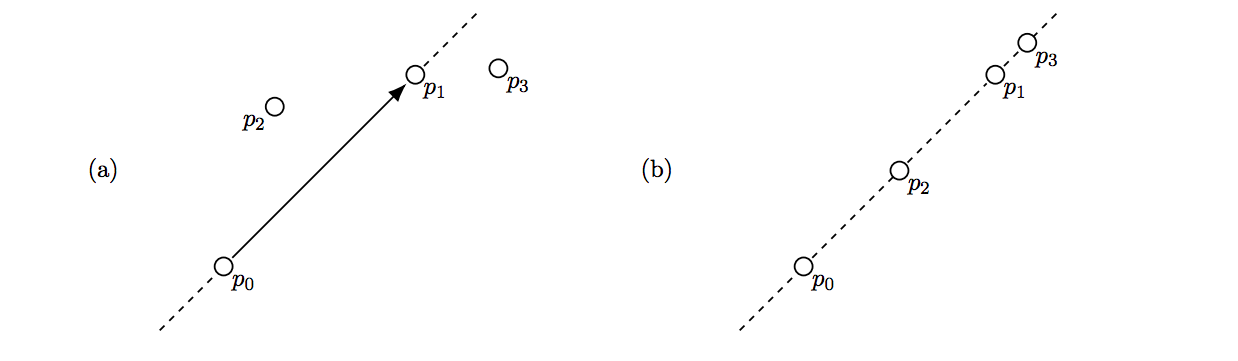
**Introduction**

A set S in the plane is called convex if it satisfies the following property: for every pair of points p, q ∈ S the line segment between them pq is also in the set S (i.e., pq ⊆ S). One problem that comes up time and time again in areas such as computational geometry is the problem of finding the smallest convex set S which contains all points in some other set X. This set is called the convex hull of X. When given some set of points we can describe the convex hull S by listing the points which make up the vertices in the boundary of the convex hull.

**Part A: Point Orientation**

Let p0, p1, p2 be three points in the plane. If p2 is left of the line segment p0p1, then we write Left(p0, p1, p2). If p2 is right of the line segment p0p1, then we write Right(p0, p1, p2). If three points are on the same straight line, then we say that (p0, p1, p2) are collinear.



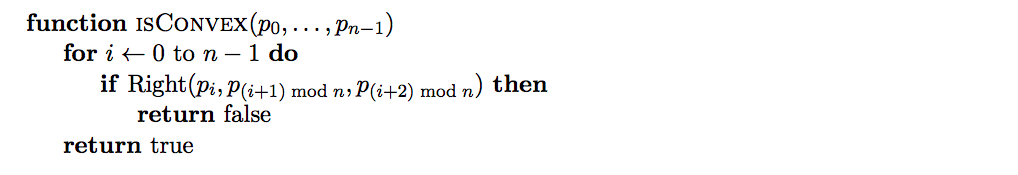
(a) In this example p2 is Left of p0p1, while p3 is Right of p0p1.

(b) Here p0, p1, p2 and p3 are all collinear points, so we say that each 3-subset of the four points are collinear e.g., p0, p2 and p3 are collinear.

Your task is to complete the implementation of orientation() in convex-hull.c which takes three points (p0, p1 and p2) and returns ‘l’ if p2 is Left of p0p1, ‘r’ if p2 is Right of p0p1 or ‘c’ if p0, p1 and p2 are Collinear. Your function must run in O(1) time.

**Part B: Checking for Convexity**

Let P = (p0, . . . , pn−1) be a sequence of n points in the Euclidean plane (R 2 ). Assume you have been given the following algorithm:



Does the algorithm above correctly determine if P is the boundary of a convex polygon in counter-clockwise order? Explain your answer.

**The Inside Hull Algorithm**

A polygon is called simple if it has no self intersections.

We describe a convex hull algorithm, called InsideHull and abbreviated with IH, that computes the convex hull of a simple polygon P = (p0, p1, . . . , pn−1), with points pi ∈ R 2 and its corresponding edges pipi+1. The points are ordered in counterclockwise order.

Although our definition of a simple polygon doesn’t exclude successive points being collinear, InsideHull assumes that this is not the case.

InsideHull uses a double ended queue abstract data type to represent the convex polygon it constructs. A double ended queue is abbreviated to deque and it has the following operations:

• push to the top of the deque

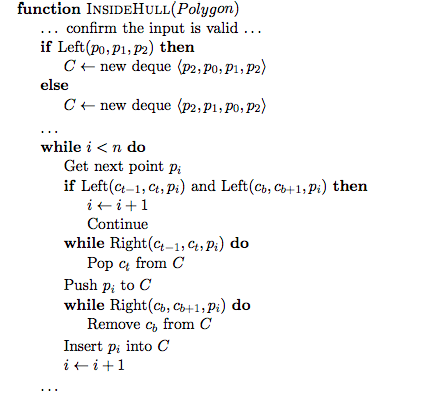
• pop from the top of the deque

• insert to the bottom of the deque

• remove from the bottom of the deque

The top and bottom of a deque c are indexed by ct and cb respectively. In the following algorithm the deque c gives rise to the convex hull C = hcb, cb+1, . . . , ct−1, cti.

The algorithm is as follows. Parts of the algorithm denoted by . . . are not complete and will require you to fill them in.



Note that during the algorithm, the deque C contains the points of the partial convex hull (which is completed by the end of the algorithm) in counter-clockwise order. At the beginning of each outer while loop cb and ct refer to the same point, and this point will always be the most recent point added to C.

**Part C: Deque Data Structure**

Implement a Deque abstract data structure and complete the definitions of the following functions in deque.c:

• new deque()

• free deque()

• deque push()

• deque insert()

• deque pop()

• deque remove()

• deque size()

It is up to you to decide on which underlying data structure will be used for your Deque. All of the operations described above must run in constant (O(1)) time .

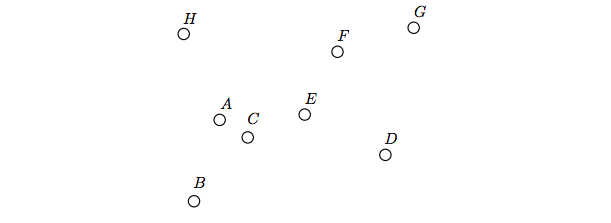
**Part D: Deque Analysis**

Explain your deque implementation and the decisions you made. Describe the time complexities of the four main operations (push, pop, insert, remove) and how these time complexities are achieved.

**Part E: Inside Hull Tracing**

Run the following example by hand and provide the points contained in the deque at each step of the algorithm. Be clear about which end of your deque is the top and which is the bottom.

The points are provided to InsideHull in alphabetic order, starting with point A.



**Part F: Inside Hull Implementation**

Implement the C function inside hull() that takes a polygon consisting of n points and computes the points of its convex hull.

The polygon will be provided as an array of n points named points, and the points will appear in counter-clockwise order.

Your function should store the points which make up the convex hull in the array hull, which will have enough memory for at least n points. These points should be stored in counter-clockwise order, and the first and last points should not be the same. This will mean the number of points in hull will be one fewer than the size of the deque C at the end of the algorithm.

The number of points stored in hull should be returned. If an error occurs return INSIDE HULL ERROR. If three consecutive points are collinear return COLLINEAR POINTS and don’t complete the algorithm (make sure you free any memory you have allocated before you return).

Note: You may assume that the input we will use to test your program will either have collinear points occurring consecutively or no collinear points at all. Note that the points across the array boundary (e.g., P olygon[n − 1], P olygon[0], and P olygon[1]) are considered consecutive.

**Part G: Inside Hull Analysis**

The best algorithms we have for computing convex hulls for a general set of n points are O(n log n), if they are not output sensitive2 .

Identify the basic operations in the InsideHull algorithm and use this to determine the time complexity of InsideHull, give your answer in Big-Oh notation. Explain how you arrived at this answer

Does this contradict the statement about runtimes above? Explain why it does or does not.