

# CS170 Fall 2007 Midterm 1 Review

October 8, 2007

## 1 RSA

Suppose Christos' public RSA key is  $(55, 3)$ . You would like to send the message  $m = 2$  (the code for "The whole class gets an A!") to the registrar's office so that it looks digitally signed by Christos. What message would you send?

## 2 RSA Solution

We know that digital signing requires us to encrypt a message with our private key so that others are able to decrypt it with our public key in order to verify that it came from us.

Since we have  $N = 55$ , we conclude that  $p = 5$  and  $q = 7$ ; therefore,  $d \equiv 3^{-1} \pmod{4 \cdot 10}$ ; therefore, we conclude that  $d = 27$ .

So now all we have to do is encrypt  $m = 2$  with the private key  $(55, 2)$ ; so what we have is

$$\begin{aligned} 2^{27} \pmod{55} &\equiv 8^9 \pmod{55} \\ &\equiv 8 \cdot 64^4 \pmod{55} \\ &\equiv 8 \cdot 9^4 \pmod{55} \\ &\equiv 8 \cdot 81^2 \pmod{55} \\ &\equiv 8 \cdot 26^2 \pmod{55} \\ &\equiv 8 \cdot 16 \pmod{55} \\ &\equiv 128 \pmod{55} \\ &\equiv 18 \pmod{55} \end{aligned}$$

So, our digitally signed message is 18.

## 3 Divide and Conquer

We would like to find a divide-and-conquer algorithm for the following problem:

CLOSEST PAIR:

*Input:* A set of points in the plane,  $\{p_1 = (x_1, y_1), \dots, p_n = (x_n, y_n)\}$

*Output:* The closest pair of points: that is, the pair  $p_i \neq p_j$  for which the distance between  $p_i$  and  $p_j$ , that is

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

is minimized.

For simplicity, assume that  $n$  is a power of two, and that all the  $x$ -coordinates  $x_i$  are distinct, as well as the  $y$ -coordinates.

Here's a high-level overview of the algorithm:

- Find a value  $x$  for which exactly half the points have  $x_i < x$  and half have  $x_i > x$ . On this basis, split the points into two groups,  $L$  and  $R$ .
  - Recursively find the closest pair in  $L$  and in  $R$ . Say these pairs are  $p_L, q_L \in L$  and  $p_R, q_R \in R$ , with distances  $d_L$  and  $d_R$  respectively. Let  $d$  be the smaller of these two distances.
  - It remains to be seen whether there is a point in  $L$  and a point in  $R$  that are less than distance  $d$  from each other. To this end, discard all points with  $x_i < x - d$  or  $x_i > x + d$  and sort the remaining points by  $y$ -coordinate.
  - Now go through this sorted list, and for each point, compute its distance the *seven* subsequent points in the list. Let  $p_M, q_M$  be the closest pair found in this way.
  - The answer is one of the three pairs  $\{p_L, q_L\}, \{p_R, q_R\}, \{p_M, q_M\}$ , whichever is closest.
- (a) In order to prove the correctness of this algorithm, start by showing the following property: any square of size  $d \times d$  in the plane contains at most four points of  $L$ .
- (b) Now show the algorithm is correct. The only case which needs careful consideration is when the closest pair is split between  $L$  and  $R$ . To do this, consider each of the three cases (our closest pair is in  $L$ , our closest pair is in  $R$ , our closest pair is split between  $L$  and  $R$  and show that our algorithm will discover it and name it the closest pair).
- (c) Write down the pseudocode for the algorithm, find its recurrence relation, and calculate its running time.

## 4 Divide and Conquer Solution

- a) Suppose 5 or more points in  $L$  are found in a square of size  $d \times d$ . Divide the square into 4 smaller squares of size  $\frac{d}{2} \times \frac{d}{2}$ . At least one pair of points must fall within the same smaller square: these two points will then be at distance at most  $\frac{d}{\sqrt{2}} < d$ , which contradicts the assumption that every pair of points in  $L$  is at distance at least  $d$ .
- b) The proof is by induction on the number of points. The algorithm is trivially correct for two points, so we may turn to the inductive step. Suppose we have  $n$  points and let  $(p_s, p_t)$  be the closest pair. There are three cases.  
If  $p_s, p_t \in L$ , then  $(p_s, p_t) = (p_L, q_L)$  by the inductive hypothesis and all the other pairs tested

by the algorithm are at a larger distance apart, so the algorithm will correctly output  $(p_s, p_t)$ . The same reasoning holds if  $p_s, p_t \in R$ .

If  $p_s \in L$  and  $p_t \in R$ , the algorithm will be correct as long as it tests the distance between  $p_s$  and  $p_t$ . Because  $p_s$  and  $p_t$  are at distance smaller than  $d$ , they will belong to the strip of points with  $x$ -coordinate in  $[x - d, x + d]$ . Suppose that  $y_s \leq y_t$ . A symmetric construction applies in the other case. Consider the rectangle  $S$  with vertices  $(x - d, y_s), (x - d, y_s + d), (x + d, y_s + d), (x + d, y_s)$ . Notice that both  $p_s$  and  $p_t$  must be contained in  $S$ . Moreover, the intersection of  $S$  with  $L$  is a square of size  $d \times d$ , which, by a), can contain at most 4 points, including  $p_s$ . Similarly, the intersection of  $S$  with  $R$  can also contain at most 4 points, including  $p_t$ . Because the algorithm checks the distance between  $p_s$  and the 7 points following  $p_s$  in the  $y$ -sorted list of points in the middle strip, it will check the distance between  $p_s$  and all the points of  $S$ . In particular, it will check the distance between  $p_s$  and  $p_t$ , as required for the correctness of the algorithm.

- c) When called on input of  $n$  points this algorithm first computes the median  $x$  value in  $O(n)$  and then splits the list of points into those belonging to  $L$  and  $R$ , which also takes time  $O(n)$ . Then the algorithm can recurse on these two subproblems, each over  $n/2$  points. Once these have been solved the algorithm sorts the points in the middle strip by  $y$  coordinate, which takes time  $O(n \log n)$  and then computes  $O(n)$  distances, each of which can be calculated in constant time. Hence the running time is given by the recursion  $T(n) = 2T(\frac{n}{2}) + O(n \log n)$ . This can be analyzed as in the proof of the Master theorem. The  $k$ th level of the recursion tree will contribute  $t_k = 2^k \frac{n}{2^k} (\log n - k)$ . Hence, the total running time will be:

$$\sum_{k=0}^{\log n} t_k = n \log^2 n - n \sum_{k=0}^{\log n} k \leq n \log^2 n - \frac{n}{2} \log^2 n = O(n \log^2 n)$$

## 5 FFT

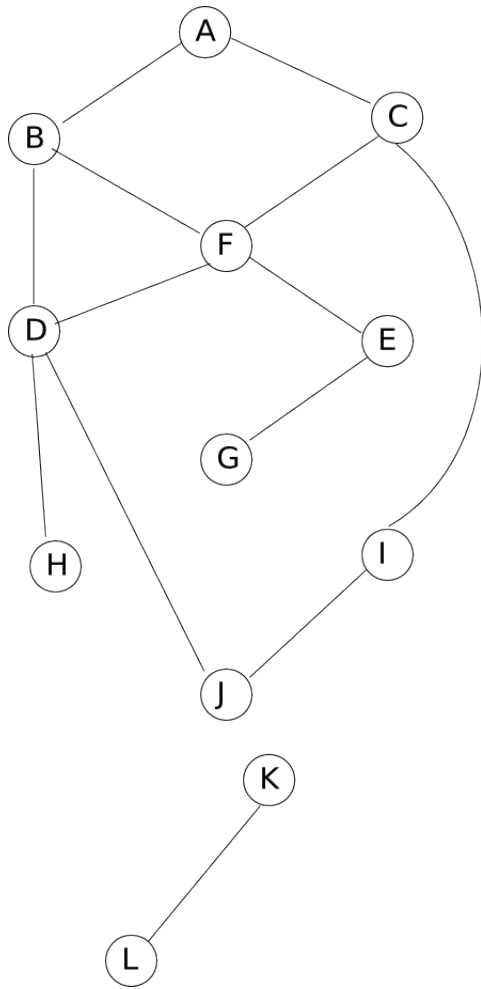
Find the FFT of  $(1, 1, 1, 1, 1, 1, 0, 0)$ .

## 6 FFT Solution

The easiest way to calculate the FFT is by direct evaluation; notice that  $(1, 1, 1, 1, 1, 1, 0, 0)$  corresponds with the polynomial  $f(x) = 1 + x + x^2 + x^3 + x^4 + x^5$ ; since  $n = 8$ , we pick the eighth root of unity,  $\omega = e^{2\pi i/8}$ . So, we conclude that the FFT is  $(f(1), f(\omega), f(\omega^2), f(\omega^3), f(\omega^4), f(\omega^5), f(\omega^6), f(\omega^7))$  (you would be expected to evaluate this on the midterm).

## 7 DFS and BFS

Run a DFS starting from A on the following graph (it is all one graph, just not connected), tracking pre and post numbers. Run a BFS on the graph starting from A, and track the order in which you visit nodes.



## 8 DFS and BFS Solution

DFS: A(1,20), B(2,19), D(3,18), F(4,15), C(5,10), I(6,9), J(7,8), E(11,14), G(12,13), H(16,17), K(21,24), L(22,23)

BFS: A, B, C, D, F, I, H, J, E, G

## 9 DAG (3.24)

Give a linear-time algorithm to determine if a directed acyclic graph  $G$  contains a directed path that touches every vertex exactly once.

## 10 DAG (3.24) Solution

Start by linearizing the DAG. Since the edges can only go in the increasing direction in the linearized order, and the required path must touch all the vertices, we simply check if the DAG has an edge

$(i, i+1)$  for every pair of consecutive vertices labelled  $i$  and  $i + 1$  in the linearized order. Both, linearization and checking outgoing edges from every vertex, take linear time and hence the total running time is linear.

## 11 Dijkstra Extension

Often there are multiple shortest paths between two nodes of a graph. Modify Dijkstra's algorithm so that it computes the shortest path and tracks the number of distinct shortest paths from a start node  $s$  to all nodes, on a graph with positive weights.

## 12 Dijkstra Extension Solution

The number of shortest paths to a node  $v$  will depend on the number of path  $u$  (which is  $prev(v)$ ). If this is a new shortest path, then the number of paths to  $u$  is the number of paths to  $v$  (since there is only one path from  $u$  to  $v$ ). If the shortest path via  $u$  is the same as the existing shortest path to  $v$ , then the number of paths to  $v$  is incremented by the number of paths to  $u$ .

```
if dist(v) > dist(u) + l(u,v)
    numpaths(v) = numpaths(u)
    dist(v) = dist(u) + l(u,v)
if dist(v) = dist(u) + l(u,v)
    numpaths(v) += numpaths(u)
```