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

Final Exam  
Tue 11 Dec 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>Definitions</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Short Answer, Part 1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Short Answer, Part 2</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Building Quantifications</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Explaining Theta</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>.dot Files</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Identifying Sort Algos</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>BST Algos</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Induction (choose 1 of 2)</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Building an NFA</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>A Graph Algorithm</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>97</strong></td>
<td></td>
</tr>
</tbody>
</table>

**TCE BONUS:** 3 points. Congrats!
1. (20 points) I have provided a list of terms on the last page of this exam. Match the **most appropriate** term to each concept. Not all terms are used. No term is used twice.

  A property of some sorting algorithms; indicates that, if there are any duplicate keys in the data, then the sorting algorithm will keep them (and their satellite data) in the same relative order.

  A bug where the outcome of a calculation depends on the order in which two or more threads do their work.

  The class of problems where a solution can be checked in polynomial time.

  One particular chain of computation; a process may contain several of these.

  An $O(n \log n)$ sorting algorithm, where the input data is partitioned into two subsets: the part of the input less than some comparison value, and the part of the input more than it. The two subsets are then sorted, recursively.

  A situation, often connected with a hardware event, where the CPU will stop executing a user program (entirely without warning), and cause the kernel to run instead.

  An automaton which has no memory (other than its states), and makes a single pass through the input; at each symbol it may duplicate itself into multiple parallel computations. The automaton accepts the input if any one of the computations is in an accept state when the input is exhausted.

  In an automaton, a link between states which does not consume any input.

  A sequence of vertices in a graph, where there exists an edge from each vertex to the next in the sequence.

  An algorithm strategy that collects small bits of information about a problem, and gradually links these together to generate a solution.
In an automaton, a state that indicates that the input was in the language.

An $O(n^2)$ sorting algorithm, where the data is scanned to find the maximum value - which is then swapped directly into its final location; this is repeated until all values have been placed in their locations.

In parallel computing, an operation that happens all at once; it is impossible for any other operations to happen “in the middle.”

A critically important open question in computer science: “If a problem can have its answer checked quickly, does that also mean that we can find it quickly? Or are there some such problems which cannot be found quickly?”

A search tree, where each node can hold up to three keys, and the number of children (if not a leaf) is always one more than the number of keys.

A system in modern CPUs, where the addresses used by a program internally are not the same as the physical addresses in memory.

In a programming language, a system which allows you to write an algorithm once, and the language automatically adapts the algorithm to support a wide variety of types.

A problem, which is in NP, which can be used to simulate any other problem in NP.

The piece of software inside the computer which has full permissions to all of the functions of the CPU; it configures virtual memory, accesses hardware, and decides when processes will run.

A way to express a language as a string-matching pattern; has the ability to express various alternative strings, repeating patterns, and concatenation.
2. (8 points)  (a) In a digraph, what is a self-loop?

(b) What does it mean if a graph is “acyclic?”

(c) What is a comparison sort?

(d) What is the worst-case performance of an ‘insert’ operation in a hash table - and in what situation does this happen?

(e) Name two different trees, which we have used in this class, but which are not BSTs.

(f) Use the Master Method to solve the recurrence below.
   **Show your work, or you will not receive any credit.**
   
   \[ T(n) = 4T\left(\frac{n}{2}\right) + n \]
3. (8 points) (a) We were able to traverse a tree using recursion, which is basically DFS. But we didn’t need to have ‘visited’ flags on the nodes, like DFS normally uses. What is two things are special about a tree which makes this possible?
(For partial credit, only mention the property which prevents infinite recursion. For full credit, also mention the property which means we don’t have to worry about traversing any node twice.)

(b) Hash tables are very fast for most operations. Name an operation which is slower in a hash table than in a BST - and explain why they are different.

(c) Suppose that you find the following widget in a red-black tree. Name the rotations which must be performed (in the proper order) to fix the widget. (Don’t worry about the recoloring that will also need to happen; just list the rotations.)

```
(No diagram is provided here.)
```
4. (8 points) Suppose that we have a predicate $K(x, y)$, where $x, y$ are nodes in a digraph and $K(x, y)$ is defined as “There is an edge going from $x$ to $y$.” (Note that self-loops are legal; if a node $A$ has a self loop, then $K(A, A)$ is true.)

Convert the following English expressions to quantifications using $K$:

(a) Every node has at least one outgoing edge - although for some, it might be a self-loop.

(b) Every node has at least one incoming edge - not counting any self-loops.

(c) There are no self-loops in the graph.

(d) A certain node has incoming edges from all of the nodes - including a self-loop.

5. (3 points) The formal definition for $\Theta(g(n))$ is as follows:

$$\Theta(g(n)) = \{ f(n) : \exists c_1 > 0, c_2 > 0 \exists n_0 > 0 \forall n \geq n_0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \}$$

The intuitive explanation for this expression might be said to be:

$\Theta(g(n))$ is the set of functions, where each function can be bounded, both above and below, by scaled versions of $g(n)$. (At least, they are bounded for sufficiently large values of $n$.)

Give a short explanation (no more than a sentence or two) of what each of the following pieces plays in the definition:

(a) $f(n)$

(b) $c_1$ and $c_2$

(c) $n_0$
6. (8 points) Give the .dot file source code for the following graph:

```plaintext
digraph {
  asdf jkl
  A
  C
  G
  B
  E
}
```
7. (8 points) Each of the functions below runs one of the sort algorithms we’ve studied, but it operates on a linked list (instead of an array). For simplicity, we’ll assume that the value types are always int. Make reasonable assumptions about what the methods of the List class do.

Identify the algorithm represented by each function. (You’re not required to explain how you identified them.)

(a) 
List sortA(List input)
{
    List retval = new List();

    while (input.count() > 0)
        retval.addTail(input.removeMin());

    return retval;
}

(b) 
List sortB(List input)
{
    List retval = new List();

    // assume that addInOrder() searches through the list, and inserts
    // the new value at the correct location
    while (input.count() > 0)
        retval.addInOrder(input.removeHead());

    return retval;
}
List sortC(List input)
{
    List a = new List(); // starts empty
    List b = new List();

    boolean tickTock = false;
    while (input.count() > 0)
    {
        int val = input.removeHead();
        if (tickTock == false)
            a.addTail(val);
        else
            b.addTail(val);
        tickTock = !tickTock;
    }

    a = sortC(a);
    b = sortC(b);
    return sortC_part2(a,b);
}

List sortC_part2(List a, List b)
{
    List retval = new List();

    while (a.count() > 0 && b.count() > 0)
    {
        // peek operations look at a value without removing it.
        if (a.peekHead() <= b.peekHead())
            retval.addTail(a.removeHead());
        else
            retval.addTail(b.removeHead());
    }

    while (a.count() > 0)
        retval.addTail(a.removeHead());
    while (b.count() > 0)
        retval.addTail(b.removeHead());
    return retval;
}
(d) List sortD(List input)
{
    if (input.count() <= 1)
        return input;

    int divider = input.removeHead();

    List a = new List();
    List b = new List();

    while (input.count() > 0)
    {
        int val = input.removeHead();

        if (val <= divider)
            a.addTail(val);
        else
            b.addTail(val);
    }

    a = sortD(a);
    b = sortD(b);

    a.addTail(divider);
    a.appendList(b);

    return a;
}
8. (8 points) Assume that we have an unbalanced BST, which uses the following type for the nodes:

```java
public class BSTNode
{
    int key;
    BSTNode left, right;

    public BSTNode(int val)
    {
        this.val = val;
        this.left = this.right = null;
    }
}
```

Give the implementation for `insert(...)`, using the x=change(x) style. **Make sure to write the complete declaration, including the parameters and return type.**

You may assume that the caller never attempts to insert duplicate keys; you don’t have to even consider that case.
Choose only one of the induction problems to do. Do not do both - if you do, we will only grade one of them!

9. (10 points) Prove the following conjecture using structural induction:

“A non-empty 2-3-4 tree with height \( h \) holds at most \( 4^{h+1} - 1 \) keys.”
Choose only one of the induction problems to do. Do not do both - if you do, we will only grade one of them!

Consider the Fibonacci numbers \( F(n) \), with \( F(1) = F(2) = 1 \).

Remember: For all \( k \geq 1 \), \( F(k + 2) = F(k + 1) + F(k) \).

Prove the following conjecture using induction.

For all \( n \geq 3 \), \( F(n) \geq \left( \frac{3}{2} \right)^{n-2} \).
10. (8 points) Draw an NFA below which recognizes the following language:

All strings, over the alphabet \(a,b,c\), which begin and end with \(c\), and which have an odd number of \(b\)'s between them.

**Remember:**

- You need to mark the start state
- Any accept state(s) should be marked with a double-circle
11. (8 points) Run Dijkstra’s Algorithm on the graph below, starting at node A. You are not required to show your work (although it might help you get partial credit). All that you are required to show is the distance to all of the other nodes.
These are the terms to use for the first question.

insertion sort  
bubble sort  
selection sort  
merge sort  
quicksort  
heap sort  
bin sort  
counting sort  
radix sort  
comparison sort  
linear sort  
stable sort  
unstable sort  
pivot  
partitioning  
proposition  
predicate  
free variable  
bound variable  
structural induction  
generics  
heap  
priority queue  
upper bound  
lower bound  
asymptotically tight  
recurrence  

recursion  
AVL Tree  
splay Tree  
B-tree  
2-3-4 tree  
red-black tree  
amortized time  
hash function  
hash table  
collision  
key  
value  
graph  
digraph  
cycle  
spanning tree  
minimum spanning tree  
path  
formal language  
DFA  
NFA  
regular expression  
Turing Machine  
decider  
recognizer  
accept state  
epsilon link  

determinism  
nondeterminism  
exponential time  
P  
NP  
NP complete  
P=NP  
undecidable  
Halting Problem  
brute force  
greedy algorithm  
divide and conquer  
backtracking  
dynamic programming  
hill climbing  
user mode  
kernel  
interrupt  
process  
virtual memory  
page table  
page fault  
copy-on-write  
thread  
race  
atomic operation  
lock