CSC 225 - SUMMER 2025
ALGORITHMS AND DATA STRUCTURES I
PROGRAMMING ASSIGNMENT 1
UNIVERSITY OF VICTORIA

Due: Sunday, June 15th, 2025 before 11:55pm. **Late assignments will not be accepted.**

1 Assignment Overview

For this programming assignment, you will implement efficient algorithms to maintain a convex hull for a point set as it changes. Your submission will take the form of a class `HullBuilder` with three methods: `addPoint`, `getHull` and `isInsideHull`. The algorithms for convex hulls and containment discussed in the lectures will be good starting points for your implementation, but due to the unique constraints of this problem, you will need to develop your own variations of the algorithms to receive full marks.

Your first priority in developing a solution is to create **working code**, implemented using algorithms whose correctness you can explain and justify, and tested rigorously (by you) to ensure correctness. 50% of the grade for this assignment will be based on correctness: if you submit a correct implementation, even if it is inefficient, you will receive at least 10/20. Note that 'correctness' is not just based on the ability of the code to pass automated tests, but also your ability to explain why your implementation would be correct on any input.

Once you have working code, your second priority is to implement an **asymptotically efficient algorithm**. For this assignment, efficiency will be judged entirely based on the worst-case running time of the algorithm (not the average-case performance or the actual time required to run the program). To receive marks for your algorithm's efficiency, you will need to explain and justify the worst case running time to the assignment evaluator (one of your instructors); you will not receive any marks for an efficient algorithm unless you can establish its efficiency. Details on the running time requirements can be found in Section 2.3.

Note that if your solution is not substantially correct, you will not receive any marks for efficiency (since the running time of broken code is irrelevant). Minor bugs (resulting in one or two test cases being incorrect) will result in a deduction of marks, but major correctness problems (including an inability to explain your algorithm's correctness) will result in no marks being awarded for efficiency.

The specification for this assignment can be found in Section 2. To help you test your code, several Java programs have been provided which use the `HullBuilder` class. These are documented in Section 2.2.

2 Specification

A Java source file `HullBuilder.java` has been posted, containing an empty class `HullBuilder` with three public methods. The methods all make use of a separate class `Point2d` which represents an (x, y) point in the plane. The source file `Point2d.java` has also been provided. Although you are welcome to use any code you want inside of `HullBuilder.java`, to ensure compatibility you may not change the function signature (names, argument types or return type) of any of the public methods, and you also may not change the public interface of the `Point2d` class.

The methods of `HullBuilder` are summarized below. Each instance of `HullBuilder` will internally maintain a set of points, which will be used to compute convex hulls and to test point containment. Notice that there is no facility to remove points from the set.

- `addPoint(P)` adds the provided point P to the point set.
- `getHull()` will return a list of points (all of which must be part of the point set) corresponding to the convex hull of the point set. Notice that the hull will change as points are added.
- `isInsideHull(P)` takes a point P (which may or may not be part of the point set) and returns `true` if P lies inside the convex hull of the point set and `false` otherwise. Note that points which lie on the boundary of the convex hull polygon (including vertices of the polygon) are considered to lie inside the hull.

The `getHull` method will return a list (of point objects) that comprises a convex hull polygon. The following definition of a convex polygon will be strictly enforced (and any sequence of points which does not meet the below criteria will not be considered a polygon).

- A polygon may not contain the same point twice (even if the duplicate point is both the beginning and end of the list).
- Every sequence of three consecutive points on the exterior of the polygon must make a right turn. This includes three-point sequences that wrap around from the end of the point list to the beginning. No straight-line or left-turn sequences are allowed.

Note that there is no requirement that the sequence start or end at a particular point (like the point with the minimum x-coordinate). See Section 2.4 for a concrete example.

2.1 Points and Right Turns

Each (x, y) point is represented by an instance of the provided `Point2d` class, which contains members `x` and `y`. You are permitted to modify the `Point2d` class if you want to, but please remember to submit your modified version with your code. You must ensure that the provided `PrintHull` and `TestContainment` programs continue to work without modification (we will be using these programs to test your code).

The lecture slides contain formulas to determine if a sequence of three points P, Q, and R forms a left turn, right turn or straight line. To save you some time, this test has been provided as the static method `Point2d.chirality`, which is also shown below.

```
/* chirality(P,Q,R)
    Given three points P, Q and R, determine whether the path
    PQR is a straight line, makes a left turn or makes a right turn.
    The return value will be:
        -1 if PQR makes a left turn
```

```
        0  if PQR is a straight line (that is, if P, Q and R are collinear)
        1  if PQR makes a right turn
    Although this method might be convenient, you are not required to use it.
*/
public static int chirality( Point2d P, Point2d Q, Point2d R ){
    double c = (Q.y-P.y)*(R.x-P.x) + (P.x-Q.x)*(R.y-P.y);
    if (c < 0)
        return -1;
    else if (c > 0)
        return 1;
    else
        return 0;
}
```

2.2 Tester Programs

Several testing programs have been provided to help you test your implementation. Note that, although some testing infrastructure is provided, as with all assignments in this course, **you are responsible for ensuring that your code is tested thoroughly**. You will likely find that the graphical interface provided by `HullViewer225` is the most convenient for debugging purposes. The other programs may be helpful if you want to run any automated testing (and will also be used by your instructors to assist in evaluating your assignment). The header comments in each program's source files contain basic instructions for running the program.

- `HullViewer225.java` provides a graphical interface for creating and viewing point sets and convex hulls. You can create points by clicking on the canvas and call the `getHull` method to compute a convex hull with a button in the interface. You can also test whether a particular point lies inside or outside the convex hull (as computed by your `isInsideHull` method) by right clicking on the canvas.
- `PrintHull.java` contains a program which reads a file of points and prints the convex hull to standard output. You can save point files from the `HullViewer225` interface for use with `PrintHull`. This program is a minimal test interface for `addPoint` and `getHull`.
- `TestContainment.java` contains a program which reads a file of points and tests if a particular point (x, y) (provided on the command line) lies inside or outside the convex hull of the points in the file. This program is a minimal test interface for `addPoint` and `isInsideHull`.

2.3 Running Time Requirements

Your mark will be determined based on both the **correctness** and **efficiency** of your implementation. The correctness of your code will be evaluated by a combination of automated testing and your ability to justify your code. The efficiency of your implementation will be evaluated entirely based on your ability to determine and justify the worst case running time of your implementation. See Section 3 for specific details of the evaluation process.

The table below summarizes the worst-case running time expectations for each part of your implementation. The evaluation for `addPoint` and `getHull` will be based on the running time for the slowest of those two functions (so, for example, having a $O(1)$ `addPoint` function will not be helpful

if `getHull` is $O(n^3)$). In the table below, the value n is used to refer to the current size of the whole point set (including points which are not vertices of the convex hull polygon) and the value h is used to refer to the current size of the convex hull.

Method	Bad	Good	Optimal
`Both addPoint and getHull`	$\Omega(n^2)$	$O(n)$	$O(h)$
`isInsideHull`	$\Omega(n^2)$	$O(h)$	$O(\log_2 h)$

To receive credit for having a particular running time, **both** of the following must apply.

- The implementation actually achieves the required running time.
- You are able to justify the running time.

Even if your code is optimal, you will only receive marks for the optimal running time if you can formally justify that running time. You will receive substantial partial marks if you can successfully justify a sub-optimal running time (even a bad running time). You will lose marks if you do not understand the running time of your code (for example, if you falsely claim that the code is optimal when it is not).

2.4 Example

Consider the following set S of 8 points in the plane.

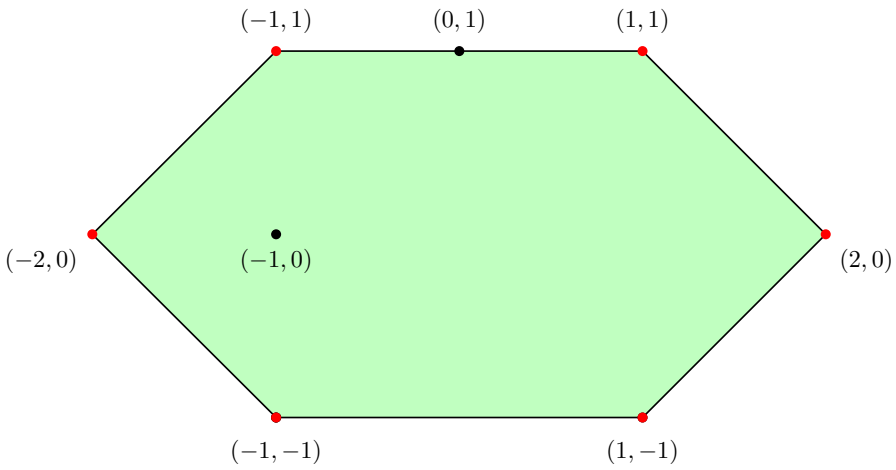
$$S = \{(-2,0),\ (2,0),\ (-1,1),\ (-1,-1),\ (1,1),\ (1,-1),\ (0,1),\ (-1,0)\}$$

To use this point set as the input to the `PrintHull` or `TestContainment` test programs (or to load it into `HullViewer225`), the point set would be represented by the following text file.

```
-2 0
2 0
-1 1
-1 -1
1 1
1 -1
0 1
-1 0
```

Note that the ordering of points in the input file is arbitrary (and your code should produce the same result regardless of the ordering of input points).

The diagram below shows the point set and the points of its convex hull (in red).



The convex hull polygon for the set S can be specified by the sequence

$$H_1 = (-2, 0),\ (-1, 1),\ (1, 1),\ (2, 0),\ (1, -1),\ (-1, -1)$$

or by the equivalent sequence

$$H_2 = (1, 1),\ (2, 0),\ (1, -1),\ (-1, -1),\ (-2, 0),\ (-1, 1)$$

or by any other ordering of the 6 points comprising the hull in which every three-point sequence makes a right turn.

Notice that the point $(0, 1)$ is not part of the convex hull polygon, even though it lies on the boundary of the polygon. This is because adding the point $(0, 1)$ to the polygon would produce the polygon sequence

$$(-2, 0),\ (-1, 1),\ (0, 1),\ (1, 1),\ (2, 0),\ (1, -1),\ (-1, -1)$$

which contains the subsequence $(-1, 1),\ (0, 1),\ (1, 1)$ that does not form a right turn (it is a straight line).

Suppose the point set S above is saved into a text file called `example.txt`. Running the `PrintHull` program on the set S above, using the command

```
java PrintHull example.txt
```

will the output below when your solution is correct (although, as mentioned above, other orderings of the polygon points would also be correct).

```
Reading points from example.txt
Read 8 points from example.txt
-2.00000 0.00000
-1.00000 1.00000
1.00000 1.00000
2.00000 0.00000
1.00000 -1.00000
-1.00000 -1.00000
```

The `TestContainment` program can be used to test the `isInsideHull` method. On the point set in `example.txt`, the following command will check whether the point $(0.5, 1)$ lies inside the convex hull.

```
java TestContainment example.txt 0.5 1.0
```

The output of the above command using a model solution is shown below.

```
Reading points from example.txt
Read 8 points from example.txt
Point (0.500000, 1.00000) is contained in the convex hull.
```

The command below will check the containment of the point $(1000, 1000)$, which is definitely not in the convex hull of the set S above.

```
java TestContainment example.txt 1000 1000
```

The output of the above command using a model solution is shown below.

```
Reading points from example.txt
Read 8 points from example.txt
Point (1000.00, 1000.00) is not contained in the convex hull.
```

You are encouraged to use the features of the Java Standard Library (including any of the data structures it provides) in your code. If you use a standard library data structure, make sure you are aware of the running times of the operations you use, since that information will be important for determining the running time of your program.

2.5 Implementation Advice

Although this assignment may seem daunting, most of the complexity lies in the specification (since this application is likely new to you) and in the finishing touches needed to make the algorithm asymptotically fast. The actual volume of code needed may be significantly less than assignments you have completed in the past. Your instructor's solution required fewer than 200 lines of Java code.

Array-Based Lists: You are welcome to use array based list types, like `ArrayList` and `Vector`. However, keep in mind that since these are backed by arrays, the running time of the `add` method can be $O(n)$ in the worst case (on a list containing n elements) if a resize operation is necessary. To avoid this having a negative affect on the running time of your code, you may want to pre-size the list (using the `ensureCapacity` method) to prevent any resize operations from being necessary.

Hash Tables: You should not use a hash table (or any hashing-based Java data structures, like `HashMap`) for your implementation if you want to achieve good worst-case performance, since a hash table with n elements has a $O(n)$ worst case running time for FIND and INSERT operations. Additionally, you may lose marks for your running time justification if you incorrectly characterize the running time of a hashing data structure. Although hash tables are very useful in most cases, they have a relatively poor worst-case running time (which we will discuss later in the course).

3 Evaluation

Submit all `.java` files needed to compile your assignment electronically via Brightspace. Your code must compile and run correctly in the lab environment (ECS 250). If your code does not compile

as submitted, you will receive a mark of zero.

This assignment is worth 10% of your final grade and will be marked out of 20 during an interactive demo with an instructor. You will be expected to explain the running time of your implementation to the evaluator, and may also be asked to explain or justify some aspects of your code. Demo times must be booked in advance (further information will be provided closer to the due date). If you do not schedule a demo time, or if you do not attend your scheduled demo, you will receive a mark of zero.

The marks are distributed among the components of the assignment as follows. See Section 2.3 for details on the running time evaluation.

Marks	Component
10	Your implementation functions correctly on a variety of test inputs and you are able to clearly describe and justify your algorithm to the evaluator.
7	Your `addPoint` and `getHull` implementations have efficient worst-case running times and you are able to justify the running times to the evaluator.
3	The `isInsideHull` implementation has an efficient worst-case running time and you are able to justify the running time to the evaluator.

If your code is not well organized, or if it is poorly documented, the evaluator may ask you explain any aspects that are unclear. If you are unable to do so, up to 4 marks may be deducted. To be clear, you are not required to have spotless, perfectly organized code, but you should be prepared to explain any hard-to-read parts of your code.

You are permitted to delete and resubmit your assignment as many times as you want before the due date, but no submissions or resubmissions will be accepted after the due date has passed. You will receive a mark of zero if you have not officially submitted your assignment (and received a confirmation email) before the due date.

Your primary submitted file must be based on the provided template `HullBuilder.java`. You may use additional files if needed by your solution. Ensure that each submitted file contains a comment with your name and student number.

Ensure that all code files needed to compile and run your code in ECS 250 are submitted. Only the files that you submit through Brightspace will be marked. The best way to make sure your submission is correct is to download it from Brightspace after submitting and test it. You are not permitted to revise your submission after the due date, and late submissions will not be accepted, so you should ensure that you have submitted the correct version of your code before the due date. After submitting your assignment, Brightspace will automatically send you a confirmation email. **If you do not receive such an email, you did not submit the assignment.** If you have problems with the submission process, contact the instructor **before** the due date.