1. In the worst case, 25000 comparisons (if the name is not present in the list, or happens to be the last name in the list).
   
   In the best case, one comparison (if the name happens to be the first name in the list).
   
   If the name is present in the list, on average it takes 12500 comparisons to find it.

2. 1 comparison: 55
   
   2 comparisons: 27 81
   
   3 comparisons: 3 39 70 93
   
   4 comparisons: 14 31 42 74 85 98

3. No, because it is difficult to find a node in the middle of a linked list. While in an array the indices of the slots can be used to determine a slot in the middle of the array.

4. (a) Yes, this is an AVL tree.
   
   (b) No, this isn’t an AVL tree. Both node 4 and node 6 violate the required property that the depths of the two subtrees differ at most one.
   
   (c) No, this is isn’t an AVL tree, because this isn’t a binary search tree. Node 2 is a child at the right of node 3, and node 7 is a child at the left of node 6.

5. For convenience, numbers are omitted from the AVL trees.

(a)
6. (a) The first sequence leads to the following trees:

```
1 1 1 2 2
\ \ / / / /
2 2 1 3 1 3
\ \ \ 4
3
```

```
2 2 2 4
/ / / / /
1 3 1 4 1 4 2 5
\ / / / / /
4 3 5 3 5 1 3 6
\ \ \ \ 
5 6
```

The second sequence leads to the following trees:

```
1 1 1 4 4
\ \ / / / /
5 5 1 5 1 5
\ / / / / /
4 3 5 3 5 1 3 6
```

```
4 4
/ / / / /
1 5 2 5
\ / / / /
3 1 3
```

(b) The AVL tree constructed after removing the root of the first AVL tree is
The AVL tree constructed after removing the root of the second AVL tree is

```
      5
     / \
    2   6
   / \
  1   3
```

7. Starting at the root, successively jump to the left child in the tree, until a node is reached that doesn’t have a left child. Return the number in that node.

An AVL tree of \( n \) nodes has a depth in \( O(\log n) \). Therefore this algorithm has a worst-case time complexity in \( O(\log n) \).

8. False. For example, in the following AVL tree, the largest number is stored in a node that is two levels above the lowest level in the tree.

```
        5
       / \  
      2   6
     / \  / \\
    1   3 4  5
```

9. Since adding a node to subtree \( B \) caused an inbalance, clearly the depth of subtree \( A \) is at most \( h \). Since after adding a node to subtree \( B \), the subtree in the lower blue node is supposed to be balanced, the depth of subtree \( A \) is exactly \( h \).

Since before adding a node to subtree \( B \) this was an AVL tree, clearly the depth of subtree \( C \) is at least \( h \). Since adding a node to subtree \( B \) caused an inbalance at the higher blue node, the depth of subtree \( C \) is exactly \( h \).