Assignment 2

Hi, it’s time for the only non-code evaluative of the course. This assignment focuses on the theory/concepts covered in lectures 1-10 and is to be submitted between 3 and 5 April. There’s no set way to answer these questions, and you can be abstract or informal with your answers. And we’re not going to put the answers online. You have to mail your answers to us (in a text file (and not doc/pdf)), and we’ll correct them and send them back in a couple of days. Refer to the class notes on the course website for help. Use the internet for clarifications, but please don’t directly Google up the problems.

E1. Quicksort is a recursive sorting algorithm, discovered by Tony Hoare in 1959. This is how it works on a list X:
Quicksort (X):

- Pick the first element of X and call it P.
- Define A = Quicksort (all elements in X smaller than P).
- Define B = Quicksort (all elements in X larger than P).
- Return a new list by joining A and B with P in the middle.

1. Intuitively, how can you prove the correctness of this algorithm? You can use an example.
2. Write a Haskell code snippet for `quicksort :: [a] -> [a]`

E2. Given an infinite supply of coins of denominations `D = [D1..Dn]`, give a Dynamic Programming algorithm (no actual code) to tell if we can make a total of T using them. For example, if `D = [2,4,7]`, then `T = 6` should return `True` and `T = 3` should return `False`.

E3. Why are Haskell programs type-safe (unlike Python)?

E4. What could be the reason of `Prelude> 2^63 :: Int` giving a negative result (unlike `2^63 :: Integer`)?

E5. ______ makes infinite lists practical in Haskell (unlike Python).

E6. Haskell doesn't allow tuples of arity 1. What could be the reason?

E7. Why does Haskell not allow infinite tuple arity (like infinite lists)?

E8. `'->'` is ______ associative while function application is ___ associative.

E9. Give one example each of polymorphic types and overloaded types in Haskell.

E10. Why wouldn't Haskell ever have the 'dangling else' problem?

E11. Print a list of the first 10 perfect squares in one line in Haskell.

E12. What section can be used to raise a number to the second power? Options: `(^2) or (2^)`. Write its lambda definition.

E13. What do you understand by the terms “pure”, “lazy”, and “referential transparency” when describing Haskell.

E14. I wish to write a 2D world in Haskell, where we have coordinates as Integers, and the four Directions (Up, Down, Left, Right). What datatypes should I define for this flatland? What will be the type signature of a function that takes a coordinate, and moves one unit in a given direction and returns the new coordinates?

E15. Now we'll play a small game that I like to call, *Have you met Ted?*
I will give you some functions in Haskell, but I'll only tell you their type signatures. You have to guess what they could possibly do. (something about their definition).

```
1. Ted1 :: a -> a
2. Ted2 :: a -> a -> a
3. Ted3 :: a -> b
4. Ted4 :: a -> b -> a
5. Ted5 :: (a -> a) -> a
```

For example, for the first one, you could say: Okay, I know that Haskell is parametric. I know that this function (Ted1) has to return a value of the same type as the input, and it has to do that for all types (polymorphism). Thus, the only possible function that could do that is the identity function itself. Hence, Ted1 must have the definition:

```
Ted1 :: a -> a
Ted1 x = x
```

Finish similar reasonings for Ted2 to Ted5. Be as specific as you can.

We'd love to hear about your inputs on this assignment. Include a note on what you learned from this assignment, and whether you'd like one more of this kind.

Instructors,
Programming: Python and Functional (PPAF),
BITS Pilani
Date Uploaded: 24 March
Due Date: 5 April