

Name \_\_\_\_\_

**Instructions** This exam consists of three problems worth a total of 100 points.

Do Problem (1). From Problems (2) and (3) on divide and conquer, *choose one*. From Problems (4) and (5) on dynamic programming, *choose one*. In other words, do a total of *three* problems. If you do more than three, only three will be graded.

On problems that ask you to *design an algorithm*, be sure to: (i) describe your algorithm using prose and pictures, (ii) argue that your algorithm is correct, and (iii) analyze its running time. Pseudocode is not required. When analyzing the time taken by a recursive algorithm, write down a recurrence and solve it.

You have 75 minutes. The exam is closed book, closed notes, and closed calculator. Good luck!

- (1) **(Short answer)** (10 points) What is the four-step framework for designing dynamic programming algorithms? Give a brief description of each of the steps.

- (2) **(Finding a pair of close elements)** (40 points) For a set  $S$  of  $n$  numbers, a pair of elements  $x, y \in S$ , where  $x < y$ , are said to be *close* if

$$y - x \leq \frac{1}{n-1} (\max S - \min S).$$

Suppose we are given an unsorted array  $A$  of  $n$  distinct numbers, and we want to find a pair of close elements in  $A$ . Using *divide and conquer*, design an algorithm that finds a pair of close elements in  $A$  in  $\Theta(n)$  time.

(This page is blank.)

(This page is blank.)

- (3) **(Largest monochromatic rectangle)** (40 points) Suppose you are given an  $m \times n$  array  $A[1:m, 1:n]$ , where each element  $A[i, j]$  is either 0 or 1. We view array  $A$  as representing a black-and-white image of pixels, where value 0 corresponds to a black pixel, and value 1 corresponds to a white pixel.

Using *divide and conquer*, design an algorithm that finds a subarray  $A[a:b, c:d]$  within  $A$  whose elements are *all black*, and that has the greatest *area*. In other words, find the largest black rectangle in image  $A$ . Your algorithm should run in  $O(mn^2)$  or  $O(nm^2)$  time.

(This page is blank.)

(This page is blank.)

- (4) **(Longest increasing subsequence)** (50 points) Given an array  $A[1 : n]$  of numbers, an *increasing subsequence* of  $A$  is a subsequence of  $A$  whose elements are strictly increasing. More formally, an increasing subsequence of  $A$  is given by a series of indices  $1 \leq i_1 < i_2 < \dots < i_k \leq n$  where

$$A[i_1] < A[i_2] < \dots < A[i_k].$$

The *length* of this subsequence is its number of elements, namely  $k$ .

Using *dynamic programming*, design an algorithm that finds a longest increasing subsequence of  $A$  in  $O(n^2)$  time. Remember to follow the four-step framework for designing dynamic programming algorithms.

(This page is blank.)

(This page is blank.)

- (5) **(Best approximation by lines)** (50 points) Suppose you have an  $x$ - $y$  plot of points that you wish to approximate by a series of straight lines. Formally, given a collection of  $n$  points in the plane,

$$(x_1, y_1), (x_2, y_2), \dots (x_n, y_n),$$

where

$$x_1 \leq x_2 \leq \dots \leq x_n,$$

and an integer  $k \geq 1$ , you wish to find a partition of the points into  $k$  consecutive runs, and fit a line to the points in each run, such that the sum of the errors of the least-squares lines through each of the runs is minimum.

You may assume that you have access to a function  $e(i, j)$  that finds a *single* line for points  $i, i+1, \dots, j$  that has the minimum least-squares error on these points, and that returns its corresponding error. This function  $e(i, j)$  runs in  $\Theta(j-i+1)$  time.

Using *dynamic programming*, design an algorithm that finds the best approximation of the  $n$  points by  $k$  lines in  $O(kn^2)$  time. Remember to follow the four-step framework for designing dynamic programming algorithms.

(This page is blank.)

(This page is blank.)