

CS 350 2020-21 Homework 3

(Each question, whether a top-level question or a sub-question, carries 10 marks.)

1. Write a quicksort routine in Haskell which can sort any list of the following type.

`Ord a => [a] -> [a]`

2. Write a function

`uniq :: Eq a => [a] -> [a]`

which, given a list of elements, returns a list of unique elements.

3. Use higher-order functions to write a Haskell function for the following. Consider the following array.

$$\begin{pmatrix} a_{0,0} & \dots & a_{0,9} \\ & \ddots & \\ a_{9,0} & \dots & a_{9,9} \end{pmatrix}.$$

We represent the array using some suitable data structure (e.g. a list of lists). The concrete representation of the array is not important for this question.

A position in the array is represented by a pair (i, j) .

Write a Haskell function

```
neighbors :: (Ord a1, Ord a2, Num a1, Num a2) =>
           a1 -> a2 -> [(a1, a2)]
```

which, given i and j , computes a list of positions which are immediately adjacent to (i, j) . Note that there can be exactly 3, 5, or 8 neighbors depending on the position.

For example,

```
neighbors 0 0
```

should return $[(0, 1), (1, 0), (1, 1)]$. The output should be sorted ascending in lexicographic order of tuples. (You can use quicksort from Question 1 for sorting the output list.)

4. Using higher-order functions alone, write a Haskell function which can compute the number of words in a string. Assume that the string contains newline characters which designate the end of lines.

Hint: You may use the functions `lines` and `words` from the Haskell Prelude.

5. Write the function

```
compose_multiple :: [b -> b] -> b -> b
```

which takes a list of functions, and an initial value, and performs compositions of functions in the list on the second argument. For example,

```
compose_multiple [succ, (\x -> 2*x)] 3
```

should return 7.

6. Code the following in Haskell.

- (a) Define a data type to construct a binary tree. A binary tree is either empty, or has a root (containing a value) with two subtrees, both of which are binary trees.
- (b) For the tree defined above, define a `maptree f t` method of the type

```
maptree :: (a->b) -> BinaryTree a -> b
```

- (c) For the tree defined above, define

```
foldTree :: (a -> b -> b -> b) -> b -> BinaryTree a -> b
```

which takes a ternary function f , an identity value, a binary tree t , and returns the result of the folding operation of f on t . For example,

```
foldTree add3 0 (Node 3 Nil (Node 2 Nil Nil))
```

should evaluate to 5.