Your Name:

<table>
<thead>
<tr>
<th>1.1</th>
<th>1.2</th>
<th>1.3</th>
<th>2.1</th>
<th>2.2</th>
<th>2.3a</th>
<th>2.3b</th>
<th>3.1</th>
<th>3.2</th>
<th>3.3</th>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>[5]</td>
<td>[5]</td>
<td>[20]</td>
<td>[10]</td>
<td>[10]</td>
<td>[10]</td>
<td>[10]</td>
<td>[15]</td>
<td>[10]</td>
<td>[5]</td>
<td>[100]</td>
</tr>
</tbody>
</table>

CRN 21736/21751  CIS 4/513  Winter 2017

Final Exam
(due Wednesday March 22 by 13:10.)

This is the usual “open-everything, no web search and no outside help” take-home test. Provide answers with their justifications. Your solutions are due in my mailbox (in 120 Deschutes) around 1 pm. If you are not able to comply with the format specified in the Midterm (such solutions will be returned ungraded), please submit your typeset solutions to me via email.

Check https://classes.cs.uoregon.edu/17W/cis413/news.html, where I will post “frequently asked questions” about the test – but communicate those questions to me only, via private email messages.

1. Cutting and Linking Trees.

Label the nodes of the tree $T$ in Figure 5.5(a) (represented as a collection of paths) by upper case letters A, B, ..., W assigned in the breadth-first manner (as are the lower-case labels in the Figure.) The upper case labels are to avoid conflict of identifiers during execution of expose which uses lower-case letters as variables.

1. Draw the tree $T$ as defined above.

2. Note that a virtual tree has the splay trees in the head-to-tail order, so that three of the trees in Figure 5.5(b) are incorrect. Draw a correct virtual tree $T'$ representing $T$.

3. Draw the sequence of virtual trees $T', T'', ...$ representing the actual tree $T$ during execution of expose($V$).
2. Disjoint Sets

*ADT Disjoint Sets* (DS with operations MakeSet, Union and Find) implemented by trees with “Union by rank” and “Find with path compression” have almost constant (but growing) amortized complexity per operation.

Binomial trees are defined as follows:

- $B_0$ has one node,
- $B_{i+1}$ consists of a $B_i$ whose root has another $B_i$ as a principal subtree.

1. Prove that the amortized complexity of DS with arbitrary Union (no rank involved) and Find with path compression is constant when all Union operations precede any Find operation.

2. Consider DS with arbitrary Union and Find with path compression and no constraint on the string of operations. Prove that the amortized complexity of this implementation is $\Omega(\log n)$ by first building (by a series of arbitrary Unions) a DS $S$ with $n = 2^k - 1$ elements as a Binomial Tree of rank $k - 1$ and then iterating the following string of operations: \{S:=Union(S, MakeSet(i)), Find(j)\}, where $i$ varies from $n/2 + 1$ to $n$ and $j$ is the deepest node in $S$.

3. The handout linked from the news page (see above) defines the *Depth Determination* problem, where an efficient solution via non-isomorphic representation improves the amortized complexity of operation FIND-DEPTH from linear to almost constant.

(a) Show how to modify Find to implement FIND-DEPTH. Your implementation should perform path compression, and its running time should be linear in the length of the find path. Make sure that your implementation updates pseudo-distances correctly.

(b) Show how to implement GRAFT($r, v$), which combines the sets containing $r$ and $v$ (these sets are Find($r$) and Find($v$)), by modifying the UNION procedure. Make sure that your implementation updates pseudo-distances correctly. Note that the root of a set $S_i$ is not necessarily the root of the corresponding tree $T_i$. 


3. Priority Queues

*Heapify* creates a *Priority Queue (PQ)* from a list of PQs.

A tree has *Heap Property (HP)* if every node other than the root has key not smaller than its parent’s key.

1. (a) Assume that the element keys are 1, 2, ..., n given on-line as a random permutation. Prove that when PQ is implemented as a Binary Heap (an almost complete binary tree with HP), the complexity of on-line *Heapify* (by iterated insertions into the originally empty PQ) is $\Omega(n \log n)$.

(b) Prove the complexity of an off-line *Heapify* bottom-up algorithm, whereby the original almost complete binary tree $T$ containing all the elements acquires the HP by propagating the invariant: “the smallest $n - i$ subtrees of $T$ have HP” ($i = 0, ..., n - 1$).

2. (a) *Mergeable PQ* can be implemented as *Eager Binomial Heap*, a forest of Binomial Trees, no two of the same rank: What is the complexity of *Heapify* by iterated insertions?

(b) *Lazy Binomial Heap* implementation does not require that all binomial trees have different ranks. What is the complexity of *Heapify* by iterated insertions?

3. How does *lazy deletion* (marking a node deleted in constant time) affect the amortized complexity of *Mergeable PQ*?

4. Graduate students only

Amortized complexity of splaying $1 + 3(rank(x') − rank(x))$ that maintains credit invariant based on $rank(x) = \lfloor \log(tw(x)) \rfloor$ assumes that $tw(x)$ is defined by sum of individual weights of $x$’s descendants in the splay tree. Is this the case of analysis of Sleator and Tarjan in the handout quoted in Homework #5?