CS-203 – PROJECT 2 – 2023 FALL

GIUSEPPE TURINI – KETTERING UNIVERSITY

INTRODUCTION

2D Closest-Pair Problem

The closest-pair problem consists in finding the two closest points (*i.e.*, the closest pair) in a given set S of n distinct points. [1]

This problem can be specified for various different geometric domains: one-dimensional (1D), two-dimensional (2D), three-dimensional (3D), etc. For simplicity, here we consider only the 2D closest-pair problem (see Figure 1).

Additionally, we assume that the 2D points in the input set S are specified by their Cartesian coordinates (x and y) as integer values, and that the distance between two 2D points is measured using the standard Euclidean distance. [2]

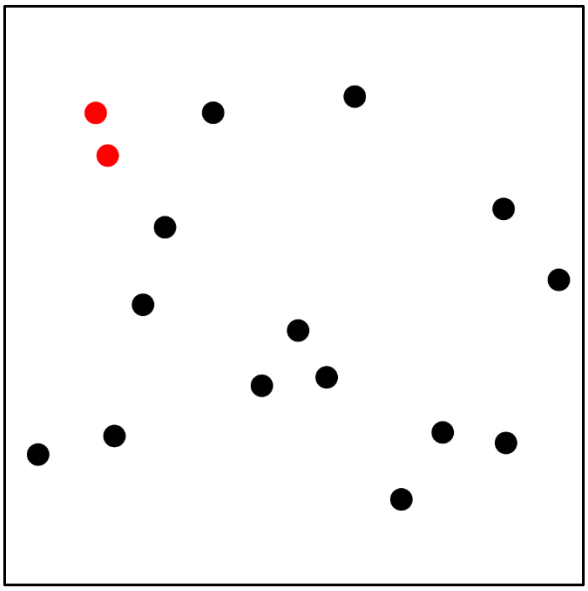

Figure 1. A 2D closest-pair problem, with 16 distinct points and solution in red.

[1] See: Wikipedia – Closest-Pair of Points Problem.

[2] See: Wikipedia – Euclidean Distance.

Solution by Exhaustive Search or Brute Force

The 2D closest-pair problem can be easily solved by an exhaustive search algorithm or a brute force algorithm (these solutions will be discussed and partially implemented in class).

Solution by Divide-and-Conquer

Another approach to solving the 2D closest-pair problem is by using a divide-by-2-and-conquer algorithm. This strategy begins by splitting the input set S into 2 halves, then solves these 2 smaller subproblems, and finally computes the final solution by analyzing the results of the subproblems already solved. [3] [4] [5]

THE PROJECT

Implement a Divide-and-Conquer Algorithm to Solve the 2D Closest-Pair Problem

Write a Java program which, given in input an integer n (n>2): (a) generates n distinct 2D points, (b) finds a solution to the 2D closest-pair problem by divide-and-conquer (see attachment), and (c) prints the result (*i.e.*, closest-pair points coordinates, and the distance of the closest-pair) to standard output.

Analyze the Efficiency of your Algorithm

Analyze your algorithm performing both the theoretical analysis and the empirical analysis to evaluate the time efficiency. Each analysis should include:

- The analysis (description, method, formulas, efficiency class, data, etc.).
- Few visuals illustrating the analysis results (graphs, units, approximation function, etc.).
- Brief conclusions commenting the analysis results (comparison theoretical vs empirical).
- Brief comparison with simpler algorithms/solutions (exhaustive search, brute force, etc.).

Include these analyses in the technical report for this project (~3 pages in total). [6]

[3] Levitin A. *"Introduction to the Design and Analysis of Algorithms (3rd Edition)."* Pearson, § 5.5. 2012.
[4] Johnsonbaugh R. and Schaefer M. *"Algorithms."* Pearson Education. 2004.
[5] Shamos M.I. and Hoey D. *"Closest-Point Problems."* In IEEE SFCS, pp. 151-162. 1975.
[6] Note: the analysis should include the initial sorting of input arrays required by the divide-and-conquer algorithm.

SUBMISSION

Deadline and Procedure

Before Monday of 10th week, send an email to the instructor (gturini@kettering.edu):

- Send the email using your Kettering email account (subject: "CS-203: Project 2").
- The email must include in attachment your project (2 files, see below).
- Late submissions will be assessed a -5% penalty (-5 points) for each day of delay. [7]

Content

The content of your submission must be:

- A Java file ("Solver2DClosestPair.java") with your code.
- A technical report ("CS-203-Project-2-Report.pdf") with your analysis.

[7] Note: the maximum penalty for a late submission is -30% (-30 points).

EVALUATION

This is the form used to grade this project.

PROJECT 2 – EVALUATION FORM

Implementation, Performance, and Analysis (70/100)

Divide-and-Conquer Implementation (25/100)
Proper internal data representation, original algorithm structure, no deviations.

Divide-and-Conquer Performance (15/100)
Proper internal data representation, no re-sorting.

Theoretical Analysis (15/100)
Input size, best- and worst- cases, basic operation counts, efficiency classes.

Empirical Analysis (15/100)
Experiment setup, data collection, data visualization, proper result comparison.

Design, Style, and Submission (30/100)

OOP Design and Compilation Requirements (…/100)
Classes, fields, functions, variables properly designed, no unhandled exceptions.

Coding Style and Commenting (…/100)
File/class/function headers, variables comments, comment/code ratio, naming, indentation.

Submission Procedure and Delay (…/100)
Proper submission (email, subject, attachments), no delay.