

# CS513 Design and Analysis of Algorithms

## Fall 2008 - HW4 solution

Instructor: S. Muthukrishnan

TA: André Madeira

### Problem 1:

- (a) We use Strassen's idea as in class. Let  $X$  and  $Y$  be divided into three groups of  $n/3$  size each, or  $X = \sum_{i=0}^2 2^{in/3} x_i$  and  $Y = \sum_{i=0}^2 2^{in/3} y_i$ . Then,  $XY = x_2 y_2 2^{4n/3} + (x_2 y_1 + x_1 y_2) 2^{3n/3} + (x_2 y_0 + x_1 y_1 + x_0 y_2) 2^{2n/3} + (x_1 y_0 + x_0 y_1 + 1) 2^{n/3} + x_0 y_0$ . Now, observe that given  $x_2 y_2$ ,  $x_1 y_1$ ,  $x_0 y_0$ ,  $(x_2 + x_1)(y_2 + y_1)$ ,  $(x_2 + x_0)(y_2 + y_0)$ , and  $(x_1 + x_0)(y_1 + y_0)$ , one can perform all the above multiplications. Example:  $(x_2 y_1 + x_1 y_2) = (x_2 + x_1)(y_2 + y_1) - x_2 y_2 - x_1 y_1$ . Thus, we have  $T(n) = 6T(n/3) + O(n)$  and the M.T. tells us that  $T = O(n^{\log_3 6})$ , which is a little slower than the two-way Strassen multiplication, which is  $T = O(n^{\log_2 3})$ .
- (b) There are many matrix multiplication algorithms out there. Closer to our class, check out the matrix multiplication algorithm by Strassen at [http://en.wikipedia.org/wiki/Strassen\\_algorithm](http://en.wikipedia.org/wiki/Strassen_algorithm).
- (c) This is a classical application of binary search. One just needs to note that for a given position  $1 \leq i \leq n$  in the array, if  $A[i] > i$  then  $\forall j > i$ ,  $A[j] > j$ , as the elements are distinct and the array is sorted (w.l.o.g. in ascending order). A similar argument holds whenever  $A[i] < i$ . The algorithm is then clear, i.e., return if  $A[i] = i$  otherwise recurse on the appropriate half. The running time is then  $O(\log n)$ .

---

**Problem 2:** Consider:

$$H_k v = \begin{bmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} \end{bmatrix} \cdot \begin{bmatrix} U \\ L \end{bmatrix} = \begin{bmatrix} H_{k-1}U + H_{k-1}L \\ H_{k-1}U - H_{k-1}L \end{bmatrix}$$

Thus, to compute  $H_k v$  we only need to compute two subproblems  $H_{k-1}U$  and  $H_{k-1}L$  of size  $n/2$  each. Since all basic arithmetic operations on matrix/vector elements take  $O(1)$  time, vector additions and subtractions of size  $n$  take time  $O(n)$ . We then have a running time  $T(n) = 2T(n/2) + O(n) = O(n \log n)$  by the M.T.

---

**Problem 3:** Divide the points  $S$  into two sets  $S_1, S_2$  by a vertical line  $\ell$  defined by the median  $x$ -coordinate in  $S$ . Let  $\delta = \min(\delta_1, \delta_2)$ , where  $\delta_1$  and  $\delta_2$  are the closest pair distance of sets  $S_1$  and  $S_2$ . Compute  $\delta$  recursively using sets  $S_1$  and  $S_2$ . Note, however, that the closest pair could be points  $(p, q)$  such that  $p \in S_1$  and  $q \in S_2$ . We are only interested in those points whose distance is less than  $\delta$ . This implies that the points  $p$  and  $q$  should be at most a distance  $\delta$  from  $\ell$ . Unfortunately, all points could satisfy this condition and thus require a  $O(n^2)$  calculation. Fortunately, however, we can exploit the fact that all coordinates are distinct to limit the number of points we need to look at.

Now, note that all points  $q \in S_2$  that are within distance  $\delta$  for a given  $p \in S_1$  must lie in a  $\delta \times 2\delta$  rectangle. Observe that there can be at most 6 points in that rectangle by the definition of  $\delta$ . In other words, if there were additional points, then one point cannot be at the extremity and  $\delta_2 < \delta$ , a contradiction. Therefore, we should perform only  $6 \times n/2$  distance comparisons, as in the worst case  $n/2$  elements are within distance  $\delta$  of  $\ell$ .

We proceed as follows. Let  $P_1$  and  $P_2$  be the points within distance  $\delta$  of  $\ell$  in  $S_1$  and  $S_2$  respectively. Project all points in  $P_2$  onto line  $\ell$ . Then, for a given point  $p \in P_1$ , pick out at most 6 points whose projection is within  $\delta$  of  $p$  (can you see why?). This can be done by performing a linear search in sorted  $P_1$  and  $P_2$ , which can be achieved by sorting  $S_1$  and  $S_2$  as a pre-computation by your favorite  $O(n \log n)$  sorting algorithm. Therefore, the recursive procedure takes  $T(n) = 2T(n/2) + O(n) = O(n \log n)$  and the total time is thus  $O(n \log n)$ .

---

**Problem 4: (Extra Credit).** As the hint suggests, we compute  $AB + BA$  using only squaring as follows. Note that  $AB + BA = [A + B]^2 - A^2 - B^2$ . Matrix additions takes time  $O(n^2)$  and since  $T(n) = \Omega(n^2)$ , we have that  $AB + BA$  can be computed in  $O(T(n))$ . Now, consider the  $2n \times 2n$  matrices

$$A = \begin{bmatrix} 0 & X \\ 0 & 0 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0 & 0 \\ Y & 0 \end{bmatrix}.$$

Then, we have

$$AB + BA = \begin{bmatrix} XY & 0 \\ 0 & YX \end{bmatrix}.$$

Thus  $XY$  can be computed in  $O(T(n))$  time.