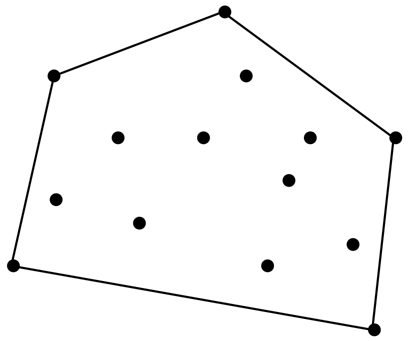
Convex Hull Algorithm

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points on a plane. Each point has an x and a y coordinate: $p_i = (x_i, y_i)$. The CONVEX HULL of a set of points is the smallest convex polygon that fits around the whole set. You can think of the convex hull as a rubber band that is stretched around the set of points, as shown below.



The following algorithm computes the convex hull of a set of points, P. The algorithm maintains a stack of points S. The stack allows the operations PUSH, POP, TOP, and NEXT-TO-TOP (where TOP return the top element of the stack and NEXT-TO-TOP returns the element just below the top of the stack; neither of these two functions pops the stack). When execution terminates, the stack S contains the points on the convex hull in clockwise order.

```
% Determine CH(P)

Let q0 be the point in P with minimum y coordinate.
Let < q1, q2, ..., q(n-1) > be the remaining points of P
  sorted by angle counterclockwise around q0.
Initialize the stack S to empty.

push( q0, S )
push( q1, S )
push( q2, S )

for i: 3 .. n-1
   while angle( next-to-top( S ), top( S ), qi ) < 0 do
      pop( S )
   end while
   push( qi, S )
end for
result S
```

Determine the asymptotic running time of this convex hull algorithm. Assume that the functions ANGLE, PUSH, POP, TOP, and NEXT-TO-TOP all take time $\mathcal{O}(1)$. Hint: You will have to use an amortized analysis to answer this question.

Convex Hull Analysis

- Finding q_0 takes $\mathcal{O}(n)$ time.
- Sorting $< q_1, q_2, \ldots, q_{n-1} >$ counterclockwise around q_0 takes $\mathcal{O}(n \log n)$ time.
- Initializing the stack takes $\mathcal{O}(1)$ time.

Let us amortize the cost of `pop` and `angle`: charge three credits for `push` (use one and store two), zero credits for `pop`, and zero credits for `angle` *if* it is immediately followed by a `pop`.

- Pushing q_0, q_1, and q_2 takes $\mathcal{O}(1)$ time.
- The `for` loop is executed $\mathcal{O}(n)$ times. In each iteration of the `for` loop, the `while` loop is executed k_i times.
  - The first $k_i - 1$ iterations will each result in a call to `angle` and to `pop`. These $2(k_i - 1)$ calls cost nothing, since they are paid for by the two credits stored on each item that was popped.
  - The k_i^{th} iteration costs $\mathcal{O}(1)$ to pay for the call to `angle`.
  - The final `push` operation is costs $\mathcal{O}(1)$.

  Thus each iteration of the `for` loop has constant (amortized) cost, and the `for` loop takes $\mathcal{O}(n)$ time total.

Thus the algorithm takes $\mathcal{O}(n \log n)$ time.