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>Sort Algos</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Time Costs</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Misc, Page 1</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>Misc, Page 2</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Misc, Page 3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Master Method</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Quantifiers</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Induction (choose 1 of 2)</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
1. (5 points) Give the name of each of the sort algorithms described below.

Move the values in the array around, until all of the ones less than some important value are on the left, and all of the ones greater are on the right. Then recurse into both sides.

Divide the array into “sorted” and “unsorted” sections. Repeat $n$ times: scan the unsorted section for its smallest value. Then swap that value into place, making the sorted section one element larger.

Divide the array into “sorted” and “unsorted” sections. For each element in the “unsorted” section, slide that element left (one step at a time) until it is in the proper location, compared to all of the other elements already in the “sorted” section.

Swap values through the array until it is in a special arrangement that is optimized for removing the maximum value. Then, $n$ times, remove the maximum value from that data structure; store the values that you remove back into the array, starting at the right end.

Divide the array into upper and lower halves. Recurse into each half (that is, sort each one). Once both halves are sorted, mix them together to form a single, unified array, which is now entirely sorted.

2. For each algorithm below, give the time that it takes to run, on average. If its worst case is worse than its average, then give the worst case as well.

(a) (2 points) Quicksort

(b) (2 points) BST insert

(c) (2 points) Insert into a Max-Heap

(d) (2 points) Merge Sort

(e) (2 points) Counting Sort (one iteration, non-recursive)
3. (a) (4 points) When we append elements into an array, doubling the size each time that it fills, up, we get some unexpected performance results. What is the...

Worst-case time to insert one additional element in the array?

Worst-case time to insert a total of \( n \) additional elements in the array, starting with an empty array and then inserting them one at a time?

**Amortized** worst-case time to insert one additional element in the array?

(b) (5 points) Give the set-builder definition for big-Oh.

(c) (4 points) BSTs and max-heaps have different ordering guarantees (that is, rules about the values stored in their nodes). What is the ordering guarantee for a BST, and what is it for a max-heap?

(d) (4 points) BSTs and heaps are both (conceptually) binary trees. What is unusual about how a heap is physically stored, and what has to be true about the heap to make this possible?
4. (a) (3 points) What is the difference between Strong Induction and Weak Induction?

(b) (6 points) Using the asymptotic operators, translate each of the following English statements into symbols:

“f(n) and g(n) grow at essentially the same rate”

“f(n) definitely, without a doubt, grows much more slowly than g(n).”

“f(n) is lower bounded by g(n).”

(c) (5 points) Assume that we have an algorithm whose runtime is represented by the recurrence

$$T(n) = 4T(n/2) + n^2$$

What do the 4, 2, and $n^2$ represent?
5. (a) (6 points) Assume that I already know what the “bubble up” and “bubble down” algorithms are for a max-heap. Briefly explain how to perform an insert of a single value into an existing max-heap.

Now, explain how to perform the remove-max operation.

(b) (9 points) Using the formal definition of little-Oh, prove the following:
If \( f(n) = o(g(n)) \) and \( g(n) = o(h(n)) \), then \( f(n) = o(h(n)) \).
6. (10 points) For each part below, use the Master Method to solve the recurrence. If the recurrence cannot be solved by the Master Method, give a short explanation why not.

Show your work, or you will not receive any credit.

(a) \( T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n} \)

(b) \( T(n) = 9T\left(\frac{n}{3}\right) + n \lg n \)

(c) \( T(n) = 4T(n - 2) + n^2 \)

(d) \( T(n) = 4T\left(\frac{n}{4}\right) + n\sqrt{n} \)
7. Assume that we have a predicate $W(x,y)$, where $x, y$ are chess players. The predicate is defined as:

$W(x,y)$ is true iff player $x$ has won at least one game against player $y$.

For each English language statement below, write the quantified expression which represents it.

(a) (2 points) There is a player out there, somewhere, who has never won a game against anyone.

(b) (4 points) There is a player out there, somewhere, who has won games against every other player.

**NOTE:** $W(x,x)$ is false for all $x$. For full credit, make sure your solution accounts for this complexity!

(c) (2 points) At least one player has won at least one game.

(d) (4 points) There is some poor player who has lost a game to every other player. (Again, remember to think about the $W(x,x)$ case.)

(e) (2 points) The player “Dave” has won at least one game.
Choose only one of the induction problems to do. Do not do both - if you do, we will only grade one of them!

8. (15 points) Consider a (non-empty) binary tree; assume that every node in the tree either has zero children, or it has exactly two. (Don’t worry how we built this, just assume that it exists.)

Using structural induction, prove that the total number of nodes in the tree is odd. (You may use any of the structural induction strategies.)

Do not assume that the tree is balanced, or completely full. It can have lots of holes!
Choose only one of the induction problems to do. Do not do both - if you do, we will only grade one of them!

Prove the following conjecture using induction:

$$\sum_{i=0}^{n-1} ar^i = \frac{a(1 - r^n)}{1 - r}, \text{ where } n \in \mathbb{Z}^+.$$  

**NOTE:** Assume that $r \neq 1.$