Week 1 F

> [!WARNING] Direct quote from solution notes contain words of venting

```
// Initially it is thought to make least amount of swaps (cycle_length - 1)
// However, the problem becomes much clearer to instead
// constructively swap sort (identical to qsort without pivoting)
// so we have
/*
 *  ord <- true;
 *  while (ord)
 *      ord <- false
 *      for i : 1 -> n
 *          if (as[i] > as[i + 1])
 *              ord <- true
 *              swaps ++ <pos[i], pos[i + 1]>
 *              swap arr val and pos idx
 *
 */
```

Initially I did not take in the idea of an "inversion" went for the "shortest swapsort" by partitioning the sequences into mutex cycles. The inversion swapsort logic is inspired from quick sort, where if a number is bigger than the picked pivot, it will swap with the pivot, essentially forming an inversion (if the pivot is before the current pointer).

The brief logic is easy, for each element sweeped in every forward pass, we check if it is bigger than the next element (an inversion), we swap if it is. Such swaps are guaranteed to propagate the inversion in order until no inversions exists, at which point the swapsort is completed.

What's funny is that after completing the question, I had a read and realized this is just an instance of bubble sort.

I have a feeling that this algorithm could be further accelerated since we could heuristically guess where each element's "destination" would be, therefore perform lookahead inversion swaps. Such algorithm is bounded by the "lookahead", which its preprocessing would be a typical sort that takes O(n log n).

The total runtime of this algorithm is worst case $O(n^2)$.