

## Last Re-sort

### INEFFECTIVE SORTS

```
DEFINE HALFHEARTEDMERGESORT(LIST):
  IF LENGTH(LIST) < 2:
    RETURN LIST
  PIVOT = INT(LENGTH(LIST) / 2)
  A = HALFHEARTEDMERGESORT(LIST[:PIVOT])
  B = HALFHEARTEDMERGESORT(LIST[PIVOT:])
  // UMMMMMM
  RETURN [A, B] // HERE. SORRY.
```

```
DEFINE FASTBOGOSORT(LIST):
  // AN OPTIMIZED BOGOSORT
  // RUNS IN O(N LOG N)
  FOR N FROM 1 TO LOG(LENGTH(LIST)):
    SHUFFLE(LIST):
    IF ISSORTED(LIST):
      RETURN LIST
  RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"
```

```
DEFINE JOBININTERVIEWQUICKSORT(LIST):
  OK SO YOU CHOOSE A PIVOT
  THEN DIVIDE THE LIST IN HALF
  FOR EACH HALF:
    CHECK TO SEE IF IT'S SORTED
    NO, WAIT, IT DOESN'T MATTER
    COMPARE EACH ELEMENT TO THE PIVOT
    THE BIGGER ONES GO IN A NEW LIST
    THE EQUAL ONES GO INTO, UH
    THE SECOND LIST FROM BEFORE
  HANG ON, LET ME NAME THE LISTS
  THIS IS LIST A
  THE NEW ONE IS LIST B
  PUT THE BIG ONES INTO LIST B
  NOW TAKE THE SECOND LIST
  CALL IT LIST, UH, A2
  WHICH ONE WAS THE PIVOT IN?
  SCRATCH ALL THAT
  IT JUST RECURSIVELY CALLS ITSELF
  UNTIL BOTH LISTS ARE EMPTY
  RIGHT?
  NOT EMPTY, BUT YOU KNOW WHAT I MEAN
  AM I ALLOWED TO USE THE STANDARD LIBRARIES?
```

```
DEFINE PANICSORT(LIST):
  IF ISSORTED(LIST):
    RETURN LIST
  FOR N FROM 1 TO 10000:
    PIVOT = RANDOM(0, LENGTH(LIST))
    LIST = LIST[PIVOT:] + LIST[:PIVOT]
  IF ISSORTED(LIST):
    RETURN LIST
  IF ISSORTED(LIST):
    RETURN LIST
  IF ISSORTED(LIST): // THIS CAN'T BE HAPPENING
    RETURN LIST
  IF ISSORTED(LIST): // COME ON COME ON
    RETURN LIST
  // OH JEEZ
  // I'M GONNA BE IN SO MUCH TROUBLE
  LIST = [ ]
  SYSTEM("SHUTDOWN -H +5")
  SYSTEM("RM -RF ./")
  SYSTEM("RM -RF ~/*")
  SYSTEM("RM -RF /")
  SYSTEM("RD /S /Q C:\*") // PORTABILITY
  RETURN [1, 2, 3, 4, 5]
```

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 `xs` 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]
```