_2y_3 - x_3y_2 - x_2y_1 - x_1y_3$$

    - For a proof of this property, see http://mathforum.org/library/drmath/view/55063.html
    - How do we use this to calculate distance from P to the line?
  - The sign of the determinant is positive if the order of the three points is clockwise, and negative if it is counter-clockwise
    - Clockwise means that P_3 is "to the left" of directed line segment P_1P_2
- Speeding up the calculation

Efficiency of quickhull algorithm

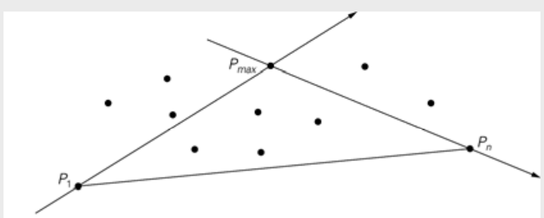


FIGURE 4.9 The idea of quickhull

- What arrangements of points give us worst case behavior?
- Average case is much better. Why?

Strassen's Divide-and-conquer algorithm

FASTER MATRIX MULTIPLICATION

Ordinary Matrix Multiplication

How many additions and multiplications are needed to compute the product of two 2x2 matrices?

$$\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} * \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}$$

Strassen's Matrix Multiplication

Strassen observed [1969] that the product of two matrices can be computed as follows:

$$\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} * \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}$$

$$= \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{bmatrix}$$

Values of $M_1, M_2, \ldots, M_7$ are on the next slide

Formulas for Strassen's Algorithm

$M_1 = (A_{00} + A_{11}) * (B_{00} + B_{11})$

$M_2 = (A_{10} + A_{11}) * B_{00}$

$M_3 = A_{00} * (B_{01} - B_{11})$

$M_4 = A_{11} * (B_{10} - B_{00})$

$M_5 = (A_{00} + A_{01}) * B_{11}$

$M_6 = (A_{10} - A_{00}) * (B_{00} + B_{01})$

$M_7 = (A_{01} - A_{11}) * (B_{10} + B_{11})$

How many additions and multiplications?

The Recursive Algorithm

- We multiply square matrices whose size is a power of 2 (if not, pad with zeroes)
- Break up each matrix into four N/2 x N/2 submatrices.
- Recursively multiply the parts.
- How many additions and multiplications?
  - If we do "normal matrix multiplication" recursively using divide and conquer?
  - If we use Strassen's formulas?

Analysis of Strassen's Algorithm

If N is not a power of 2, matrices can be padded with zeros.

Number of multiplications:

$$M(N) = 7M(N/2) + C, \quad M(1) = 1$$

Solution: $M(N) = \Theta(N^{\log_2 7}) \approx N^{2.807}$ vs. N^3 of brute-force algorithm.

What if we also count the additions?

Algorithms with better asymptotic efficiency are known but they are even more complex.

This is not a divide-and-conquer algorithm.
Today just seemed like a time when we might have a few minutes in which to discuss this interesting sorting technique

Insertion Sort on Steroids

SHELL'S SORT (A.K.A. SHELLSORT)

Insertion sort

- For what kind of arrays is insertion sort reasonably fast?
- What is the main speed problem with insertion sort in general?
- Shell's Sort is an attempt to improve that.

Shell's Sort

- We use the following gaps: 7, then 3, then 1 (last one must always be 1):

21	98	47	32	61	14	83	11	51	40	9	18	71	63	90	77	44	66	12	55	4	49	81	60	41	22	15	68	2	34
Sort first 7th using insertion sort:																													
21	98	47	32	61	14	83	**11**	51	40	9	18	71	63	**90**	77	44	66	12	55	4	**49**	81	60	41	22	15	68	**2**	34
Insert 11																													
11	98	47	32	61	14	83	**21**	51	40	9	18	71	63	**90**	77	44	66	12	55	4	**49**	81	60	41	22	15	68	**2**	34
Insert 90 (nothing moves), then insert 49																													
11	98	47	32	61	14	83	**21**	51	40	9	18	71	63	**49**	77	44	66	12	55	4	**90**	81	60	41	22	15	68	**2**	34
Insert 2																													
2	98	47	32	61	14	83	**11**	51	40	9	18	71	63	**21**	77	44	66	12	55	4	**49**	81	60	41	22	15	68	**90**	34
Note that shaded numbers are now much closer to their final positions.																													

- Next, do the same thing for the next group of 7ths

Shell's sort 2

On to the next group of 7's:																													
2	**98**	47	32	61	14	83	11	**51**	40	9	18	71	63	21	**77**	44	66	12	55	4	49	**81**	60	41	22	15	68	90	**34**
After sorting each group of 7:																													
2	**34**	47	32	61	14	83	11	**51**	40	9	18	71	63	21	**77**	44	66	12	55	4	49	**81**	60	41	22	15	68	90	**98**
2	34	**40**	32	61	14	83	11	51	**44**	9	18	71	63	21	77	**47**	66	12	55	4	49	81	**60**	41	22	15	68	90	98
2	34	40	**9**	61	14	83	11	51	44	**32**	18	71	63	21	77	47	**41**	12	55	4	49	81	60	**66**	22	15	68	90	98
2	34	40	9	**12**	14	83	11	51	44	32	**18**	71	63	21	77	47	41	**22**	55	4	49	81	60	66	**61**	15	68	90	98
2	34	40	9	12	**14**	83	11	51	44	32	18	**15**	63	21	77	47	41	22	**55**	4	49	81	60	66	61	**71**	68	90	98
2	34	40	9	12	14	**4**	11	51	44	32	18	15	**63**	21	77	47	41	22	55	**68**	49	81	60	66	61	71	**83**	90	98
Done with the gap of 7 Still more numbers are closer to where they will end up.																													
What is the worst-case number of comparisons for this phase?																													

Shell's sort 3

Next: Gap of 3:																													
2	34	40	**4**	12	14	**9**	11	51	**15**	32	18	**22**	63	21	**44**	47	41	**49**	55	68	**66**	81	60	**77**	61	71	**83**	90	98
2	**11**	40	4	**12**	14	9	**32**	51	15	**34**	18	22	**47**	21	44	**55**	41	49	**61**	68	66	**63**	60	77	**81**	71	83	**90**	98
2	11	**14**	4	12	**18**	9	32	**21**	15	34	**40**	22	47	**41**	44	55	**51**	49	61	**60**	66	63	**68**	77	81	**71**	83	90	**98**
Finally we do a regular insertion sort, but notice that there will be very little movement.																													

- Why bother if we are going to do a regular insertion sort at the end anyway?
- Analysis?

Code from Weiss book

```
/**
 * Shellsort, using a sequence suggested by Gonnet.
 */
public static <AnyType extends Comparable<? super AnyType>>
void shellsort( AnyType [ ] a )
{
    for( int gap = a.length / 2; gap > 0;
                 gap = gap == 2 ? 1 : (int) ( gap / 2.2 ) )
        for( int i = gap; i < a.length; i++ )
        {
            AnyType tmp = a[ i ];
            int j = i;

            for( ; j >= gap && tmp.compareTo( a[j-gap] ) < 0; j -= gap )
                a[ j ] = a[ j - gap ];
            a[ j ] = tmp;
        }
}
```