Isai Lopez Rodas
1605542 ilopezro@ucsc.edu

CMPS 102 — Spring 2020 – Homework 2

Four problems, 10 points each, due Monday April 27 (11:59 PM) on Gradescope.

Before you begin the assignment, please read the following carefully.

- Read the *Homework Guidelines*.
- The assignment consists of four problems worth ten points each. Unless otherwise stated, algorithms and their proofs of correctness are weighted equally.
- Due to class size, it is likely that only a subset of the problems will be graded. We will announce the problem(s) which will be graded after the due date of the assignment.
- Every part of each question begins on a new page. Do not change this.
- This does not mean that you should write a full page for every question. Your answers should be short and precise. Lengthy and wordy answers will lose points.
- Do not change the format of this document. Simply type your answers as directed.
- You are **not** allowed to work in teams.

I have read and agree to the collaboration policy. – Isai Lopez Rodas, ilopezro@ucsc.edu
Collaborators: None

Recommended exercises (not to be turned in)

1. The solved exercises in 5 (Divide and Conquer).

Continued on next page.

Problems to be turned in

1. You are interested in building a coffeeshop and charging station beside a freeway running between two cities. Let n be the number of interchanges between the cities, and number them consecutively so that interchange 0 is at the first city, 1 is the next interchange, and so on, with $n-1$ being the interchange at the other city. We say each interchange i is adjacent to interchanges $i-1$ and $i+1$ (only).

   Each interchange i has a cost $c(i)$ to construct a coffee shop/charging station. Call an interchange i a *local minimum* if (and only if) the cost of building there is less than the costs of building at the adjacent interchanges, i.e. both $c(i) < c(i+1)$ and $c(i) < c(i-1)$. You want to ensure that you build at a local minimum so that no one can build a coffeeshop/charging station more cheaply at an adjacent interchange.

   You can assume that:

   - $n \geq 3$, so there is at least one interchange between the cities.
   - The costs at the cities $c(0)$ and $c(n-1)$ are both known to be the same large number (say $100n$).
   - The other costs are all different and are all strictly less than $c(0)$ (and thus also less than $c(n-1)$).

   The difficulty is that the $c(i)$ values are not known ahead of time (except for $c(0)$ and $c(n-1)$) and it is expensive to determine the cost of building at any interchange.

   The goal is to create a divide and conquer algorithm that finds any interchange k that is a local minimum while examining relatively few of the $c(i)$ values. Although there may be many local minimums, your algorithm need only find one of them.

(a) (2 points) First, prove that a local minimum exists.

Solution.

(b) (3 points) Clearly describe a divide and conquer algorithm for the problem

Algorithm.

(c) (3 points) Prove that your algorithm is correct

Proof of correctness.

(d) (2 points) Analyze the (worst case) number of interchange costs (i.e. $c(i)$ values) examined by your algorithm as a function of n.

Solution.

I have read and agree to the collaboration policy. – FirstName LastName, email@ucsc.edu
Collaborators:

---

2. We all know how to search a sorted 1-dimensional array. This problem involves searching two dimensional arrays that are partially sorted.

   Call a two-dimensional square ($n \times n$) array A *line-sorted* if

   - the entries of A are all distinct numbers,
   - the entries in each row are sorted in ascending order (left to right), and
   - the entries in each column are sorted in ascending order (top to bottom).

   If A is line-sorted then both $A[i-1,j] < A[i,j] < A[i+1,j]$ and $A[i,j-1] < A[i,j] < A[i,j+1]$ (assuming the indices are inside the array).

   Devise a *two-dimensional divide and conquer* algorithm which takes a line-sorted array A and number k to search for, and returns either indices i and j such that $A[i,j] = k$ or an indicator that k is not in the array. Describe your algorithm (in pseudo-code), argue that your algorithm is correct, give and and analyze its (worst-case) running time through a recurrence. You may assume that n is a power of two.

   The goal is for your algorithm to take asymptotically less time than the number of elements in the array, but it will probably not be as efficient as simply performing binary search on each row.

Continued on the next page.

Algorithm.

Proof of correctness.

I have read and agree to the collaboration policy. – FirstName LastName, email@ucsc.edu
Collaborators:

---

3. Problem 3 in Chapter 5 (Divide and Conquer). This is the equivalent back card problem: determine if *more than* half the n possible fraudulent bank cards are from the same account using a machine that detects if two cards are from the same account or not. If this doesn't sound like Problem 3 in Chapter 5 of your version of the text, then let us know.

   Clearly describe your algorithm, prove that it is correct, and analyze its running time. For full credit, the running time should be $O(n \lg n)$.

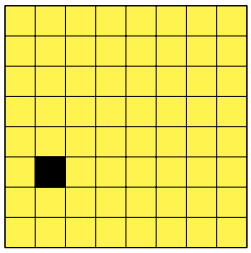
   Algorithm.

Proof of correctness.

I have read and agree to the collaboration policy. – FirstName LastName, email@ucsc.edu
Collaborators:

---

4. Tiling a Room. You have a square room that is 2^k feet on a side (for some $k \geq 1$), so you can think of the room as a grid of $2^k \times 2^k$ cells. One of these cells is used for a floor vent. You are to find a way to fill in the $(2^k \times 2^k) - 1$ other cells perfectly with L-shaped tiles, like the one shown. The L-shaped tiles can be rotated in four different ways, but cannot be cut or extend outside the room. Give a divide and conquer algorithm that finds a tiling and argue that it is correct.



For full credit, your algorithms should take $O(n)$ time where $n = 2^{2k} = (2^k)^2$ is the number of cells in the room. The intent of the problem is to show that rooms of this kind can be tiled, do not worry about the book-keeping needed to record the particular solution found by your divide and conquer algorithm.

Continued on next page.

Algorithm.

Proof of correctness.