Read all of the following information before starting the exam:

- Show all work, clearly and in order, if you want to get full credit.
- Circle or otherwise indicate your final answers.
- Please keep your written answers brief; be clear and to the point. I may take points off for rambling and for incorrect or irrelevant statements.
- This test has 10 problems on 10 total pages, including this one, the score sheet page, and a scratch page. It is your responsibility to make sure that you have all of the pages!
- Each problem is worth 10 points for a total of 100 points on this test.
- If you are going to write a portion of an answer on the back of a page, clearly indicate you are doing so on the front of that same page.
- Good luck!
Score sheet

<table>
<thead>
<tr>
<th>Problem</th>
<th>Score</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>
1. Please clearly mark the following statements as True or False. Do not write a single letter; spell the words out.
   (a) Inserting the numbers 1, 2, 3, 4, 5 into a binary search tree in that order, gives the binary search tree the largest possible height.
       
       TRUE
   
   (b) We covered an $O(n)$ time comparison based sorting algorithm in class.

       FALSE
   
   (c) AVL trees have the minimum possible height.

       FALSE
   
   (d) The push and pop operations for a linked list based stack take $O(1)$ time.

       TRUE
   
   (e) On a list of 1000 numbers, a sorting algorithm that takes $\Theta(n^2)$ time will always be slower than a sorting algorithm that takes $\Theta(n \log n)$ time.

       FALSE
   
   (f) $\sum_{i=1}^{n} i = O(n^2)$

       TRUE

2. For each pair of functions below circle the true statements.
   
   (a) $f(n) = n^2$ and $g(n) = n$

       Only $f(n) = \Omega(g(n))$ is correct.
   
   (b) $f(n) = n \log n$ and $g(n) = n^{3/2}$

       Only $f(n) = O(g(n))$ is correct.
   
   (c) $f(n) = n^2 - 2000n$ and $g(n) = 10n^2 + 5 \log n + 3$

       All three are correct.
3. Consider the following piece of code, where \( l \) is either a vector or list and \( n \) is an int:

```cpp
for (int i = 0; i < n; i++) {
    // Inserts \( i \) at the end of the list
    l.insert(l.end(), i);
}
for (int i = 0; i < n; i++) {
    // Removes the element at the beginning of the list
    l.erase(l.begin());
}
```

(a) If \( l \) is a vector (which is an array list in C++), in terms of \( n \) what is the runtime for this code?

\( \Theta(n^2) \)

(b) If \( l \) is a list (which is a doubly linked list with a tail pointer in C++), in terms of \( n \) what is the runtime for this code?

\( \Theta(n) \)

4. Circle the nodes that will be visited in the following skiplist when trying to look up the key 42.

It is okay to circle the column of 44’s too.
5. (a) Insert the keys A, B, C, D, E, and F into the hash table below using separate chaining and the given hash values. Assume no resizing takes place.

- \( \text{hash}(A) = 3 \)
- \( \text{hash}(B) = 0 \)
- \( \text{hash}(C) = 5 \)
- \( \text{hash}(D) = 3 \)
- \( \text{hash}(E) = 3 \)
- \( \text{hash}(F) = 2 \)

Rearranging the elements within bucket 3 is fine.

(b) What is the load factor of the table after inserting the values?

\[
\frac{6}{7}
\]

6. (a) If a hash table has an initial capacity of 10 buckets and doubles in size whenever the load factor passes a threshold of 0.75.

What is the load factor of the hash table after inserting 35 elements?

\[
\frac{35}{80} = \frac{7}{16}
\]

(b) What is wrong with choosing a load factor threshold of 1.1 when using linear probing?

A load factor greater than one means there is more than one element per bucket. However for linear probing each bucket can only hold one element. Once the hash table fills up linear probing will loop forever.
7. (a) Write the preorder traversal of the above tree:

UAADTTSRRTCUES

(b) Write the inorder traversal of the above tree:

DATASTRUCTURES

(c) Write the postorder traversal of the above tree:

DTASRTACUTSERU
8. (a) Draw the above AVL tree after inserting the key 18.

(b) Draw the above AVL tree after deleting the key 31 which is at the root node. (Do the deletion in the ORIGINAL tree, not the tree with 18)
9. Describe an algorithm to build a binary search tree of minimum height from a sorted list. So given the list 1, 2, 3, 4, 5, 6, 7 you should return:

```
4
2
1 3 5 7
6
```

Select the median element to be the root of the current tree and recursively build the left and right subtrees with the portions of the list to the left and right of the median element. If we copy the input list into a left and right, this will take \(O(n \log n)\) time. Because we only need to keep track of the start and stop indices for each recursive call, we do not need to do this copying and can implement it in \(O(n)\) time. Or in pseudocode:

```python
def build_tree(L, i, j):
    if j < i:
        return nullptr
    if i == j:
        return new TreeNode(key = L[i])
    mid = (i + j)/2
    n = new TreeNode(key = L[mid])
    n->left = build_tree(L, i, mid-1)
    n->right = build_tree(L, mid+1, j)
    return n
```

Which can be called as `Node* root = build_tree(L, 0, L.size() - 1)`.

An alternate solution is to insert the values in order into an AVL tree. While AVL trees are not guaranteed to have the minimum possible height in general, this particular insertion order does give them the minimum possible height. This takes \(O(n \log n)\) time.
10. Describe an algorithm to find all of the duplicate elements in an unsorted list. For example given the list 5, 2, 3, 2, 4, 4 you should return 2 and 4. Hint: use a different data structure.

Full credit will be awarded for an $O(n)$ time algorithm, $O(n \log n)$ time will receive partial credit.

Iterate through the list, inserting each element into a hashmap. If the element is already in the hashmap output it.

Or in pseudocode:

```python
def find_dupes(L):
    unordered_set<int> seen
    unordered_set<int> is_dupe
    for x in L:
        if seen.find(x):
            is_dupe.insert(x)
        seen.insert(x)
    return is_dupe
```
Scratch Paper