Final Exam
Mon 7 May 2018

Solutions

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><strong>Total:</strong></td>
<td><strong>98</strong></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 $o(g(n))$.

**Solution:**

$$\{ f : \forall c > 0, \exists n_0 > 0, \forall n \geq n_0 (0 \leq f(n) \leq cg(n)) \}$$

(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.**

$\forall x \exists y Q(x, y)$

**Solution:** NOTE: Technically, the predicate $Q(x, x)$ is legal and covered by the quantification. However, we won’t enforce this when grading - if students assume that people only give gifts to others, that’s OK.

Everybody gave at least one gift to somebody.

$\forall y Q(SantaClaus, y)$

**Solution:** Santa Claus gave gifts to everybody.

$\exists x \forall y Q(x, y)$

**Solution:** Somebody gave gifts to every single person.

$\exists x \forall y Q(y, x)$

**Solution:** Somebody received gifts from all of the other people.

$\exists x, y (Q(x, y) \land \neg Q(y, x))$

**Solution:** Somebody gave a gift, who did not receive one back from the same person.

$\exists x \forall y \neg Q(x, y)$

**Solution:** Somebody did not give any gifts to anybody.
(c) (2 points) When you write a Java method in the `x=change(x)` style (which modifies a BST), why should it be static?

**Solution:** Because the parameter might be null (representing an empty tree). If the method was a member method, we would get `NullPointerException`.

(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?

**Solution:** Because it is so wide, it has very few layers; that is, it takes very few steps to get from the root to any leaf.

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

**Solution:** It must not have “holes” - that is, every layer must be full (except for the last), and the last layer must fill up from left to right.

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

**Solution:** The parent’s key must be greater-or-equal than the keys of both of its children.

(g) (2 points) What is the difference between “amortized” and “average” time?

**Solution:** Amortized time is the average, **in a single run**, over many operations; individual operations may be slow, but the average of all of them is good.

Average time is the average **over many runs** - individual runs may be terribly slow, but over many runs, it averages out.
(h) (2 points) Counting Sort is an improvement on Bucket Sort. **Briefly** explain what Counting Sort does, which makes it better than Bucket Sort.

**Solution:** It pre-scans the data, counting how many elements show up in each of the bins. This allows it to know, on the second pass, where to copy each element (since it knows exactly where, in the final array, each bin will begin).

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

**Solution:** There must be a small number of keys (or a small number of categories of keys) - and the sorting algorithm must know them ahead of time.

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

**Solution:** In a digraph, the edges have direction. In an undirected graph, they do not have direction and thus are symmetric.

(k) (2 points) The graph below is an NFA, not a DFA. Explain how you know that this is true.

**Solution:** At node 0, there are two outbound arrows for the symbol 'a'.
(l) (2 points) Why is it impossible to build a nondeterministic computer in practice?

**Solution:** Because you would need to build exponentially much hardware, to execute the many different copies of the machine.

(m) (3 points) Each of the recurrences below cannot be solved by the Master Method\(^1\) 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**.

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

**Solution:**

\[a = 4\]
\[b = 2\]
\[\log_b a = 2\]
\[f(n) = n^2 \lg n\]

Since \(f(n)\) and \(n^{\log_b a}\) are not polynomially different, the Master Method doesn’t know how to solve this.

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

**Solution:**

\[b = \frac{3}{4}\]

Since \(b < 1\), Master Method cannot solve this.

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

**Solution:** This is not in the standard form (subtraction instead of division).

\(^1\) (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.)

Solution: $O(1)$ - because there are a constant (max) number of inputs. When there is some constant max on the problem size, the asymptotic time cost is also constant.

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

Solution: The number of children must be exactly 1 more than the number of keys, in each node. (Optional note:) Except for leaves.

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 $\text{rotateLeft()}, \text{rotateRight()}$ methods, both written in the $x=\text{change}(x)$ style, which perform rotations (including updating height fields).
Write your solution here:

**Solution:** **Instructor’s Note:** We will not mark off for missing `static` keyword. However, the parameter type and return type are important. (Also, we do not care if the student uses public, private, or omits either one.)

```java
private static AVLNode rebalance(AVLNode node)
{
    int diff = getHeight(node.left) - getHeight(node.right);

    if (diff > 1) // left side too large
    {  
        if (getHeight(node.left.right) > getHeight(node.left.left))
            node.left = rotateLeft(node.left);
        return rotateRight(node);
    }

    --- MIRROR CODE HERE ---

    return node;
}
```
3. (a) (2 points) On average, what is the runtime of QuickSort? In the worst case, what is its runtime? Why are they different?

**Solution:** Average: $O(n \lg n)$  Worst: $O(n^2)$
While (on average) the pivot (very roughly) splits the data in half, occasionally we get unlucky (many times in a row), and end up always choosing near-extreme pivot values. Thus, we end up recursing $O(n)$ times instead of $O(\lg n)$ times.

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

**Solution:** We look at the first, last, and middle values as three candidates for the pivot; we choose the median of them (by value, not position) as the pivot.

(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.

**Solution:** Sorted data is terrible, because (on every recursive call), we would choose the first (which is also the min) as the pivot - and thus, one of the two partitions would be empty! Median-of-3 solves this because it includes the middle value as a candidate pivot; in this case, it chooses the perfect pivot instead of the worst one.

(d) (3 points) Does the median-of-3 algorithm guarantee that QuickSort will have good performance in all cases? Why or why not?

**Solution:** No. We can still get very unlucky, where (on every recursive call) all 3 of the candidates we consider are very extreme (close to the min or max).
(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 (___________);
}
```

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

while (_____ A < B _____)
{
    while (_____ A < B && array[A] <= pivot _____)  
        A++;
    
    while (_____ A < B && array[B] >= pivot _____)  
        B--;
    
    if (_____ A < B _____)
        swap(array, A,B);
}

if (_____ A != start _____)
    swap(array, start,A);

return (_____ A _____);
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.

Solution: Instructor’s Note: I don’t care about the keys, so long as they are sorted inside of each widget. They do not need to use the same keys in each different version.

- 10 20 10 20
  - / \ / \ / \ 
  - 10 R 20 R 10 R 30 R

(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.)

Solution:
5. (15 points) Use induction to prove that $n^3 - n$ is divisible by 3 for all $n \in \mathbb{Z}^+$. 

**Solution:**

**Base:** $n = 1$

$1^3 - 1 = 0$, which is divisible by 3.

**Inductive:**

Assume that the conjecture holds for some $n = k$. We will try to prove that it also holds for $n = k + 1$.

\[
\begin{align*}
(k + 1)^3 - (k + 1) \\
k^3 + 3k^2 + 3k + 1 - k - 1 \\
k^3 + 3k^2 + 2k \\
(k^3 - k) + 3k^2 + 3k
\end{align*}
\]

By the I.H., we know that $k^3 - k$ is divisible by 3 - and both of the other terms have a coefficient of 3. Thus, the whole expression is divisible by 3, and thus the conjecture holds for $n = k + 1$.

**Summary:**

Thus, the conjecture holds 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.

```
<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>
```

Solution:
```
<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>11</td>
<td>3</td>
<td>12</td>
<td>7</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.)

**Solution:** $S, e, c, b, a$

(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!