1. Suppose we want to maintain an array $X[1..n]$ of bits, which are all initially subject to the following operations.

   - LOOKUP(i): Given an index i, return $X[i]$.
   - BLACKEN(i): Given an index $i < n$, set $X[i] \leftarrow 1$.
   - NEXTWHITE(i): Given an index i, return the smallest index $j \geq i$ such that $X[j] = 0$. (Because we never change $X[n]$, such an index always exists.)

   If we use the array $X[1..n]$, it is trivial to implement LOOKUP and BLACKEN in $O(1)$ time and NEXTWHITE in $O(n)$ time. But you can do better! Describe data structures that support LOOKUP in $O(1)$ worst-case time and the other two operations in the following time bounds. (We want a different data structure for each set of time bounds, not one data structure that satisfies all bounds simultaneously!)

   (a) The worst-case time for both BLACKEN and NEXTWHITE is $O(\log n)$.

   (b) The amortized time for both BLACKEN and NEXTWHITE is $O(\log n)$. In addition, the *worst-case* time for BLACKEN is $O(1)$.

   (c) The amortized time for BLACKEN is $O(\log n)$, and the worst-case time for NEXTWHITE is $O(1)$.

   (d) The worst-case time for BLACKEN is $O(1)$, and the amortized time for NEXTWHITE is $O(\alpha(n))$. *[Hint: There is no* WHITEN*.]*

2. Recall that a standard (FIFO) queue maintains a sequence of items subject to the following operations:

   - PUSH(x): Add item x to the back of the queue (the end of the sequence).
   - PULL(): Remove and return the item at the front of the queue (the beginning of the sequence).

   It is easy to implement a queue using a doubly-linked list and a counter, using $O(n)$ space altogether, so that each PUSH or PULL requires $O(1)$ time.

   (a) Now suppose we want to support the following operation instead of PULL:

   - MULTIPULL(k): Remove the first k items from the front of the queue, and return the kth item removed.

   Suppose further that we implement MULTIPULL using the obvious algorithm:

```
MULTIPULL(k):
  for i ← 1 to k
      x ← PULL()
  return x
```

   Prove that in any intermixed sequence of PUSH and MULTIPULL operations, starting with an empty queue, the amortized cost of each operation is $O(1)$. You may assume that k is never larger than the number of items in the queue.

   (b) Now suppose we ***also*** want to support the following operation instead of PUSH:

   - MULTIPUSH(x, k): Insert k copies of x into the back of the queue.

   Suppose further that we implement MULTIPUSH using the obvious algorithm:

```
MultiPush($k, x$):
    for $i \leftarrow 1$ to $k$
        Push($x$)
```

Prove that for any integers ℓ and n, there is a sequence of ℓ MultiPush and MultiPull operations that require $\Omega(n\ell)$ time, where n is the maximum number of items in the queue at any time. Such a sequence implies that the amortized cost of each operation is $\Omega(n)$.

(c) Finally, describe a data structure that supports arbitrary intermixed sequences of MultiPush and MultiPull operations in $O(1)$ amortized cost per operation. Like a standard queue, your data structure must use only $O(1)$ space per item. *[Hint: **Don't** use the obvious algorithms!]*

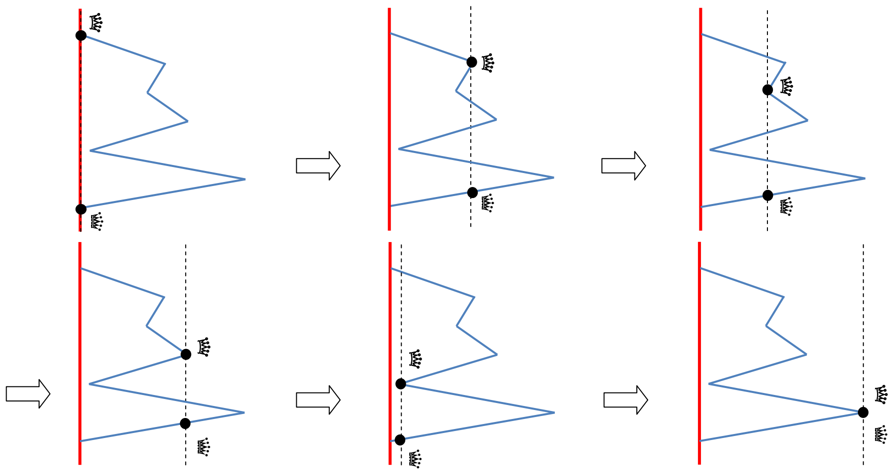
3. In every cheesy romance movie there's always that scene where the romantic couple, physically separated and looking for one another, suddenly matches eyes and then slowly approach one another with unwavering eye contact as the music rolls and in and the rain lifts and the sun shines through the clouds and kittens and puppies....

Suppose a romantic couple—in grand computer science tradition, named Alice and Bob—enters a park from the northwest and southwest corners of the park, locked in dramatic eye contact. However, they can't just walk to one another in a straight line, because the paths of the park zig-zag between the northwest and southwest entrances. Instead, Alice and Bob must traverse the zig-zagging path so that their eyes are always locked perfectly in vertical eye-contact; thus, their x-coordinates must always be identical.

We can describe the zigzag path as an array $P[0..n]$ of points, which are the corners of the path in order from the southwest endpoint to the northwest endpoint, satisfying the following conditions:

- $P[i].y > P[i-1].y$ for every index i. That is, the path always moves upward.
- $P[0].x = P[n].x = 0$, and $P[i].x > 0$ for every index $1 \le i \le n-1$. Thus, the ends of the path are further to the left then any other point on the path.

Prove that Alice and Bob can *always* meet.[1] *[Hint: Describe a graph that models all possible locations of the couple along the path. What are the vertices of this graph? What are the edges? What can we say about the degrees of the vertices?]*



[1]It follows that every cheesy romance movie (that reaches this scene) must have a happy, sappy ending.