There are 5 questions numbered 0..4 on 8 pages. Check now that you have all the pages. Write your answers in the boxes provided. Use the back of the pages for workspace. Ambiguous answers will be considered incorrect. The exam is closed book and closed notes. Do not begin until instructed. You have 90 minutes. Good luck!

Question 0. (1 points) Write your name and Cornell netid, legibly, on every page.
1. True-false questions (20 points)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>b</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>c</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>d</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>e</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>f</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>g</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>h</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>i</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>j</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>k</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>l</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>m</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>n</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>o</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>p</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>q</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>r</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>s</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>t</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>
2. Short-answer questions (28 points).
(a) (5 points) Write the definition of asymptotic complexity; that is write down when f(n) is O(g(n)) for functions f(n) and g(n).

\[ f(n) = O(g(n)) \] if there exist constants \( c > 0 \) and \( N > 0 \) such that

\[ f(n) \leq c \cdot g(n) \] for all \( n \geq N \)

(b) (5 points) Prove that \( n + 100 \) is \( O(n) \)

Proof we start with \( f(n) \) and show that it is \( < c \cdot g(n) \), finding \( c \) and \( N \) as we go

\[
f(n) = \begin{cases} n + 100 & \text{if we use only } n, n \geq 100 \\ n + n & \end{cases} = 2n
\]

\[ c \cdot g(n) \] SO WE HAVE \( c = 2 \) and \( N = 100 \)

(c) (5 points) The preorder of a tree with 5 nodes is A B C E D. The inorder is B A E C D. Draw the tree:

```
    A
   / \  \
  B   C
 / \   /
E   D  
```

(d) (5 points) Consider hashing using open addressing (not chaining). When removing a value, the array element it occupies is not set to null; instead, a flag is set to indicate that the element is no longer in the set. Explain why.

Linear probing for a value requires looking at the values at index \( h \) (hash address), \( h+1, h+2, ... \), until either null or the value is found. Removing a value by setting the array element to null can mess up later linear probes.

(e) (5 points) List the sequence of nodes visited by BFS (the breadth-first search algorithm) starting at A of the following graph. When a choice of nodes has to be made, choose them in alphabetical order

```
    A
   / \  \
  C   G
 / \   /
H   E  
```

Put the nodes in the order they are visited here: A B C G D F H E

(f) (5 points) For the graph in part (e), state the order of nodes visited by DFS (the recursive depth-first search algorithm, not the iterative one), starting at A. When a choice of nodes has to be made, choose them in alphabetical order.
Put the nodes in the order they are visited here: A B D E H C F G

(g) (5 points) Selectionsort, to sort array b[0..b.length-1], has this invariant:

\[ b[0..k-1] \text{ is sorted and every value of } b[0..k-1] \leq \text{ every value of } b[k..] \]

Write down the body of the loop. We expect it to be at the same level of abstraction as we have described it in lecture. We do not want to see an inner loop.

Swap \( b[k] \) with \( b[j] \), where \( b[j] \) is the smallest value in \( b[k..] \).

\[ k = k+1; \]

(h) (6 points) An array segment \( b[h..k] \) contains weights on edges of a graph. The weights can be negative, 0, or positive. We want a loop with initialization that swaps the values so that all the negative weights are to the left and all the positive weights to the right. Below is the pre- and post-condition for the loop. Below the pre- and post-condition, draw 3 of the 4 possible loop invariants that arise by combining the pre- and post-condition. You do not have to put variables in to mark boundaries. Do not write the initialization or the loop.

The answer is postcondition \( b \) but with a segment marked ? in three different places. Four places are before <0 segment, after <0 segment, after 0 segment, and after >0 segment.

3. (15 points) Create a min-heap of ints (of maximum size eight) by inserting five values 60, 7, 10, 20, -5 in the order shown (leaving room for three more). Draw the resulting heap first as a tree (exactly as seen in class) and then as it is stored in an array (or ArrayList).

(a) 5 points. The heap:

(b) 5 points. The corresponding array (or ArrayList) (put a slash / through “empty” elements):

| -5 | 7  | 10 | 60 | 20 | /  | /  | /  |

(c) 5 points. Suppose poll() is now called once. Put the polled value here ____________ and below, show the array (or ArrayList) after that call on poll().
4. (23 points) **Recursion and asymptotic complexity.** For this question, consider trees whose nodes, of class Node, contain three fields:

- `value`: the value at this node. Its type is some class — not a primitive type
- `left`: left subtree (null if empty left subtree)
- `right`: right subtree (null if empty right subtree)

(a) (6 points) Consider a binary search tree (BST) implemented using class Node. Unfortunately, the person who created a BST used `>` instead of `<` for comparison, so the tree got put in kind of backward. For example, the inorder traversal of one BST should have been `(1, 3, 4, 5, 6, 9)` but was `(9, 6, 5, 4, 3, 1)`. Write method reverse, below, which can be used to rectify the situation. You can use the high-level statement “Swap b and c” to swap the values of two variables b and c. *Hint: values in the nodes should not be changed or compared with each other.*

```java
/**
 * Precondition: t is a binary search tree.
 * If t != null, change as many nodes of t as necessary to reverse
 * the inorder traversal of the tree. */
 public static void reverse(Node t) {
   if (t == null) return;
   Swap t.left and t.right;
   reverse(t.left);
   reverse(t.right);
 }
```

(b) (6 points) Write the body of function equals, given below.

```java
/** Return true iff tree t equals tree s –meaning they have the same shape and same values
 * in their corresponding nodes. */
 public static boolean equals(Node s, Node t) {
   if (s == null || t == null) return s == t;
   return s.value.equals(t.value) && equals(s.left, t.left) && equals(s.right, t.right);
 }
```

(c) (5 points) Suppose we use this function equals to test equality of M balanced BSTs each containing N nodes. That is, we want to test whether all these M balanced BSTs are identical. What is the worst-case complexity of the resulting program, expressed in terms of N and M? Justify your answer by briefly explaining how you arrived at it.

The fact that the trees are balanced has nothing to do with the problem. The answer is \(O(M*N)\). To test two trees — balanced or not — for equality requires in the worst case checking all nodes, and that takes time \(O(N)\). Since we have to do this \(M-1\) times, the time is \(O((M-1)*N)\), which is \(O(M*N)\).
(d) (6 points). Below is a version of depth-first search, explained at a high level of abstraction. The specification is correct — it is what we want — but the algorithm has errors. Fix them. You may explain what is wrong below the algorithm and cross off things in it. Just keep things legible.

```java
/** Node u is unvisited. Visit all nodes that are REACHABLE from u. */

public static void dfs(int u) {
    for all edges (u, v) leaving u:
        if (v is unvisited) {
            visit node v;
            dfs(v);
        }
    visit node u
}
```

1. First statement of body should be visit node u;
2. Delete the statement “visit node v”
3. Delete the last statement “visit node u”.

So it is:

```java
visit node u;
for all edges (u, v) leaving u:
    if (v is unvisited)
        dfs(v);
```

This is as presented in the lecture on DFS.