Full Name: ______________________________

Andrew ID: ______________ **Section:** ______________

15–210: Parallel and Sequential Data Structures and Algorithms

Practice Exam I (Solutions)

February 2017

- There are 11 pages in this examination, comprising 6 questions worth a total of 99 points. The last few pages are an appendix detailing some of the 15-210 library functions and their cost bounds.
- 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.

Circle the section YOU ATTEND

Sections		
A	9:30am - 10:20am	Andra/Charles
B	10:30am - 11:20am	Aashir/Anatol
C	12:30pm - 1:20pm	Oliver
D	12:30pm - 1:20pm	Rohan/Serena
E	1:30pm - 2:20pm	John/Christina
F	1:30pm - 4:20pm	Vivek/Teddy
G	3:30pm - 5:20pm	Ashwin/Sunny

Full Name: ______________________ **Andrew ID:** ______________________

Question	Points	Score
Recurrences	20	
Short Answers	21	
Missing Element	12	
Interval Containment	13	
Quicksort	17	
Parentheses Revisited	16	
Total:	99	

Question 1: Recurrences (20 points)

Recall that $f(n)$ is $\Theta(g(n))$ if $f(n) \in O(g(n))$ and $g(n) \in O(f(n))$. Give a closed-form solution in terms of Θ for the following recurrences. Also, state whether the recurrence is dominated at the root, the leaves, or equally at all levels of the recurrence tree.

You do not have to show your work, but it might help you get partial credit.

(a) (4 points) $f(n) = 5f(n/5) + \Theta(n)$

Solution: $\Theta(n \lg n)$, balanced.

(b) (4 points) $f(n) = 3f(n/2) + \Theta(n^2)$

Solution: $\Theta(n^2)$, root-dominated.

(c) (4 points) $f(n) = f(n/2) + \Theta(\lg n)$

Solution: $\Theta(\lg^2 n)$, approximately balanced.

(d) (4 points) $f(n) = 5f(n/8) + \Theta(n^{2/3})$

Solution: $\Theta(n^{\lg_8 5})$ (roughly $\Theta(n^{0.77})$) leaf-dominated.

(e) (4 points) $f(n) = f(n/2) + f(n/4) + \Theta(\log n)$
This one does not need a closed form solution.

Solution: $\Theta(n^\alpha)$, where α satisfies $1 = (1/2)^\alpha + (1/4)^\alpha$. Roughly $\alpha = .69424$. Leaf-dominated.

Question 2: Short Answers (21 points)

(a) (5 points) Assume you are given a function f : `int Seq.t` $\times$ `int Seq.t` $\rightarrow$ `int Seq.t` where $f(A, B)$ requires $O\left((|A| + |B|)^2\right)$ work and $O(\log(|A| + |B|))$ span, and returns a sequence of length $|A| + |B|$. Give the work and span of the following function as tight Big-O bounds in terms of $|S|$.

```
fun foo S =
  Seq.reduce f (Seq.empty ()) (Seq.map Seq.singleton S)
```

Solution: Work: $O(|S|^2)$. Span: $O(\log^2 |S|)$

(b) (7 points) Suppose we implement a function `fastJoin` which has the same specification as the BST function `join`, except that it requires only $O(\log(\min(|T_1|, |T_2|)))$ work and span for inputs T_1 and T_2. Give the work and span of the following function as tight Big-O bounds in terms of $|S|$. Assume S is presorted by key.

```
fun bar S =
  Seq.scan Tree.fastJoin (Tree.empty ()) (Seq.map Tree.singleton S)
```

Solution: Work: $O(|S|)$. Span: $O(\log^2 |S|)$.

(c) (5 points) Implement `reduce` using contraction. You can assume the input length is a power of 2.

Solution:

```
fun reduce f b s =
     case length s
      of 0 => b
       | 1 => f(b, nth s 0)
       | n =>
         let
           val x = tabulate
           (fn i => case i = (n div 2) of
                    true => (nth s (2*i))
                     | _ => f(nth s (2*i), nth s (2*i + 1)))
                       (((n-1) div 2)+1)
         in reduce f b x
         end
```

(d) **Guessing Games** I am thinking of a random non-negative integer, X. Of course, I can't mean *uniformly* random, as that would mean that at least half the time I'm thinking of an infinite integer! As it turns out, the expected value of positive integers I think of is 1000.

i. (4 points) For some reason, I like to choose 15210 a lot. Give an upper bound on the probability with which I can choose $X = 15210$ (while still obeying the condition $\mathbf{E}[X] = 1000$).

Solution: From Markov's inequality,

$$\mathbf{Pr}\left[X \geq 15210\right] \leq \frac{E[X]}{15210}$$

So, $\mathbf{Pr}\left[X = 15210\right] \leq 1000/15210$. Equivalently, you could assume that I only think of the numbers 0 and 15210, and solve from there.

Question 3: Missing Element (12 points)

For 15210, there is a roster of n **unique** Andrew ID's, each a string of at most 9 characters long (so `String.compare` costs $O(1)$).

In this problem, the roster is given as a **sorted** string sequence R of length n. Additionally, you are given another sequence S of $n-1$ **unique ID's from** R. The sequence S is **not necessarily sorted**. Your task is to design and code a divide-and-conquer algorithm to find the missing ID.

(a) (7 points) Write an algorithm in SML that has $O(n)$ work and $O(\log^2 n)$ span.

```
open ArraySequence
fun missing_elt(R: string seq, S: string seq) : string =
let fun lessThan a b = (String.compare(b, a)=LESS) (* is b<a? *)
in
   case (length R)
    of 0 => raise Fail "should not get here"
     | 1 => nth R 0
     | n =>
       let val p = nth R (n div 2)
           val Sleft = filter (lessThan p) S
           val Sright = filter (not o (lessThan p)) S
           val Rleft = take (R, n div 2)
           val Rright = drop (R, n div 2)
       in if (length Sleft < length Rleft) then
              missing_elt (Rleft, Sleft)
          else
              missing_elt (Rright, Sright)
       end
end
```

(b) (5 points) Give a brief justification of why your algorithm meets the cost bounds.

Solution: We maintain the variant that $|R| = |S| + 1$. The body of the function contains only `filter`, `take`, and `drop`, which have $\Theta(|R|)$ work and $\Theta(\log |R|)$ span. Furthermore, the algorithm makes only one recursive call on the problem of size $|R|/2$, so we have $W(n) = W(n/2) + \Theta(n)$ and $S(n) = S(n/2) + \Theta(\log n)$. These recurrences solve to $W(n) = \Theta(n)$ and $S(n) = \Theta(\log^2 n)$.

Question 4: Interval Containment (13 points)

An interval is a pair of integers (a, b). An interval (a, b) is contained in another interval (α, β) if $\alpha < a$ and $b < \beta$. In this problem, you will design an algorithm

`count: (int * int) seq` $\rightarrow$ `int`

which takes a sequence of intervals (i.e., ordered pairs) $(a_0, b_0), (a_1, b_1), \ldots, (a_{n-1}, b_{n-1})$ and computes the number of intervals that are contained in some other interval. If an interval is contained in multiple intervals, it is counted only once.

For example, `count` $\langle(0, 6), (1, 2), (3, 5)\rangle = 2$ and `count` $\langle(1, 5), (2, 7), (3, 4)\rangle = 1$. Notice that the interval $(3, 4)$ is contained in both $(1, 5)$ and $(2, 7)$, but the count is 1.

You can assume that the input to your algorithm is sorted in increasing order of the first coordinate and that all the coordinates (the a_i's and b_i's) are distinct.

(a) (5 points) Give a brute force solution to this problem (code or prose).

Solution:

```
open ArraySequence

fun count s =
  let fun or (p,q) = p orelse q
      fun inOther (a,b) =
        reduce or false (map (fn (x,y) => (x < a) andalso (b < y)) s)
  in reduce (op+) 0 (map (fn iv => if inOther iv then 1 else 0) s)
  end
```

(b) (8 points) Design an algorithm that has $O(n)$ work and $O(\log n)$ span. Carefully explain your algorithm; you don't have to write code. Hint: The algorithm is short.

Solution:

```
open ArraySequence

fun count (s : (int*int) seq) =
  let val ends = map (fn (_,b) => b) s
      val (maxCovered, _) = scan Int.max (Option.valOf Int.minInt) ends
      fun inOther ((a,b), covered) =
          if b < covered then 1 else 0

  in reduce (op+) 0 (zipWith inOther (s, maxCovered))
  end
```

Question 5: Quicksort (17 points)

Assume throughout that all keys are distinct.

(a) (3 points) TRUE or FALSE. In randomized quicksort, each key is involved in the same number of comparisons.

Solution: FALSE

(b) (7 points) What is the probability that in randomized quicksort, a random pivot selection on an input of n keys leads to recursive calls, both of which are no smaller than $\frac{n}{16}$? Show your work.

Solution: $\frac{7}{8}$

(c) (7 points) Consider running randomized quicksort on a permutation of $1, \ldots, n$. What is the probability that a quicksort call tree has height exactly n? Note: the height of a tree is the number of nodes on its longest path.

Solution: This happens only when we pick the maximum or the minimum element in the input repeatedly. The probability of that is:

$$\frac{2}{n} \times \frac{2}{n-1} \times \cdots \times \frac{2}{2} = \frac{2^{n-1}}{n!}$$

Question 6: Parentheses Revisited (16 points)

A parenthesis expression is called *immediately paired* if it consists of a sequence of open-close parentheses — that is, of the form ”()()()() . . . ()”.

(a) (8 points) **Longest immediately paired subsequence (LIPS) problem.** Given a (not necessarily matched) parenthesis sequence s, the longest immediately paired subsequence problem requires finding a (possibly non-contiguous) longest subsequence of s that is immediately paired. For example, the LIPS of “(((((((**()()()**)))**()**((**()**(**()**)” is “()()()()()()” as highlighted in the original sequence.

Write a function that computes the *length* of a LIPS for a given sequence. Your function should have $O(n)$ work and $O(\lg n)$ span.

(**Hint:** Try to find a property that simplifies computing LIPS. This problem might be difficult to solve otherwise.)

```
datatype paren = L | R
fun findLIPS (s: paren Seq.t) : int =
```

Solution: The algorithm simply extracts immediately paired parentheses and counts them. We prove below why this is sufficient.

```
fun findLIPS (s: paren Seq.t) =
    let
      fun isIP i =
          case (Seq.nth s i, Seq.nth s (i+1))
            of (L, R) => 2
             | _ => 0
    in
      Seq.reduce op+ 0 (Seq.tabulate isIP (Seq.length s - 1))
    end
```

(b) (8 points) Prove succintly that your algorithm correctly computes LIPS.

Solution: Consider any parenthesis expression and let () be an immediately paired parenthesis in the result. Let i and j be the positions of the parenthesis in the original sequence. Note that $i < j$. Let k be the leftmost RPAREN and note that $i < k \leq j$ and the parenthesis at $k-1$ and k are immediately paired. In other words, there exists one immediately paired parentheses in the contiguous subsequence defined by i and j, e.g., “(....()....)”, “(....()”, “()...)”. It thus suffices to count the immediately paired parenthesis in the input.

%inputtreaps

Appendix: Library Functions

```
signature SEQUENCE =
sig
  type 'a t
  type 'a seq = 'a t
  type 'a ord = 'a * 'a -> order
  datatype 'a listview = NIL | CONS of 'a * 'a seq
  datatype 'a treeview = EMPTY | ONE of 'a | PAIR 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 zip : 'a seq * 'b seq -> ('a * 'b) seq
  val zipWith : ('a * 'b -> 'c) -> 'a seq * 'b seq -> 'c seq

  val enum : 'a seq -> (int * 'a) seq
  val filterIdx : (int * 'a -> bool) -> 'a seq -> 'a seq
  val mapIdx : (int * 'a -> 'b) -> 'a seq -> 'b seq
  val update : 'a seq * (int * 'a) -> 'a seq
  val inject : 'a seq * (int * '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 splitHead : 'a seq -> 'a listview
  val splitMid : 'a seq -> 'a treeview
  val iterate : ('b * 'a -> 'b) -> 'b -> 'a seq -> 'b
  val iteratePrefixes : ('b * 'a -> 'b) -> 'b -> 'a seq -> 'b seq * 'b
  val iteratePrefixesIncl : ('b * 'a -> 'b) -> 'b -> 'a seq -> 'b seq
  val reduce : ('a * 'a -> 'a) -> 'a -> 'a seq -> 'a
  val scan : ('a * 'a -> 'a) -> 'a -> 'a seq -> 'a seq * 'a
  val scanIncl : ('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
  val argmax : 'a ord -> 'a seq -> int

  val $ : 'a -> 'a seq
  val % : 'a list -> 'a seq
end
```

ArraySequence	Work	Span

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

`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 $\|s\| = n$
`zipWith f (s, t)`
if $f(s[i], t[i])$ has W_i work and S_i span, and $\min(\|s\|, \|t\|) = n$
	$O\left(\sum_{i=0}^{n-1} W_i\right)$	$O\left(\max_{i=0}^{n-1} S_i\right)$

`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 $\|s\| = n$
	$O(n)$	$O(\lg n)$

`flatten s`	$O\left(\sum_{i=0}^{n-1} \left(1 + \|s[i]\|\right)\right)$	$O(\lg \|s\|)$
`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 $\|t\| = m$
	$O(m + n)$	$O(\lg(m + n))$

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