Introduction to Programming: Quiz 3

Date: November 18, 2026 **Time:** 1030 – 1130 hrs **Marks:** 40 **Weightage:** 10%

Name: _____ **Roll no.:** _____

Part A has six multiple-choice questions of 2 marks each. Circle (or tick) exactly one option.
Part B (12 marks) and Part C (16 marks) ask for short code fragments. Write only what is asked.
Write answers in the space provided. Use your own paper for rough work.
Use the lecture definitions unless stated otherwise. *Write your name and roll number on top.*

Part A: Multiple choice

(6 × 2 = 12 marks)

1. Let `xss` be a list of k lists, each of length m. How many steps are taken by `foldl (++) [] xss` and `foldr (++) [] xss` to compute the full result?

   (A) Both take $O(km)$ steps
   (B) `foldl`: $O(k^2m)$, `foldr`: $O(km)$
   (C) `foldl`: $O(km)$, `foldr`: $O(k^2m)$
   (D) Both take $O(k^2m)$ steps

2. What is the running time of `h` on a list of length n?

```
h :: [Int] → Int
h [] = 0
h xs = length xs + h (take (length xs `div` 2) xs)
```

   (A) $O(\log n)$ (B) $O(n)$ (C) $O(n \log n)$ (D) $O(n^2)$

3. The file `Stack.hs` contains the following.

```
module Stack(Stack, push, pop, empty, isEmpty) where
data Stack a = Stack [a] deriving Show
push x (Stack xs) = Stack (x:xs)
 ...
```

   Another file imports `Stack` and evaluates `show (push 2 (push 1 empty))`. What is the result?

   (A) `"Stack [2,1]"` (B) `"[2,1]"` (C) `"Stack [1,2]"`
   (D) Compile-time error: `show` cannot be used, since the data constructor `Stack` is not exported

4. Other files must be able to write the signature `s :: Stack Int`, but must *not* be able to use the data constructor `Stack` (to build or pattern-match stacks). Which module header achieves this?

   (A) `module Stack where`

   (B) `module Stack(push, pop, empty) where`

   (C) `module Stack(Stack(..), push, pop, empty) where`

   (D) `module Stack(Stack, push, pop, empty) where`

5. Recall insertion sort (`isort = foldr insert []`) and quicksort with the first element as pivot. On which input of length n do *both* take $O(n^2)$ steps?

   (A) `[1 .. n]` (B) `[n, n-1 .. 1]` (C) `replicate n 7`

   (D) None of these: on each of them, at least one of the two sorts is faster than $O(n^2)$

6. Recall the efficient two-list queue from the lectures, `data Queue a = Queue [a] [a]`. Recall also the function `enqueue x q` which adds `x` at the rear of `q`.

```
q0      = enqueue 3 (enqueue 2 (enqueue 1 empty))
(_, q1) = dequeue q0
q2      = enqueue 5 (enqueue 4 q1)
```

   What is the internal representation of `q2` (not what `show` prints)?

   (A) `Queue [2,3] [5,4]` (B) `Queue [2,3,4,5] []`

   (C) `Queue [2,3] [4,5]` (D) `Queue [] [5,4,3,2]`

Extra space for spillover answers. Clearly mark the question number. Not for rough work!

Part B: A double-ended queue (12 marks)

7. A *double-ended queue* (deque) allows adding and removing elements at *both* ends. We implement it in `Deque.hs`, using two lists as for the efficient queue.

```
data Deque a = Deque [a] [a]
empty     :: Deque a                  empty              = Deque [] []
isEmpty   :: Deque a → Bool           isEmpty q          = ...
toList    :: Deque a → [a]            toList (Deque f b) = f ++ reverse b
pushFront :: a → Deque a → Deque a    pushFront x q      = ...
pushBack  :: a → Deque a → Deque a    pushBack x q       = ...
```

[3] (a) Define the three undefined functions above. Each should take one line.

[4] (b) Define `popFront :: Deque a → (a, Deque a)`, which removes and returns the first element. If the deque is empty, call `error`.
Hint: when `f` is empty but `b` is not, do not move all of `b` to the front. Instead, move some elements of `b` into `f` and retain the rest, so that both lists are non-empty, if possible. What is the simplest strategy for this, which also ensures that the new deque still represents the same sequence?

[2] (c) Modify the above code to symmetrically implement `popBack :: Deque a → (a, Deque a)`.

[3] (d) (i) Write the internal representation of `d'`, where

```
(_, d') = popFront (foldl (\d x → pushBack x d) empty [1..6])
```

Part C: Binary search trees with sizes (16 marks)

8. We augment binary search trees so that every node stores the number of nodes in its subtree.

```
data STree a = Leaf | Node Int (STree a) a (STree a)
```

In `t = Node n tl x tr`, `n` is the number of nodes in `t`, every value in `tl` is less than `x`, and every value in `tr` is greater than `x`.

[3] (a) Define `size :: STree a → Int`, which must take $O(1)$ steps. Also define a "smart constructor" `node :: STree a → a → STree a → STree a` that builds a node with the correct size field, assuming its subtrees have correct sizes.

[3] (b) Define `insert :: Ord a ⇒ a → STree a → STree a`, which adds a value to the tree (leaving it unchanged if the value is already present), keeping all size fields correct.

[5] (c) Define `kth :: Int → STree a → a` that returns the element of rank `k`. This function should satisfy the equation `kth k t = inorder t !! k`, so `kth 0 t` is the minimum. Assume `0 ⩽ k < size t`. *Hint:* compare `k` with the size of the left subtree. What is the running time of your `kth`, and why is it better than `inorder t !! k`?

[5] (d) Define `rank :: Ord a ⇒ a → STree a → Int` that returns the number of elements of the tree that are strictly smaller than `v`. The value `v` need not be in the tree. Your function should take time proportional to the height of the tree.