COS 326 Functional Programming Princeton University Final Exam, January 18, 2016

Name (print clearly):

Login:

You are allowed to bring a hand-written sheet (8.5x11in page, one side) into the exam containing any information you want. Other than that, the work on this exam must be your own, closed-book.

Honor Pledge and signature:

General Instructions:

- **Write your name and login on this page. Write and sign the Honor Code pledge** on this page. Write your name and login on an exam booklet (no need to write the pledge again).
- You have **3 hours** to complete this exam.
- There are **6 parts** to the exam (each part may have several questions). Write answers in the spaces provided. We will give partial credit on questions. So part of a solution, if it is correct, or at least sensible, is better than no solution at all.
- Minor errors in OCaml syntax will not be penalized significantly, if at all. However, we have no choice but to penalize errors that render a solution incomprehensible.
- If you cannot complete the details of a proof but can show that you know how to structure the proof, you will receive some credit. Show that you know how to break down a proof into appropriate cases. Show that you know what the induction hypothesis is by writing it down clearly. Show that you know what must be proven. Do not give two answers to a question hoping that one of them is correct. If you give two different answers to a question, you will receive no credit if one of the answers is correct and one is incorrect. If ambiguous, circle the answer you intend.

Grades on the 6 parts:

1 _____ / 2	4 _____ / 17	**Total _____ / 71**
2_____ / 4	5 _____ / 20	
3_____ / 8	6 _____ / 20	

There are 7 pages in all

PART 1. This paragraph is copied verbatim from Assignment 7:

> A naive implementation of the mapreduce function would first call map and then call reduce. However, such an implementation would create an unnecessary intermediate sequence from the map that would subsequently be used solely for the reduce. This leads to extra copying that can hinder performance. A better approach would be to combine the map and reduce functions in a single pass over the data. Be wary of creating intermediate data structures in your implementation.

There is a single-word technical term that describes this concept;

write it here: _____

PART 2. As you learned in your Algorithms class (such as COS 226), quicksort takes O(*N*log*N)* time and quick-select takes O(*N*) average-case time. Quick-select finds the median element of a sequence, or more generally, the *k*th-largest element.

Based on the parallel quicksort algorithm presented in lecture, fill in the blanks, then explain your answers in 50 words or less. You don't have to present the full algorithm, you assume we are familiar with the algorithm for quicksort presented in lecture.

In parallel, average-case,

Quicksort can sort a sequence in O(_____) work and O(_____) span.

Quick-select can find the median in O(_____) work and O(_____) span.

PART 3. Match each of these computer scientists with their award citation:

___ Edsger Dijkstra

___ Tony Hoare

___ Xavier Leroy

___ Barbara Liskov

___ John McCarthy

___ David MacQueen

___ Robin Milner

___ Simon Peyton Jones

A. Turing Award "For three distinct and complete achievements: **LCF, the first theoretically based yet practical tool for machine assisted proof construction**; **ML, the first language to include polymorphic type inference** together with a type-safe exception-handling mechanism; CCS, a general theory of concurrency."

B. Turing Award "For contributions to practical and theoretical foundations of programming language and system design, especially related to **data abstraction**, fault tolerance, and distributed computing." That is, **modules with representation hiding.**

C. Turing Award "For fundamental contributions to the definition and design of programming languages" including "***Communicating Sequential Processes."***

D. Turing Award for many contributions to Artificial Intelligence including the **LISP programming language.**

E. Turing Award "For fundamental contributions to programming as a high, intellectual challenge; for eloquent insistence and practical demonstration that programs should be composed correctly, not just debugged into correctness;" including the use of a stack for recursive functions, **graph algorithms such as shortest-path**, and the invention of **"semaphores" for synchronizing shared-memory concurrent programs.**

F. ACM Fellow "For significant research contributions in type theory and programming language design, especially for work on the design and implementation of Standard ML" and **the ML module system.**

G. ACM Fellow "For contributions to safe, high-performance functional programming languages and compilers, and to compiler verification" such as the **OCaml compiler** and the **CompCert verified C compiler**.

H. ACM Fellow "For contributions to functional programming languages" including the **Haskell programming language.**

PART 4. Consider these four functions:

```
let a (f: int->int->int) (z: int) (n: int) : int =
  let rec aux i =
    if i<=0 then z else f i (aux (i-1))
  in aux n

let b (f: int->int->int) (z: int) (n: int) : int =
  let rec aux i x =
    if i<=0 then x else aux (i-1) (f i x)
  in aux n z

let c (f: int->int->int) (z: int) (n: int) : int =
  let rec aux i x =
    if i>n then x else aux (i+1) (f i x)
  in aux 1 z

let rec d (f: int->int->int) (z: int) (n: int) : int =
 let rec aux i j =
  if i>j then z
  else if i=j then i
  else let k = (i+j)/2
      in f (aux (k+1) j) (aux i k)
  in aux 1 n
```

Q1. Assuming that f is not recursive (and does not call any recursive functions),
assuming that the ML compiler handles tail calls efficiently, what is the asymptotic space usage of each of these functions, as a function of n?

a _____ b _____ c _____ d _____

Q2. What properties must f and z have such that for all n, a f z n = b f z n ?
(Don't prove it. **Using a page in the exam booklet to work this out**, set n=4, simplify as much as possible, then eyeball it. Use standard mathematical terminology to describe the properties.)

Q3. What properties must f and z have such that for all n, a f z n = c f z n ?

Q4. What properties must f and z have such that for all n, a f z n = d f z n ?

PART 5. The following program implements "recent lists." Recent lists are like lists, but you can only access the front 4 elements.

```
module type RECENT =
 sig
  type 'a t
  val create: 'a -> 'a -> 'a -> 'a -> 'a t
  val cons: 'a -> 'a t -> 'a t
  val nth: 'a t -> int -> 'a option
 end

module R : RECENT =
 struct
  type 'a t = 'a list
  let create a b c d = [a;b;c;d]
  let cons a r = a::r
  let nth r i =    if 0 <= i && i < 4     then Some (List.nth r i)     else None
 end

let force (x: 'a option) : 'a =    match x with Some i -> i

let rec step (r: int R.t) =   Recent.cons (force (R.nth r 0) + force (R.nth r 1)) r

let rec repeat n g r =   if n<=0 then r else repeat (n-1) g (g r)

let f i = force (R.nth (repeat i step (R.create 1 1 1 1)) 2)

let mylist = List.map f [1;2;3;4;5;6]
```

ANSWER THESE QUESTIONS IN AN EXAM BOOKLET. PUT YOUR NAME ON THE BOOKLET **NOW!**

Q1. Show the value of mylist.

Q2. The implementation of R.nth calls upon List.nth. But List.nth raises an exception if the index is out of range. Prove that R.nth will never raise an exception (no matter what client uses it, not just the example client shown here). Prove using the methods and terminology taught in this course. *State the most important 1- or 2-word technical term relevant to how you proved it, and underline it.*

Q3. This implementation of Recent uses a data structure that can grow arbitrarily large. Implement a module R2 : RECENT that "works the same" but uses a bounded-size data structure.

Q4. Prove that R2 "works the same" as R, whatever that means. Part of the answer is to define formally what that means (in this case). *State the most important 1- or 2-word technical term relevant to how you proved it, and underline it.*

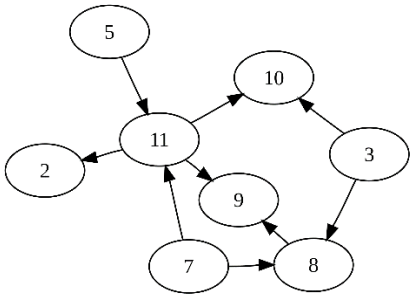
Q5. Assume that no operation of R raises an exception; but the client's "force" function can raise an exception. Prove that the "step" function and the "f" function never raise an exception.

Operators for map-reduce programming

These operations are from Assignment 7, plus **sort** and **filter**. For this exam, you can assume each one of these runs in "Parallel," i.e., polylog span (assuming the function *f* runs reasonably fast).

tabulate (f: int→'a) (n: int) : 'a seq	Create seq of length n, element i holds f(i)
seq_of_array: 'a array → 'a seq	Create a sequence from an array
array_of_seq: 'a seq → 'a array	Create an array from a sequence
length: 'a seq → int	Return the length of the sequence
empty: unit → 'a seq	Return the empty sequence
cons: 'a → 'a seq → 'a seq	(nondestructively) cons a new element on the beginning
singleton: 'a → 'a seq	Return the sequence with a single element
append: 'a seq → 'a seq → 'a seq	(nondestructively) concatenate two sequences
nth: 'a seq → int → 'a	Get the nth value in the sequence. Indexing is zero-based.
map (f: 'a → 'b) → 'a seq → 'a seq	Map the function f over a sequence
reduce (f: 'a → 'a → 'a) (base: 'a): 'a seq → 'a	Fold a function f over the sequence. f must be associative, and base must be the unit for f.
mapreduce: ('a→'b)→('b→'b→'b)→ 'b → 'a seq → 'b	Combine the map and reduce functions.
flatten: 'a seq seq → 'a seq	flatten [[a0;a1]; [a2;a3]] = [a0;a1;a2;a3]
repeat (x: 'a) (n: int) : 'a seq	repeat x 4 = [x;x;x;x]
zip: ('a seq * 'b seq) → ('a * 'b) seq	zip [a0;a1] [b0;b1;b2] = [(a0,b0);(a1,b1)]
split: 'a seq → int → 'a seq * 'a seq	split [a0;a1;a2;a3] 1= ([a0],[a1;a2;a3])
scan: ('a→'a→'a) → 'a → 'a seq → 'a seq	scan f b [a0;a1;a2;...] = [f b a0; f (f b a0) a1; f (f (f b a0) a1) a2; ...]
sort: ('a→'a→int) → 'a seq → 'a seq	sort compare [3;1;4;1;5] = [1;1;3;4;5]
filter: ('a→bool) → 'a seq → 'a seq	filter even [3;1;4;1;5;9;2;6] = [4;2;5]

PART 6. Write a parallel map-reduce algorithm for graph reachability.



A directed graph is represented by ordered pairs:

[(3,8); (5,11); (7,11); (7,8); (8,9); (3,10); (11,2);(11,9); (11,10)]

Given: list of ordered pairs *G*; root node *r*; a distance *n*;
Produce: a list of of pairs (*i,d)* where *i* is a node reachable in $d \le n$ hops from *r*, or (i,∞) if *i* is not reachable within *n* hops.

Write an algorithm using the map-reduce operators shown in the table. You can assume that *n* is small, and you are permitted to use *span* proportional to *n*. However, the out-degree of any node may be large! And the node-numbers may be *sparse* (in the graph shown, nodes 1,4,6 don't exist).

Accomplish this using a packed-adjacency-list representation, illustrated below. In the triple (*i,k,m*), *i* is a node-number, *k* is the index in the "arrowheads" list of the first out-edge from *i,* and *m* is the number of out-edges from *i* (which will appear consecutively in the arrowheads list).

Q1. Show a map-reduce program that will produce "arrowheads" and "adjacency lists" from "sorted graph". Illustrate the sequence-values produced as intermediate results, on the example input shown; label these illustrations using variable names from your program.

Q2. Show a map-reduce program that will produce "answer for *n*+1" from "arrowheads", "adjacency lists", and "answer for *n*." Illustrate the sequence-values produced as intermediate results, on the example input shown; label these illustrations using variable names from your program.

[(3,8); (5,11); (7,11); (7,8); (8,9); (3,10); (11,2);(11,9); (11,10)] (input graph)

0 1 2 3 4 5 6 7 8 (slot numbers)
[(3,8) ; (3,10); (5,11) ; (7,8) ; (7,11) ; (8,9) ; (11,2) ; (11,9) ; (11,10)] (sorted graph)

[8 ; 10 ; 11 ; 8 ; 11 ; 9 ; 2 ; 9 ; 10] (arrowheads)
0 2 1 2 1 0 0 3

[(2,0,0); (3,0,2); (5,2,1); (7,3,2); (8,5,1); (9,6,0); (10,6,0); (11,6,3)] (adjacency lists)

[(2,∞); (3,∞); (5,0); (7,∞); (8,∞); (9,∞); (10,∞); (11,∞)] (answer for *r*=5, *n*=0)
[(2,∞); (3,∞); (5,0); (7,∞); (8,∞); (9,∞); (10,∞); (11,1)] (answer for *r*=5, *n*=1)
[(2,2); (3,∞); (5,0); (7,∞); (8,∞); (9,2); (10,2); (11,1)] (answer for *r*=5, *n*=2)

ANSWER Q1 AND Q2 IN THE EXAM BOOKLET.

What to do if you have extra time when you've finished everything: Space out for 5 minutes, then copy over your answer to Part 6 or Part 5 so that it's as clear and as readable as can be.