CSc 345: Analysis of Discrete Structures
Spring 2018 (Lewis)

Final Exam
Mon 7 May 2018

Name: ____________________________________________ NetID: ______________________

Person to your left: ________________________ Person to your right: ______________________

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short Answer</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>AVL Rebalancing</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Quicksort</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Pretty Pictures</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Induction</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Graph Algorithms</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>98</td>
<td></td>
</tr>
</tbody>
</table>

(I’m giving the class 2 bonus points, because a lot of you filled out your TCE in the last few days. Thanks!)
1. (a) (3 points) Write out the formal definition of the set \( o(g(n)) \).

(b) (6 points) Suppose that the predicate \( Q(x, y) \) means that “\( x \) gave a gift to \( y \).” For each quantification below, restate the quantification in English. **Do not use the phrases “for all” or “exists” in your explanation.**

\[ \forall x \exists y Q(x, y) \]

\[ \forall y Q(SantaClaus, y) \]

\[ \exists x \forall y Q(x, y) \]

\[ \exists x \forall y Q(y, x) \]

\[ \exists x, y (Q(x, y) \land \neg Q(y, x)) \]

\[ \exists x \forall y \neg Q(x, y) \]
(c) (2 points) When you write a Java method in the \texttt{x=change(x)} style (which modifies a BST), why should it be \texttt{static}?

(d) (2 points) I said in class that a B-tree, with a very large node width, was very useful for storing data on a disk. What about a B-Tree makes it better (for this purpose) than a BST?

(e) (2 points) In a max-heap, what is required about the shape of the tree?

(f) (2 points) In a max-heap, what is required about the keys of parent and child nodes throughout the tree?

(g) (2 points) What is the difference between “amortized” and “average” time?
(h) (2 points) Counting Sort is an improvement on Bucket Sort. **Briefly** explain what Counting Sort does, which makes it better than Bucket Sort.

(i) (2 points) The “linear sorts” can be very fast because they place some limitations on the input. What are the limitations?

(j) (2 points) What is the difference between a digraph and an undirected graph?

(k) (2 points) The graph below is an NFA, not a DFA. Explain how you know that this is true.
(l) (2 points) Why is it impossible to build a nondeterministic computer in practice?

(m) (3 points) Each of the recurrences below cannot be solved by the Master Method. For each one, explain why the Master Method cannot be used.

If you have to perform some calculations to answer these parts, then show your work.

\[ T(n) = 4T\left(\frac{n}{2}\right) + n^2 \lg n \]

\[ T(n) = 3T\left(\frac{4n}{3}\right) + \sqrt{n} \]

\[ T(n) = 2T(n - 1) + n \]

\(^1\)(As we’ve presented it in class. Don’t use any tricks that are more advanced than what Russ knows about!)
(n) (2 points) In QuickSort and Merge Sort, we often use a simpler sort (such as Insertion Sort) to sort the individual blocks of data, after we have broken them down into small enough pieces. What is the asymptotic time cost of running Insertion Sort on one of these blocks, assuming that there are no more than 8 elements in the block? Explain. (This is a trick question.)

(o) (2 points) In a 2-3-4 tree, how are the number of keys and the number of children related?

2. (10 points) (Write your solution on the next page.)

Write a function, in the $x=\text{change}(x)$ style, that will check to see if an AVL shape violation exists at the current location. If no violation exists, then it should return an unmodified subtree; if one does, then the function must perform all required rotations and return the root of the updated subtree. (If grandchildren tie for the “cause” of an imbalance, then follow the same strategy we did in the project.)

Your function must run in $O(1)$ time.

The type of the AVL nodes is named $\text{AVLNode}$. 

To save space, you may skip over the possibility that the right child might be larger (since it is just a mirror of the left side). However, to make clear where you would check this, include the line

```--- MIRROR CODE HERE ---
```

at the proper location.

You may assume (don’t double-check these):

- The parameter is not null.
- The $\text{height}$ field of all nodes in the subtree (including the root) are correct when your function begins.
- You have a $\text{getHeight()}$ method, which returns the height of a node (or -1 if the parameter is null).
- You have $\text{rotateLeft(), rotateRight()}$ methods, both written in the $x=\text{change}(x)$ style, which perform rotations (including updating height fields).

Page 6
Write your solution here:
3. (a) (2 points) On average, what is the runtime of QuickSort? In the worst case, what is its runtime? Why are they different?

(b) (3 points) What is the median-of-3 algorithm, and how does it work? (We’ll ask why you might use it next.)

(c) (3 points) Naive QuickSort implementations always use the first element in the array as the pivot. Give an example of a common, simple type of input data which would have terrible performance if we used this choice; explain why that is. Then, explain why the median-of-3 algorithm works so much better for that input.

(d) (3 points) Does the median-of-3 algorithm guarantee that QuickSort will have good performance in all cases? Why or why not?
(e) (5 points) Fill in the missing code in the implementation of QuickSort’s `partition()` algorithm below.

This function must partition the data, and finally move the pivot into place (between the two partitions); it returns the index of the pivot. Since this is used inside QuickSort, it might not partition the entire array; it has parameters which indicate what subset of the array to partition.

Other notes:
- For simplicity, this uses the first element as the pivot, even though that sometimes is terrible.
- Assume that we have a `swap()` function which takes as inputs the array, and the two index positions.
- The `end` parameter is exclusive. (`start` is inclusive.)

```c
int quicksort_partition(int[] array, int start, int end)
{
    int pivot = array[start];
    int A = start+1;
    int B = end-1;

    while (________________________________________)
    {
        while (____________________________________)
        A++;

        while (____________________________________)
        B--;

        if (_______________________________________)
            swap(array, A,B);
    }

    if (_______________________________________)
        swap(array, start,A);

    return (___________);
}
```
4. (a) (4 points) A red-black tree emulates a 2-3-4 tree by replacing each node with a “widget.” Draw all of the possible widgets. (If there are multiple versions of a widget, which are mirrors of each other, draw all of the mirrored versions.) Mark red nodes with an ‘R’; add arbitrary keys to each node - but honor the BST property.

(b) (4 points) Perform the CoolSort Transform on the root of the following tree. (Only perform it once; you do not have to perform it multiple times and find the minimum in the tree.)
5. (15 points) Use **induction** to prove that $n^3 - n$ is divisible by 3 for all $n \in \mathbb{Z}^+$. 
6. (a) (5 points) Use Dijkstra’s algorithm in the graph below, with node $S$ as the source, to compute the distance to every node (the cost of the shortest path). Write the answer in the table below.

Answer:

<table>
<thead>
<tr>
<th>Vertex</th>
<th>S</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) (3 points) What is the shortest path from $S$ to $a$? (List the vertices of the path.)

(c) (5 points) Use Prim’s algorithm to find a minimum-cost spanning tree (MST) in this graph. Start at vertex $S$.

Draw the spanning tree, next to the original graph. To make it easy to grade, **please arrange the vertices in the same shape!**