Circle the section YOU ATTEND

<table>
<thead>
<tr>
<th>Sections</th>
<th>Time</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>9:30am - 10:20am</td>
<td>Andra/Charles</td>
</tr>
<tr>
<td>B</td>
<td>10:30am - 11:20am</td>
<td>Aashir/Anatol</td>
</tr>
<tr>
<td>C</td>
<td>12:30pm - 1:20pm</td>
<td>Oliver</td>
</tr>
<tr>
<td>D</td>
<td>12:30pm - 1:20pm</td>
<td>Rohan/Serenia</td>
</tr>
<tr>
<td>E</td>
<td>1:30pm - 2:20pm</td>
<td>John/Christina</td>
</tr>
<tr>
<td>F</td>
<td>1:30pm - 4:20pm</td>
<td>Vivek/Teddy</td>
</tr>
<tr>
<td>G</td>
<td>3:30pm - 5:20pm</td>
<td>Ashwin/Sunny</td>
</tr>
<tr>
<td>Question</td>
<td>Points</td>
<td>Score</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>Recurrences</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Short Answers</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>Missing Element</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Interval Containment</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>Quicksort</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>Parentheses Revisited</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>99</td>
<td></td>
</tr>
</tbody>
</table>
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 \( \Theta \) 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) \( f(n) = 5f(n/5) + \Theta(n) \)

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

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

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

(e) (4 points) \( f(n) = f(n/2) + f(n/4) + \Theta(\log n) \)
   This one does not need a closed form solution.
Question 2: Short Answers  (21 points)

(a) (5 points) Assume you are given a function $f : \text{int Seq.t} \times \text{int Seq.t} \rightarrow \text{int Seq.t}$ where $f(A, B)$ requires $O((|A| + |B|)^2)$ 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 $\text{Big-O}$ bounds in terms of $|S|$.

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

(b) (7 points) Suppose we implement a function $\text{fastJoin}$ which has the same specification as the BST function $\text{join}$, 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 $\text{Big-O}$ 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)
```

(c) (5 points) Implement $\text{reduce}$ using contraction. You can assume the input length is a power of 2.
(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 $E[X] = 1000$).
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 \texttt{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.

\[
\text{fun missingElt (R: string Seq.t, S: string Seq.t): string =}
\]
\[
\begin{align*}
&\text{let fun lessThan a b = (String.compare(b, a) = LESS) \% is b < a?} \\
&\text{in} \\
&\text{case (length R) of}
\end{align*}
\]
\[
\begin{align*}
&0 \Rightarrow \text{raise Fail "should not get here"} \\
&1 \Rightarrow \text{______________________________} \\
&n \Rightarrow \% \text{recursive step}
\end{align*}
\]
\[
\begin{align*}
&\text{let val p = ________________________________} \\
&\text{val Sleft = Seq.filter (lessThan p) S} \\
&\text{val Sright = Seq.filter (not o (lessThan p)) S} \\
&\text{val Rleft = ________________________________} \\
&\text{val Rright = ________________________________} \\
&\text{in} \\
&\text{______________________________} \\
&\text{______________________________} \\
&\text{______________________________} \\
&\text{______________________________}
\end{align*}
\]
\[
\text{end}
\]

(b) (5 points) Give a brief justification of why your algorithm meets the cost bounds.
Question 4: Interval Containment  (13 points)

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

\[
\text{count: } (\text{int } \times \text{int}) \text{ seq } \rightarrow \text{int}
\]

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, \(\text{count } \langle (0, 6), (1, 2), (3, 5) \rangle = 2\) and \(\text{count } \langle (1, 5), (2, 7), (3, 4) \rangle = 1\). 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).

(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.
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.

(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.

(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.
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.)

```plaintext
datatype paren = L | R

fun findLIPS (s: paren Seq.t) : int =
```

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

ArraySequence

<table>
<thead>
<tr>
<th>Method</th>
<th>Work</th>
<th>Span</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty ()</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>singleton a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>length s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nth s i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subseq s (i, len)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tabulate f n</td>
<td>O(∑(W_i))</td>
<td>O(∑(max.S_i))</td>
</tr>
<tr>
<td>map f s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>zipWith f (s, t)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reduce f b s</td>
<td>O(n)</td>
<td>O(lg n)</td>
</tr>
<tr>
<td>scan f b s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>filter p s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flatten s</td>
<td>O(∑(1 +</td>
<td>s[i]</td>
</tr>
<tr>
<td>sort cmp s</td>
<td>O(n lg n)</td>
<td>O(lg^2 n)</td>
</tr>
<tr>
<td>merge cmp (s, t)</td>
<td>O(m + n)</td>
<td>O(lg(m + n))</td>
</tr>
<tr>
<td>append (s, t)</td>
<td>O(m + n)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>