Full Name: ______________________________

Andrew ID: ______________ **Section:** ______________

15–210: Parallel and Sequential Data Structures and Algorithms

Practice Final

May 2014

- There are 14 pages in this examination, comprising 7 questions worth a total of 110 points. The last few pages are an appendix with costs of sequence, set and table operations.
- You have 80 minutes to complete this examination.
- Please answer all questions in the space provided with the question. Clearly indicate your answers.
- You may refer to your one double-sided $8\frac{1}{2} \times 11$in sheet of paper with notes, but to no other person or source, during the examination.
- Your answers for this exam must be written in blue or black ink.

Circle the section YOU ATTEND

Sections		
A	9:30am - 10:20am	Naman
B	10:30am - 11:20am	Sam
C	12:30pm - 1:20pm	Isaac
D	12:30pm - 1:20pm	Nikki
E	1:30pm - 2:20pm	Esther and Ronald
F	1:30pm - 2:20pm	Ivan
G	3:30pm - 4:20pm	Will and Ian

Full Name: ____________________ **Andrew ID:** ____________________

Question	Points	Score
Binary Answers	20	
Costs	12	
Short Answers	18	
Slightly Longer Answers	20	
Longest Contiguous Increasing Subsequence	16	
Median ADT	12	
Geometric Coverage	12	
Total:	110	

Question 1: Binary Answers (20 points)

Clearly mark **T** or **F** to the left of each question.

(a) (2 points) The expressions (`Seq.reduce f I A`) and (`Seq.iter f I A`) always return the same result as long as `f` is commutative.

(b) (2 points) The expressions (`Seq.reduce f I A`) and (`Seq.reduce f I (Seq.reverse A)`) always return the same result if `f` is associative and commutative.

(c) (2 points) Any parallel algorithm for a problem is always faster than a sequential algorithm for the same problem.

(d) (2 points) Solving recurrences with induction can be used to show both upper and lower bounds?

(e) (2 points) Let p be an odd prime. In open address hashing with a table of size p and given a hash function $h(k)$, quadratic probing uses $h(k, i) = (h(k) + i^2) \mod p$ as the ith probe position for key k. If there is an empty spot in the table quadratic hashing will always find it.

(f) (2 points) `scan f b L` and `reduce f b L` always have the same asymptotic cost.

(g) (2 points) If a randomized algorithm has expected $O(n)$ work, then there exists some constant c such that the work performed is guaranteed to be at most cn.

(h) (2 points) The height of any binary search tree (BST) is $O(\log n)$.

(i) (2 points) Dijkstra's algorithm always terminates even if the input graph contains negative edge weights.

(j) (2 points) A $\Theta(n^2)$-work algorithm always takes longer to run than a $\Theta(n \log n)$-work algorithm.

Question 2: Costs (12 points)

(a) (6 points) Give tight assymptotic bounds (Θ) for the following recurrence using the tree method. Show your work.

$$W(n) = 2W(n/2) + n\log n$$

(b) (6 points) Check the appropriate column for each row in the following table:

	root dominated	leaf dominated	balanced
$W(n) = 2W(n/2) + n^{1.5}$			
$W(n) = \sqrt{n}W(\sqrt{n}) + \sqrt{n}$			
$W(n) = 8W(n/2) + n^2$			

Question 3: Short Answers (18 points)

Answer each of the following questions in the spaces provided.

(a) (3 points) What simple formula defines the parallelism of an algorithm (in terms of work and span)?

(b) (3 points) Name two algorithms we covered in this course that use the greedy method.

(c) (3 points) Given a sequence of key-value pairs A, what does the following code do?

```
Table.map length (Table.collect A)
```

(d) (3 points) What is the cut property of graphs that enables MST algorithms such as Kruskal's, Prim's and Borůvka's to work correctly?

(e) (3 points) What asymptotically efficient parallel algorithm/technique can one use to count the number of trees in a forest (tree and forest have their graph-theoretical meaning)? *(Hint: the ancient saying of "can't see forest from the trees" may or may not be of help.)* Give the work and span for your proposed algorithm.

(f) (3 points) What are the two ordering invariants of a Treap? (Describe them briefly.)

Question 4: Slightly Longer Answers (20 points)

(a) (6 points) Certain locations on a straight pathway recently built for robotics research have to be covered with a special surface, so CMU hires a contractor who can build arbitrary length segments to cover these locations (a location is covered if there is a segment covering it). The segment between a and b (inclusive) costs $(b-a)^2 + k$, where k is a non-negative constant. Let $k \geq 0$ and $X = \langle x_1, \ldots, x_n \rangle$, $x_i \in \mathbb{R}_+$, be a sequence of locations that have to be covered. Give an $O(n^2)$-work dynamic programming solution to find the cheapest cost of covering these points (all given locations must be covered). Be sure to state the subproblems and give a recurrence, including the base case(s).

(b) (7 points) Consider the following variant of the optimal binary search tree (OBST) algorithm given in class:

```
function OBST(A) = let
  function OBST'(S, d) =
    if |S| = 0 then 0
    else min_{i ∈ ⟨1...|S|⟩} (OBST'(S_{1,i-1}, d + 1) + d · p(S_i) + OBST'(S_{i+1,|S|}, d + 1))
in
  OBST'(A, 1)
end
```

Recall that $S_{i,j}$ is the subsequence $\langle S_i, S_{i+1}, \ldots, S_j \rangle$ of S. For $|A| = n$, place an asymptotic upper bound on the number of distinct arguments *OBST′* will have (a tighter bound will get more credit).

(c) (7 points) Given n line segments in 2 dimensions, the 3-intersection problem is to determine if any three of them intersect at the same point. Explain how to do this in $O(n^2)$ work and $O(\log n)$ span. You can assume the lines are given with integer endpoints (i.e. you can do exact arithmetic and not worry about roundoff errors).

Question 5: Longest Contiguous Increasing Subsequence (16 points)

Given a sequence of numbers, the *longest contiguous increasing subsequence* problem is to find the largest number of contiguous increases in a sequence of numbers. For example,

```
LCIS(<7, 2, 3, 4, 1, 8>)
```

will return 2 since there are 2 increases in a row in the contiguous subsequence `<2, 3, 4>`. Note that this is different from the longest increasing subsequence problem discussed in recitation.

(a) (4 points) The LCIS problem can be solved in linear work by strengthening the problem (inductive hypothesis) and solved using divide and conquer by splitting the sequence in half and solving each half. Describe what values you would return from the recursive calls to efficiently construct the solution.

(b) (4 points) Fill in the following SML code for your recursive divide-and-conquer algorithm:

```
fun LCIS (S : int seq) : int =
let
  fun LCIS'(S : int seq) : _____________________________________ =
    case (showt S) of
        EMPTY =>  ________________________

      | ELT(x) =>  _______________________

      | NODE(L,R) =>   (* fill in below *)









  in (* fill in below *)


  end
```

(c) (4 points) Assuming a tree-based implementation of sequences in which `showt`, and `nth` take $O(\log n)$ work, write recurrences for the work and span of `LCIS'` and state the solutions of the recurrences.

(d) (4 points) The problem can also be solved with a scan. Here is the code.

```
datatype Dir = UP of int | MIX of int

fun LCIS(S : int seq) =
let
   fun up i = if (nth S (i+1)) > (nth S i) then UP(1) else MIX(0)
   val Sup = tabulate up ((length S)-1)
   val (R,MIX(v)) = scan binop (MIX(0)) Sup
   val R' = map (fn MIX(x) => x) R
in
   Int.max(v, reduce Int.max 0 R')
end
```

Fill in the following code for `binop`.

```
fun binop(_ , MIX b) =          _______________________

   | binop(MIX a , UP b) =      _______________________

   | binop(UP a , UP b) =       _______________________
```

Question 6: Median ADT (12 points)

The *median* of a set C, denoted by `median`(C), is the value of the $\lceil n/2 \rceil$-th smallest element (counting from 1). For example,

$\texttt{median}(\{1, 3, 5, 7\}) = 3$
$\texttt{median}(\{4, 2, 9\}) = 4$

In this problem, you will implement an abstract data type `medianT` that maintains a collection of integers (possibly with duplicates) and supports the following operations:

`insert`(C, v)	:	`medianT` $\times$ `int` $\rightarrow$ `medianT`	add the integer v to C.
`median`(C)	:	`medianT` $\rightarrow$ `int`	return the median value of C.
`fromSeq`(S)	:	`int Seq.seq` $\rightarrow$ `medianT`	create a `medianT` from S.

Throughout this problem, let n denote the size of the collection at the time, i.e., $n = |C|$.

(a) (5 points) Describe how you would implement the `medianT` ADT using (balanced) binary search trees so that `insert` and `median` take $O(\log n)$ work and span.

(b) (7 points) Using some other data structure, describe how to improve the work to $O(\log n)$, $O(1)$ and $O(|S|)$ for the three operations respectively. The `fromSeq S` function needs to run in $O(\log^2 |S|)$ expected span and the work can be expected case. *(Hint: think about maintaining the median, the elements less than the median, and the elements greater than the median separately.)*

Question 7: Geometric Coverage (12 points)

For points $p_1, p_2 \in \mathbb{R}^2$, we say that $p_1 = (x_1, y_1)$ *covers* $p_2 = (x_2, y_2)$ if $x_1 \geq x_2$ and $y_1 \geq y_2$. Given a set $S \subseteq \mathbb{R}^2$, the *geometric cover number* of a point $q \in \mathbb{R}^2$ is the number of points in S that q covers. Notice that by definition, every point covers itself, so its cover number must be at least 1.

In this problem, we'll compute the geometric cover number for every point in a given sequence. More precisely:

> **Input:** a sequence $S = \langle s_1, \ldots, s_n \rangle$, where each $s_i \in \mathbb{R}^2$ is a 2-d point.
>
> **Output:** a sequence of pairs each consististing of a point and its cover number. Each point must appear exactly once, but the points can be in any order.

Assume that we use the `ArraySequence` implementation for sequences.

(a) (4 points) Develop a brute-force solution `gcnBasic` (in pseudocode or Standard ML). Despite being a brute-force solution, your solution should not do more work than $O(n^2)$.

(b) (4 points) In words, outline an algorithm `gcnImproved` that has $O(n \log n)$ work. You may assume an implementation of `OrderedTable` in which `split`, `join`, and `insert` have $O(\log n)$ cost (i.e., work and span), and `size` and `empty` have $O(1)$ cost.

(c) (4 points) Show that the work bound cannot be further improved by giving a lower bound for the problem.

Appendix: Library Functions

```
signature SEQUENCE =
sig
  type 'a seq
  type 'a ord = 'a * 'a -> order
  datatype 'a listview = NIL | CONS of 'a * 'a seq
  datatype 'a treeview = EMPTY | ELT of 'a | NODE of 'a seq * 'a seq

  exception Range
  exception Size

  val nth : 'a seq -> int -> 'a
  val length : 'a seq -> int
  val toList : 'a seq -> 'a list
  val toString : ('a -> string) -> 'a seq -> string
  val equal : ('a * 'a -> bool) -> 'a seq * 'a seq -> bool

  val empty : unit -> 'a seq
  val singleton : 'a -> 'a seq
  val tabulate : (int -> 'a) -> int -> 'a seq
  val fromList : 'a list -> 'a seq

  val rev : 'a seq -> 'a seq
  val append : 'a seq * 'a seq -> 'a seq
  val flatten : 'a seq seq -> 'a seq

  val filter : ('a -> bool) -> 'a seq -> 'a seq
  val map : ('a -> 'b) -> 'a seq -> 'b seq
  val map2 : ('a * 'b -> 'c) -> 'a seq -> 'b seq -> 'c seq
  val zip : 'a seq -> 'b seq -> ('a * 'b) seq

  val enum : 'a seq -> (int * 'a) seq
  val inject : (int * 'a) seq -> 'a seq -> 'a seq

  val subseq : 'a seq -> int * int -> 'a seq
  val take : 'a seq * int -> 'a seq
  val drop : 'a seq * int -> 'a seq
  val showl : 'a seq -> 'a listview
  val showt : 'a seq -> 'a treeview

  val iter : ('b * 'a -> 'b) -> 'b -> 'a seq -> 'b
  val iterh : ('b * 'a -> 'b) -> 'b -> 'a seq -> 'b seq * 'b
  val reduce : ('a * 'a -> 'a) -> 'a -> 'a seq -> 'a
  val scan : ('a * 'a -> 'a) -> 'a -> 'a seq -> 'a seq * 'a
  val scani : ('a * 'a -> 'a) -> 'a -> 'a seq -> 'a seq

  val sort : 'a ord -> 'a seq -> 'a seq
  val merge : 'a ord -> 'a seq -> 'a seq -> 'a seq
  val collect : 'a ord -> ('a * 'b) seq -> ('a * 'b seq) seq
  val collate : 'a ord -> 'a seq ord

end
```

ArraySequence	Work	Span

`empty ()`
`singleton a`
`length s`
`nth s i`
	$O(1)$	$O(1)$

`tabulate f n`
if `f i` has W_i work and S_i span
`map f s`
if `f` s_i has W_i work and S_i span, and $\|s\| = n$
`map2 f s t`
if `f` (s_i, t_i) has W_i work and S_i span, and $\|s\| = n$
	$O\left(\sum_{i=0}^{n-1} W_i\right)$	$O\left(\max_{i=0}^{n-1} S_i\right)$

`reduce f b s`
if `f` does constant work and $\|s\| = n$
`scan f b s`
if `f` does constant work and $\|s\| = n$
`filter p s`
if `p` does constant work and $\|s\| = n$
`showt s`
if $\|s\| = n$
`hidet tv`
if the combined length of the sequences is n
	$O(n)$	$O(\lg n)$

`sort cmp s` *if* `cmp` does constant work and $\|s\| = n$	$O(n \lg n)$	$O(\lg^2 n)$

`merge cmp s t`
if `cmp` does constant work, $\|s\| = n$, and $\|t\| = m$
`flatten s`
if if $s = \langle s_1, s_2, \ldots, s_k \rangle$ and $m + n = \sum_i \|s_i\|$
	$O(m + n)$	$O(\lg(m + n))$

`append (s,t)` *if* $\|s\| = n$, and $\|t\| = m$	$O(m + n)$	$O(1)$

Table/Set Operations	*Work*	*Span*
`size`(T) `singleton`(k, v)	$O(1)$	$O(1)$
`filter` f T	$O\Big(\sum_{(k,v)\in T} W(f(v))\Big)$	$O\Big(\lg \lvert T\rvert + \max_{(k,v)\in T} S(f(v))\Big)$
`map` f T	$O\Big(\sum_{(k,v)\in T} W(f(v))\Big)$	$O\Big(\max_{(k,v)\in T} S(f(v))\Big)$
`tabulate` f S	$O\Big(\sum_{k\in S} W(f(k))\Big)$	$O\Big(\max_{k\in S} S(f(k))\Big)$
`find` T k `insert` f (k, v) T `delete` k T	$O(\lg \lvert T\rvert)$	$O(\lg \lvert T\rvert)$
`extract` (T_1, T_2) `merge` f T_1 T_2 `erase` (T_1, T_2)	$O\big(m \lg(\frac{n+m}{m})\big)$	$O\big(\lg(n+m)\big)$
`domain` T `range` T `toSeq` T	$O(\lvert T\rvert)$	$O(\lg \lvert T\rvert)$
`collect` S `fromSeq` S	$O(\lvert S\rvert \lg \lvert S\rvert)$	$O(\lg^2 \lvert S\rvert)$
`intersection` (S_1, S_2) `union` (S_1, S_2) `difference` (S_1, S_2)	$O\big(m \lg(\frac{n+m}{m})\big)$	$O\big(\lg(n+m)\big)$

where $n = \max(|T_1|, |T_2|)$ and $m = \min(|T_1|, |T_2|)$. For `reduce` you can assume the cost is the same as `Seq.reduce f init (range(T))`. In particular `Seq.reduce` defines a balanced tree over the sequence, and `Table.reduce` will also use a balanced tree. For `merge` and `insert` the bounds assume the merging function has constant work.