Massachusetts Institute of Technology — Handout 7
6.046J/18.410J: Introduction to Algorithms — February 14, 2002
Professors Michel Goemans and Piotr Indyk

Problem Set 2

This problem set is due in class on *Thursday, February 28, 2002*.

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material and will be useful in solving the assessed problems. You are responsible for material covered by the exercises. Each problem is to be done on a separate sheet (or sheets) of paper. Mark the top of each sheet with your name, 6.046J/18.410J, the problem number, your recitation section, the date, and the names of any students with whom you collaborated.

Exercise 2-1. Do Exercise 7.3-1 in CLRS page 154.

Exercise 2-2. Do Exercise 8.3-2 on page 173. (not the part that was on pset 1)

Exercise 2-3. Do problem 7.4-2 in CLRS page 159.

Problem 2-1. Egyptian Pyramid Architecture

Khufru the Ancient Egyptian believed it was very important to know how pyramids would affect his skyline. Having recently graduated from the Memphis Institute of Technology, he decided he would create an algorithm for predicting a pyramid skyline.

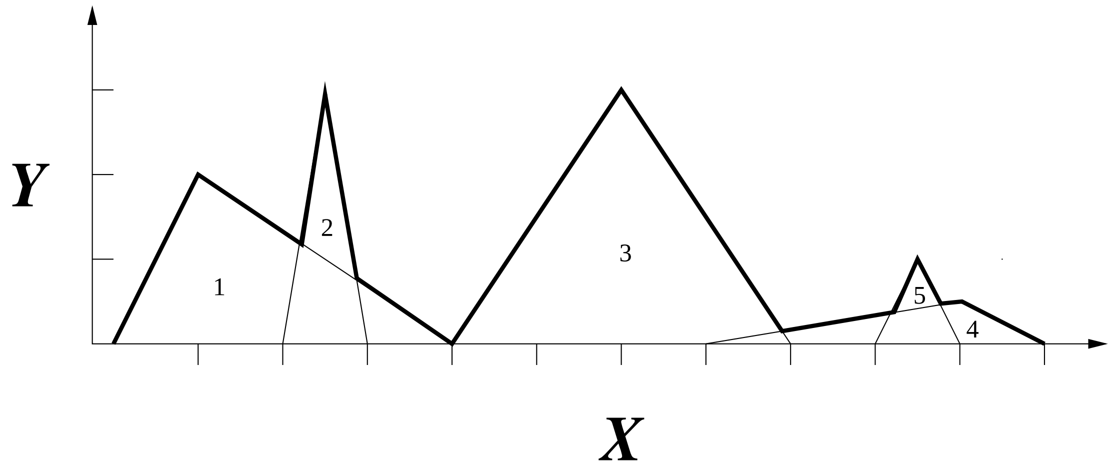

Figure 1: Egyptian Pyramid skyline

Unfortunately for Khufru, there were no computers around, there was no numeral for zero, and the Persians hadn't given him the word algorithm yet, so he died a sad, lonely, bitter man.

For this problem we shall attempt to avoid his fate by successfully coming up with an efficient way to predict in two dimensions what the skyline will look like. Examine figure 1 For a sample skyline. This skyline is formed from the following five pyramids (triangles):

Pyramid	left	top	right
Pyramid 1	$(0, 0)$	$(1, 2)$	$(4, 0)$
Pyramid 2	$(2, 0)$	$(2.5, 3)$	$(3, 0)$
Pyramid 3	$(4, 0)$	$(6, 3)$	$(8, 0)$
Pyramid 4	$(7, 0)$	$(10, 1)$	$(11, 0)$
Pyramid 5	$(9, 0)$	$(9.5, 1)$	$(10, 0)$

Note the resulting skyline in the figure. The final algorithm we design should give a list of points (x_i, y_i) from left to right (effectively sorted by x) that describe the skyline (the bold part of the figure). In the example, we want the result to be:

$$(0, 0)\ ,\ (1, 2)\ ,\ (2.2, 1.2)\ ,\ (2.5, 3)\ \cdots\ (10, 1)\ ,\ (11, 0)$$

(a) Before he can take on entire cities, Khufru needed to know if there was a way for him to merge two known skylines. So, given two valid skylines, can you design an algorithm that spits out the correct resulting skyline from the 'merging' of both? Provide a running time analysis of your merge algorithm. (You should, of course, try to design an asymptotically fastest such algorithm.)

(b) Now that we know that we can merge two sets of skylines, provide an algorithm that can solve the skyline problem. Argue its correctness, and provide a running-time analysis.

(c) Assume that someone has provided you with a computer that, given a list of pyramids, will spit out a skyline in time $\Theta(n)$, where n is the number of pyramids. Show that you can construct a machine that, using this computer, can sort n integers in time $\Theta(n)$. You may assume that no number is repeated.

Problem 2-2. Verifying Matrix Multiplication

As we have already seen, it is possible to multiply two $n \times n$ matrices in time better than $\Theta(n^3)$. However, even the simplest such algorithm (of Strassen) is a mess. Newer faster algorithms are even messier. John Student thinks he has designed a processor that can do matrix multiplications, but would really like to make sure that his circuitry isn't flawed, so he'd like to run a multiplication on his chip, and then figure out if the chip made a mistake.

This raises an interesting question: is it possible to quickly *verify* if a product of two matrices has been computed correctly ?

Piotr has described to John the following *randomized* algorithm, for checking if $A \times B = C$, where A, B, C are $n \times n$ matrices:

1. Choose a binary vector x uniformly at random from $\{0, 1\}^n$. That is, create a vector that is n values, each of which is a 0 or 1 with equal probability.
2. Check if $(A \times B)x = Cx$

(a) **Complexity:** show that the above algorithm can be implemented with running time $O(n^2)$.

(b) **Correctness I:** if $A \times B = C$, does the algorithm always output YES, for any choice of x?

(c) **Correctness II:** show that if $A \times B \neq C$, then the algorithm outputs NO with probability at least $1/2$.

(d) Given the above information, can you come up with an algorithm that can determine correctly, with probability $1 - (1/2)^k$, whether a given output of the matrix multiplication chip is correct? What is the running time of your algorithm (your running time should include n AND k as variables).

Problem 2-3. Know-it-all

Design an algorithm, that, given n integers in the range $[0..k]$, preprocesses the data and answers any query of the following form:
`how many of the` n `integers belong in the range` $[a..b]$`?`
The algorithm should take no longer than $O(n + k)$ to preprocess and should take $O(1)$ to answer ANY such question. Prove your algorithm is correct.

Problem 2-4. Paranoid Quicksort

Suppose that we are being paranoid about using bad partitions. Therefore we modify our quicksort routine the following way:

After we partition the array (with respect to a randomly chosen x), we then run a check to test if our partition is **good**. We define a **good** partition to be one such that the ratio of the lengths of the two subarrays is between $[1/4, 4]$. If we indeed have a **good** partition, we proceed as before. If it is a bad partition, we instead pick a new x and try again.

(a) Give an expected running-time analysis of the new modified algorithm.

(b) Argue that this algorithm will correctly sort the input in time $c(n \log n)$ for some constant c with probability greater than $1 - \frac{1}{n}$.