Analysis of Dictionary-based Data Structures

<table>
<thead>
<tr>
<th></th>
<th>Hash Table</th>
<th>AVL</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUHA Worst Case</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Find</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Insert</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Storage Space</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data Structures in std library:
- std::map
- std::unordered_map

A Secret, Mystery Data Structure:

<table>
<thead>
<tr>
<th>Implementation of</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>insert</strong></td>
</tr>
<tr>
<td>O(n)</td>
</tr>
<tr>
<td>O(1)</td>
</tr>
<tr>
<td>O(lg(n))</td>
</tr>
<tr>
<td>O(lg(n))</td>
</tr>
</tbody>
</table>

Q1: What errors exist in this table? (Fix them!)

Q2: Which algorithm would we use?

A New Tree-like Structure:

```
ADT:
insert
remove
isEmpty
```
Implementing a (min)Heap as an Array

A complete binary tree T is a min-heap if:

- ...
- ...

leftChild(index):

rightChild(index):

parent(index):

Insert:

Heap.cpp (partial)

```cpp
template <class T>
void Heap<T>::_insert(const T & key) {
    // Check to ensure there's space to insert an element
    // ...if not, grow the array
    if ( size_ == capacity_ ) { _growArray(); } 
    // Insert the new element at the end of the array
    item_[++size] = key;
    // Restore the heap property
    _heapifyUp(size);
}
```

```cpp
template <class T>
void Heap<T>::_heapifyUp( _________________ ) {
    if ( index > _________ ) {
        if ( item_[index] < item_[ parent(index) ] ) {
            std::swap( item_[index], item_[ parent(index) ]
        }
    
    // Restore the heap property
    _heapifyUp( ________________ );
    }
}
```

CS 225 – Things To Be Doing:

1. Theory Exam 3 starts next week (Tuesday, April 3rd)
2. MP5 deadline is Monday, April 2nd
3. lab_hash released today; due Sunday, April 1st
4. Daily POTDs are ongoing!