Last Re-sort

source: XKCD, `https://xkcd.com/1185/`

In this exercise we implement two comparison based sorting algorithms: *selection sort* and *quicksort*.

Selection Sort Given a list of elements to sort, selection sort repeatedly selects a minimal element in the list, removes it from the list and adds it to a new list. Implement this algorithm in the function. Think carefully on how to handle duplicate elements. `selectionsort :: Ord a => [a] -> [a]`
Hint: have a look in `Data.List`.
Example

```
> selectionsort [2,3,10,5,-3,2]
[-3,2,2,3,5,10]
> selectionsort (reverse [1..10])
[1,2,3,4,5,6,7,8,9,10]
```

Quicksort The well-known quicksort algorithm works by selecting an arbitrary element from the input list, the *pivot*. The list is then partitioned into two halves. Elements that are less than or equal to the pivot go in the left half, the other elements in the right half. Both halves are then sorted in turn. The sorted halves are then concatenated to obtain the final, sorted, list.

The first thing to implement is the partitioning step: this is accomplished by a function `partition :: (a -> Bool) -> [a] -> ([a],[a])`, such that `partition p xs` returns a tuple `(ys,zs)` where the `ys` contains all the elements of `xs` for which `p` is true, and `zs` contains all the elements of `ys` for which `p` is false. For the purpose of the exercise, implement the partitioning step three times: once using a fold, once using `filter` and once using list comprehensions.
For example:

```
> partitionFold (< 0) [2,3,4,-1,6,-20,0]
([-1,-20],[2,3,4,6,0])
> partitionFilter odd [1,2,3,4,5,6]
([1,3,5],[2,4,6])
> partitionLC (not . even) [1,2,3,4,5,6]
([1,3,5],[2,4,6])
```

Implement the quicksort algorithm. In a non-empty list, choose the first element as the pivot. Do not forget the base case(s).
For example:

```
> quicksort [2,3,10,5,-3,2]
[-3,2,2,3,5,10,5]
> quicksort (reverse [1..10])
[1,2,3,4,5,6,7,8,9,10]
```