

# 091M4041H - Assignment 1

## Algorithm Design and Analysis

September 28, 2017

Notice:

1. Please submit your answer in hard copy AND submit a digital version to UCAS website <http://sep.ucas.ac.cn>.
2. Hard copy should be submitted before 9 am. October 13 and digital version should be submitted before 12 pm. October 13.
3. You can choose **three** from problems 1-7, and choose **two** from problems 8-10.
4. For problems 1-7, you should do at least the following things:
  - (a) Describe your algorithm in natural language **AND** pseudo-code;
  - (b) Draw a “subproblem reduction graph”, where nodes represent subproblems, and edges describe the “reduction relationship” between them for every problem you choose in problems 1-7;
  - (c) Prove the correctness of your algorithm;
  - (d) Analyse the complexity of your algorithm.
5. For problems 8-10, you should implement your algorithm in C/C++/Java/Python with good comments.

## 1 Divide and Conquer

You are interested in analyzing some hard-to-obtain data from two separate databases. Each database contains  $n$  numerical values, so there are  $2n$  values total and you may assume that no two values are the same. You'd like to determine the median of this set of  $2n$  values, which we will define here to be the  $n^{th}$  smallest value.

However, the only way you can access these values is through *queries* to the databases. In a single query, you can specify a value  $k$  to one of the two databases, and the chosen database will return the  $k^{th}$  smallest value that it contains. Since queries are expensive, you would like to compute the median using as few queries as possible.

Give an algorithm that finds the median value using at most  $O(\log n)$  queries.

## 2 Divide and Conquer

Find the  $k^{th}$  largest element in an unsorted array. Note that it is the  $k$ th largest element in the sorted order, not the  $k^{th}$  distinct element.

INPUT: An unsorted array  $A$  and  $k$ .

OUTPUT: The  $k^{th}$  largest element in the unsorted array  $A$ .

## 3 Divide and Conquer

Consider an  $n$ -node complete binary tree  $T$ , where  $n = 2^d - 1$  for some  $d$ . Each node  $v$  of  $T$  is labeled with a real number  $x_v$ . You may assume that the real numbers labeling the nodes are all distinct. A node  $v$  of  $T$  is a *local minimum* if the label  $x_v$  is less than the label  $x_w$  for all nodes  $w$  that are joined to  $v$  by an edge.

You are given such a complete binary tree  $T$ , but the labeling is only specified in the following *implicit* way: for each node  $v$ , you can determine the value  $x_v$  by *probing* the node  $v$ . Show how to find a local minimum of  $T$  using only  $O(\log n)$  *probes* to the nodes of  $T$ .

## 4 Divide and Conquer

Suppose now that you're given an  $n \times n$  grid graph  $G$ . (An  $n \times n$  grid graph is just the adjacency graph of an  $n \times n$  chessboard. To be completely precise, it is a graph whose node set is the set of all ordered pairs of natural numbers  $(i, j)$ , where  $1 \leq i \leq n$  and  $1 \leq j \leq n$ ; the nodes  $(i, j)$  and  $(k, l)$  are joined by an edge if and only if  $|i - k| + |j - l| = 1$ .)

We use some of the terminology of problem 3. Again, each node  $v$  is labeled by a real number  $x_v$ ; you may assume that all these labels are distinct. Show how to find a local minimum of  $G$  using only  $O(n)$  probes to the nodes of  $G$ . (Note that  $G$  has  $n^2$  nodes.)

## 5 Divide and Conquer

Given a convex polygon with  $n$  vertices, we can divide it into several separated pieces, such that every piece is a triangle. When  $n = 4$ , there are two different ways to divide the polygon; When  $n = 5$ , there are five different ways.

Give an algorithm that decides how many ways we can divide a convex polygon with  $n$  vertices into triangles.

## 6 Divide and Conquer

Recall the problem of finding the number of inversions. As in the course, we are given a sequence of  $n$  numbers  $a_1, \dots, a_n$ , which we assume are all distinct, and we define an inversion to be a pair  $i < j$  such that  $a_i > a_j$ .

We motivated the problem of counting inversions as a good measure of how different two orderings are. However, one might feel that this measure is too sensitive. Let's call a pair a *significant inversion* if  $i < j$  and  $a_i > 3a_j$ . Given an  $O(n \log n)$  algorithm to count the number of significant inversions between two orderings.

## 7 Divide and Conquer

A group of  $n$  ghostbusters is battling  $n$  ghosts. Each ghostbuster is armed with a proton pack, which shoots a stream at a ghost, eradicating it. A stream goes in a straight line and terminates when it hits the ghost. The ghostbusters decide upon the following strategy. They will pair off with the ghosts, forming  $n$  ghostbuster-ghost pairs, and then simultaneously each ghostbuster will shoot a stream at his chosen ghost. As we all know, it is very dangerous to let streams cross, and so the ghostbusters must choose pairings for which no streams will cross. Assume that the position of each ghostbuster and each ghost is a fixed point in the plane and that no three positions are collinear.

1. Show that there exists a line passing through one ghostbuster and one ghost such the number of ghostbusters on one side of the line equals the number of ghosts on the same side. Describe how to find such a line in  $O(n \log n)$  time.
2. Give an  $O(n^2 \log n)$ -time algorithm to pair ghostbusters with ghosts in such a way that no streams cross.

## 8 Divide and Conquer

The attached file Q8.txt contains 100,000 integers between 1 and 100,000 (each row has a single integer), the order of these integers is random and no integer is repeated.

1. Write a program to implement the Sort-and-Count algorithms in your favorite language, find the number of inversions in the given file.
2. In the lecture, we count the number of inversions in  $O(n \log n)$  time, using the Merge-Sort idea. Is it possible to use the Quick-Sort idea instead ?  
If possible, implement the algorithm in your favourite language, run it over the given file, and compare its running time with the one above. If not, give a explanation.

## 9 Divide and Conquer

Implement the algorithm for the closest pair problem in your favourite language.

INPUT:  $n$  points in a plane.

OUTPUT: The pair with the least Euclidean distance.

## 10 Divide and Conquer

Implement the Strassen algorithm for MatrixMultiplication problem in your favourite language, and compare the performance with grade-school method.