

Programming Languages, Quiz

(Haskell and Operational Semantics)

lastname, firstname = _____, _____ score = _____ / 90

1 Mergesort

Fill in the blanks.

mergesort _____ = _____

mergesort _____ = _____

mergesort xs = _____

where (left, right) = _____

mergesorted _____ = xs

mergesorted _____ = ys

mergesorted ((x:xs), (y:ys))

| x<=y = _____

| otherwise = _____

split [] = _____

split [x] = _____

split (x:y:xs) = _____

where (left, right) = _____

Questions:

1. What's the type of these three functions? I've answered the first for you as an example.

(a) :t mergesort

Answer: mergesort :: Ord a => [a] -> [a]

(b) :t mergesorted

Answer: mergesorted :: _____

(c) :t split

Answer: split :: _____

2. What's the result of `split [1..7]`?

Answer: _____.

3. What about `mergesorted (split [1..7])`?

Answer: _____.

4. Is this merge sort faster or slower than quicksort (no random choice of pivot) in the **worst case**?

5. Is this sort stable?

2 If ... Then ... Else

In last week's lecture we have seen the syntax and semantics of simple boolean expressions:

syntax	semantics
$t ::=$ true false if t then t else t	$\frac{}{\text{if true then } t_1 \text{ else } t_2 \rightarrow t_1}$ $\frac{}{\text{if false then } t_1 \text{ else } t_2 \rightarrow t_2}$ $\frac{t \rightarrow t'}{\text{if } t \text{ then } t_1 \text{ else } t_2 \rightarrow \text{if } t' \text{ then } t_1 \text{ else } t_2}$

Below we will use Haskell's recursive datastructure (recall `Ast` from HW2) to implement the boolean expression defined above, so that

```
IFTHENELSE FALSE TRUE (IFTHENELSE TRUE FALSE TRUE)
```

means "if false then true else (if true then false else true)". The `eval` function below implements the one-step evaluation relation " \rightarrow ", so that `eval t` returns `IFTHENELSE TRUE FALSE TRUE`. Fill in the blanks.

```
data Ast = TRUE
         | FALSE
         | IFTHENELSE _____
         deriving (Show)
```

```
eval (IFTHENELSE _____) = t1
```

```
eval (IFTHENELSE _____) = t2
```

```
eval (IFTHENELSE t t1 t2) = _____
```

The `evalstar` function implements the multi-step evaluation " \rightarrow^* " relation, which is the reflexive, transitive closure of one-step evaluation \rightarrow (i.e., keep evaluating until you can't evaluate any more). E.g., calling `evalstar t` for the `t` defined above returns `FALSE`. Fill in the blanks.

```
evalstar TRUE = _____
```

```
evalstar FALSE = _____
```

```
evalstar t = _____
```

Questions:

- `:t IFTHENELSE`

Answer: `IFTHENELSE :: _____`.

- let `t' = IFTHENELSE (IFTHENELSE TRUE FALSE TRUE) TRUE (IFTHENELSE FALSE FALSE TRUE)`

`eval t'`

Answer: _____.

`eval (eval t')`

Answer: _____.

`evalstar t'`

Answer: _____.