CS2521 Assignment

Revision 1.3

Overview and Marking

This project consists of two tasks. The first will count for 60% of the overall project mark, while the second will count for 40%.

The project aims to evaluate your ability to create, analyse, implement and evaluate algorithms.

Hand-in

The hand-in date will be 23:59:59, on the 1st of April 2016. Handing in up to 24 hours late will attract a 10% penalty, while handing in up to a week late will attract a 25% penalty, deducted as a percentage of the mark obtained. Work handed in more than a week late will be treated as a "no paper".

Submission should take place via myAberdeen — upload a *zip* file containing your solutions to Task 1, *and a separate zip file* containing your solutions to Task 2. Different submission areas will be made available for each section. Please submit your reports within the relevant *zip* files (**not RAR, ARC, ARJ, LZH or any other format**), *as PDF documents*. **Reports submitted as word documents, .odt documents, RTF, etc will not be marked.**

Plagiarism

Plagiarism will not be tolerated, it is a serious offence, with punishments ranging from a 0 mark to expulsion. If you are unsure about whether your work counts as plagiarised, please contact me.

1 Ticketing

The GreedyCine movie chain has put you in charge of their ticketing system. Their main priority is making money, and they've decided their current approach to ticketing is not profitable enough. Their system, as it currently stands, involves selling tickets on a first-come first-serve basis. However, since large groups of people would often like to sit together (and will not attend a movie otherwise), many seats are often left empty. Your task is to identify several possible improvements to the way tickets are sold, as described below.

1.1 Formalising the problem

We treat a movie theatre as containing a net total of n seats. You are given a list of groups $\mathcal{G} = [G_1, \ldots, G_m]$ where G_i, $1 \leq i \leq m$ is an integer between 1 and n (inclusive), representing the number of seats desired by group G_i.

1.2 Subtasks

1. Implement the first-come first-serve heuristic, and plot the average number of empty seats that remain with this heuristic with changing n and m. In your report, provide a counter-example showing where

this heuristic will not provide an optimal seating arrangement. You can use the GenGroup.java file to generate groups for this, and all following tasks.

2. Describe and implement an optimal algorithm for seat allocation. Prove its correctness, and analyse its complexity. Provide an empirical evaluation (e.g., in the form of a plot) of its average running time as m and n increases.

3. Describe and implement a heuristic for seat allocation that runs in $\Theta(m \log m)$ time. Prove that if it is possible to admit k people, your heuristic will admit at least $k/2$ people. Provide an empirical evaluation of your heuristic's average running time as m and n increases.

4. Identify a counter-example, and with it show that asymptotically as n gets large, the ratio between your heuristic and the optimal seating allocation approaches $1/2$ (in other words, a perfect algorithm would seat double the people your heuristic would).

5. Describe and implement a heuristic for seat allocation that runs in $\Theta(m)$ time, and prove that for this heuristic, if it is possible to admit k people, it will admit at least $k/2$ people. How would this heuristics cope with the counter-example identified in the previous task? Provide an empirical evaluation of your heuristic's average running time as m and n increases.

1.3 Marking

Marks will be allocated as follows:

Task	Mark breakdown	Total marks for task
1	Implementation 3, plot 3, counter-example 3	9
2	Description/implementation 6, plot 3, complexity analysis 3, proof 3	15
3	Description/Implementation 6, plot 3, proof 3	12
4	Counter example 3, proof 3	6
5	Description: 6, implementation 3, proof 3, plot 3, counter example 3	18

An additional 3 marks will be allocated for the quality of your report (e.g. appropriate citations, additional insights obtained beyond and above what is required above, etc.) and code (comments). This section will be marked out of 60 (with 63 marks available).

2 Convex Hulls

In this section, you will implement a divide and conquer algorithm for finding the convex hull of a set of points and you will analyze the algorithm both theoretically and empirically.

2.1 Background

The convex hull of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior. To be rigorous, a polygon is a piecewise-linear, closed curve in the plane. That is, it is a curve, ending on itself that is formed by a sequence of straight-line segments, called the sides of the polygon. A point joining two consecutive sides is called a vertex of the polygon. If the polygon is simple, as we shall generally assume, it does not cross itself. The set of points in the plane enclosed by a simple polygon forms the interior of the polygon, the set of points on the polygon itself forms its boundary, and the set of points surrounding the polygon forms its exterior. A simple polygon is convex if, given any two

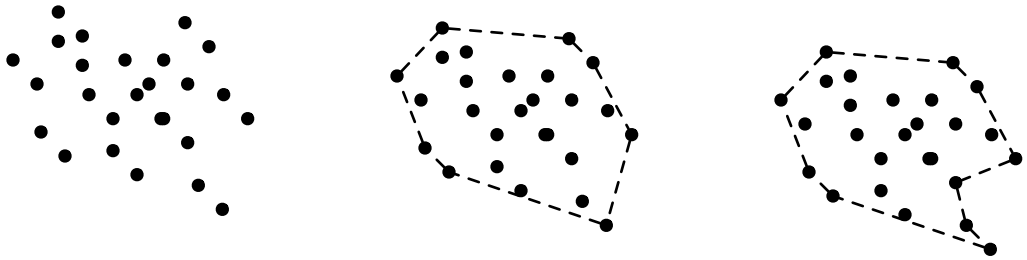

Figure 1: Convex and non-convex hulls.

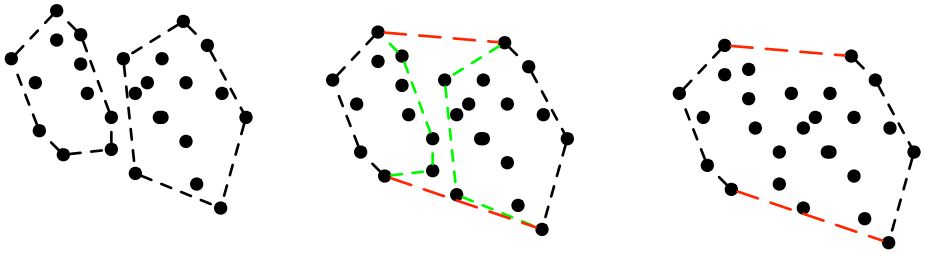

Figure 2: Creating a convex hull.

points on its boundary or in its interior, all points on the line segment drawn between them are contained in the polygon's boundary or interior.

Figure 1 shows how a set of points on the left, the convex hull for this set of points in the center, and a non-convex hull for a set of points on the right.

2.2 Creating a convex hull

You can utilise the following algorithm to find a convex hull:

1. Take the set of n points, and divide it into two subsets, L, containing the leftmost $\lceil n/2 \rceil$ points and R, containing the rightmost $\lfloor n/2 \rfloor$ points.

2. The convex hulls of L and R are computed recursively.

3. To combine the convex hulls of L and R, it is necessary to find two edges known as the upper and lower common tangents. A common tangent of two simple convex polygons is a line segment in the exterior of both polygons intersecting each polygon at a single vertex. If continued infinitely in either direction, the common tangent would not intersect the interior of either polygon.

4. The upper common tangent can be found by scanning aground the left hull in a clockwise direction, and around the right hull in a counterclockwise direction. Some guidance with regard to finding the common tangents is given below; although you will need to work out some additional details. The two tangents divide each hull into two pieces. The right edges belonging to the left subset and the left edges belonging to the right subset must be deleted.

The remaining part of the algorithm is a solution for the base case (i.e., the leaves of your recursion). The algorithm is illustrated in Figure 2. The lines in red indicate the upper and lower common tangent.

function FINDING THE UPPER COMMON TANGENT

Start with the rightmost point of the left hull and the leftmost point of the right hull

while the edge is not upper tangent to both left and right **do**
while the edge is not upper tangent to the left **do**
move counter clockwise to the next point on the left hull ▷ Hint: We want to move to the next point(s) on the left hull as long as the slope decreases
end while
while the edge is not upper tangent to the right **do**
move clockwise to the next point on the right hull
end while
end while
end function

Some Other Hints:

Maintain clockwise (or counter clockwise) ordering when merging (natural if start that way). Note below that from one point (e.g. left-most) to each other point, clockwise order will be by decreasing slopes.

Handle the base cases ($n < 4$) properly by getting started with appropriately ordered hulls.

Be careful with your hull data structure, as you might need to loop around when moving around the hull (clockwise or counterclockwise).

2.3 For Interest

More generally beyond two dimensions, the convex hull for a set of points Q in a real vector space V is the minimal convex set containing Q.

Algorithms for some other computational geometry problems start by computing a convex hull. Consider, for example, the two-dimensional farthest-pair problem: we are given a set of n points in the plane and wish to find the two points whose distance from each other is maximum. This pair is also referred to as the diameter of the set of points. You can prove that these two points must be vertices of the convex hull.

The problem of finding convex hulls also finds its practical applications in pattern recognition, image processing, statistics and GIS.

2.4 Requirements

1. Write the full, unambiguous pseudo-code for your divide-and-conquer algorithm for finding the convex hull of a set of points Q. Be sure to label the parts of your algorithm. Also, label each part with its worst-case time efficiency.

2. Analyze the whole algorithm for its worst-case time efficiency. State the Big-Theta asymptotic bound. Include the recurrence relation.

3. Implement your divide and conquer algorithm.

4. Conduct an empirical analysis of your algorithm by running several experiments as follows:

   (a) For each of $n \in \{10, 100, 1000, 10000, 100000, 500000, 1000000\}$

   (b) Generate a set of n (x, y) points in the plane. For each point set, find the convex hull and record the elapsed time.

   (c) For each size n, compute the mean time t required, and plot n vs t.

5. Find the relation of your plot to your theoretical analysis. In other words, if your theoretical analysis tells you that for a set of n points, complexity is $\Theta(g(n))$, identify the constant c. If it does not fit your data, what function does fit?

6. If the theoretical and empirical analyses differ, discuss the reasons for the difference.

You must submit a single PDF file as your answer to this question, which will be marked out of 40 marks. Marks breakdown is as follows:

1. Correct, concrete, readable pseudo code and the worst case analysis of each procedure/function, 8 marks.
2. Theoretical analysis for the entire algorithm including discussion of the recurrence relation, showing your work, 8 marks.
3. Your raw and mean experimental outcomes, 4 marks.
4. The plot, 2 marks.
5. Discussion of the pattern in your plot, including identification of order of growth and estimate of constant of proportionality (showing your calculations and assumptions), 6 marks.
6. Discussion of differences between theoretical and empirical analysis (if there are none, discuss why), 2 marks.
7. A screenshot of output of your program with 100 points is worth 2 marks.
8. A screenshot of your output for a program with 1000 points is worth 2 marks.
9. Your source code, correctly indented, commented etc, 6 marks.

2.5 Further information

See, Cormen, Leiserson, Rivest, & Stein. Introduction to Algorithms. Second Edition. MIT Press. 2001. pp. 939, 947-947, as well as the wikipedia article titled "Convex hull".

3 Acknowledgements

The ticketing problem is adapted from the Stanford nifty problem set.
The convex hull problem is adapted from
`http://axon.cs.byu.edu/~martinez/classes/312/Projects/Project2/project2.html`.