Instructions:

• Print your name in the space provided below.
• This examination is closed book and closed notes, aside from the permitted one-page formula sheet. No calculators or other computing devices may be used.
• Answer each question in the space provided. If you need to continue an answer onto the back of a page, clearly indicate that and label the continuation with the question number.
• If you want partial credit, justify your answers, even when justification is not explicitly required.
• There are 9 questions, priced as marked. The maximum score is 100.
• When you have completed the test, sign the pledge at the bottom of this page and turn in the test.
• Note that either failing to return this test, or discussing its content with a student who has not taken it is a violation of the Honor Code.

Do not start the test until instructed to do so!

Name ________________________________

Pledge: On my honor, I have neither given nor received unauthorized aid on this examination.

_________________________________________
signed
1. [15 points] Determine whether each of the following statements is true or false. No justification is required.

   a) \( \boxed{F} \) \( 7n \log n + 150n + 1000 \) is \( \Theta(n^2) \)

   b) \( \boxed{T} \) \( 7n \log n + 150n + 1000 \) is \( \Omega(n) \)

   c) \( \boxed{T} \) \( 7n \log n + 150n + 1000 \) is \( O(n^2) \)

Comments:

   a) From the theorems, the function is actually \( \Theta(n \log n) \), which is not \( \Theta(n^2) \).

   b) From the theorems, the function is \( \Omega(n \log n) \). However, since \( \Omega \) is a lower bound, the function would also be \( \Omega \) of any function that is smaller than \( n \log n \) (in \( \Theta \) terms).

   c) Similarly, the function is \( O(n \log n) \), but since \( O \) is an upper bound, the function would also be \( O \) of any function that is larger than \( n \log n \) (in \( \Theta \) terms).

2. [15 points] Assuming that each assignment, arithmetic operation, comparison, and array index costs one unit of time, analyze the complexity of the body of the following function that computes an approximation of the Euler number \( e \), and give a simplified exact count complexity function \( T(N) \) and state its big-\( \Theta \) category:

   \[
   \text{double Euler(unsigned int N) \{} \\
   \hspace{1cm} \text{double Approx, Denom;} \\
   \hspace{1cm} \text{Approx} = 1.0; \hspace{2cm} \text{// 1} \\
   \hspace{1cm} \text{Denom} = 1.0; \hspace{2cm} \text{// 1} \\
   \hspace{1cm} \text{for (unsigned int Iter = 1; Iter <= N; ++Iter) \{} \hspace{1cm} \text{// 1 before, 2 per pass,} \\
   \hspace{3cm} \text{// 1 to exit} \\
   \hspace{4cm} \text{Denom} = \text{Denom} * \text{Iter;} \hspace{2cm} \text{// 2 per pass} \\
   \hspace{4cm} \text{Approx} = \text{Approx} + 1.0 / \text{Denom;} \hspace{2cm} \text{// 3 per pass} \\
   \hspace{1cm} \text{\}} \\
   \hspace{1cm} \text{return Approx;} \hspace{2cm} \text{// 1} \\
   \text{\} \\
   \text{\}}
   \]

   So, the complexity function would be: \( T(N) = 3 + \sum_{\text{Iter}=1}^{N} 7 + 2 = 7N + 5 \)

   Obviously, \( T(N) \) is \( \Theta(N) \).
3. [12 points] Referring to the binary tree shown below, write down the values in the order they would be visited if an enumeration was done using:

a) an inorder traversal: 10 25 40 50 75 80 90 95 (left, then parent, then right)

b) a preorder traversal: 50 25 10 40 75 90 80 95 (parent, then left, then right)

For each of the questions 4 and 5, start with the following BST:

4. [9 points] Draw the resulting BST if 35 is inserted.

5. [9 points] Draw the resulting BST if 50 is deleted.
For question 6, assume the following template declaration for an implementation of a doubly-linked list:

```cpp
// DListT.h
//
// template <typename T> class DListT {
private:
    // private members are not shown
public:
    // some irrelevant public members are not shown
    iterator begin(); // return iterator to first data node (or end())
    iterator end(); // one-past-end
    const_iterator begin() const; // return const_iterator objects similarly
    const_iterator end() const;
};
``` 

6. [10 points] Write an implementation of the following client function that finds the smallest element in a `DListT` object. You may call any public member functions of the template, but you should concentrate on producing an efficient solution. Be careful not to make any assumptions about the contents of the list; in particular, there is no guarantee the list will be non-empty, or that it will not contain duplicate values. You may assume that the data elements can be compared with the usual relational operators, and that the parameter `L` will be non-empty.

```cpp
template <typename T> T Minimum(const DListT<T>& L) {
    DListT<T>::const_iterator Current = ++L.begin();
    DListT<T>::const_iterator minSoFar = L.begin();

    while ( Current != L.end() ) {
        if ( *minSoFar < *Current )
            minSoFar = Current;

        ++Current;
    }

    return ( *minSoFar );
}
```

One solution is shown above. The major issues that came up were:

- `const_iterators` must be used instead of `iterators`, because `L` is passed as `const`.
- There is no way to access the data except via iterators.
For question 7, assume the following template declarations for an implementation of a binary tree:

```
template <typename T> class NodeT {
public:
    T      Element;
    NodeT<T>* Left;
    NodeT<T>* Right;
    // irrelevant members not shown
};
```

```
template <typename T> class BST {
protected:
    NodeT<T>* Root;
    // irrelevant members not shown
public:
    // irrelevant members not shown
};
```

7. [15 points] Consider the proposed new private BST helper member function below.

```
// CheckHelper() traverses the nodes of the tree and determines whether the
// values in the tree satisfy the BST property, returning true or false to
// indicate the result of the check.
//
// template <typename T>
bool BST<T>::CheckHelper(BinNodeT<T>* sRoot) const {
    if ( sRoot == NULL ) return true;   // empty tree is OK
    if ( sRoot->Left != NULL && sRoot->Left->Element >= sRoot->Element )
        return false;
    if ( sRoot->Right != NULL && sRoot->Right->Element <= sRoot->Element )
        return false;
    return ( CheckHelper(sRoot->Left) &&
            CheckHelper(sRoot->Right) );
}
```

The given code does compile. Assume that the BST object is structurally correct; i.e., that there are no dangling or uninitialized pointers. But does it work as advertised? That is, will it return true whenever the tree satisfies the BST property and return false whenever the tree does not? If not, why not? Would there be runtime errors? If so, why? Would it return the wrong result in only some cases? Would it return the wrong result in all cases? If so, give an example. Discuss your analysis below:

If the tree meets the specification, there is no possibility of a runtime error. The design verifies that the pointer sRoot is not NULL before it is ever dereferenced. In the second and third if-statements, the design verifies that each child pointer is not NULL before it is dereferenced. [Some of you need to look up “Boolean short-circuiting”, which works the same way in C++ and in Java.]

If the tree does have the BST property, the function will always return true.

Unfortunately, the function will also return true for some trees that do not have the BST property:

- The basic problem is that you can’t get by with just a local check.

Whether it’s a BST depends on more than just how the children compare to the parent.
8. [10 points] Complete the PR quadtree that would store the points listed below, assuming the world is a coordinate space bounded by the corners (0, 0) and (32, 32).

<table>
<thead>
<tr>
<th>Label</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>(21, 21)</td>
</tr>
<tr>
<td>B</td>
<td>(29, 29)</td>
</tr>
<tr>
<td>C</td>
<td>(9, 15)</td>
</tr>
<tr>
<td>D</td>
<td>(9, 27)</td>
</tr>
<tr>
<td>E</td>
<td>(27, 27)</td>
</tr>
</tbody>
</table>

9. [5 points] We have a theorem that a binary tree of N nodes must have between \( \log(N + 1) \) and \( N \) levels. Is there a similar upper bound for the number of levels that may occur in a PR quadtree that stores \( N \) points? If yes, conjecture what it would be. If no, explain why not. [Hint: consider a PR quadtree that stores points in the xy-plane within the square bounded by the origin and the point (1.0, 1.0).]

When a value is inserted into a PR quadtree, the coordinate space must be partitioned, by splitting the subregions, until the new value does not lie in the same subregion as any other value. How many splittings must be performed depends on how close the new value is to any of the existing values in the tree.

If the new value is sufficiently close, then many splittings must be performed, and each splitting adds a new internal node and makes the relevant branch of the PR quadtree longer than it was before.

In principle, there is no upper bound on how many levels a PR quadtree may have, even if it contains only two points.