Joshua Nelson
CSCE 146

List Search Lab Report

Problem

Given a sorted list, create a linear search and binary search algorithm to find an arbitrary value. The value may or may not be in the list. Compare the time taken over 20 tries to find a value.

Proposed Solution

Linear search: for (int I : list) { if (I == wanted_value) return I;

Binary search:

Start in the middle of the list. If the current value is less than the wanted, the current is the new minimum; if it is more, it is the new maximum; otherwise, it is equal – return it.

Tests and Results

Note that warnings of a missing element come *before* the corresponding number of checks.

```
% java SearchStats
Test 1. Binary checks: 10, linear checks: 269
Test 2. Binary checks: 10, linear checks: 869
Test 3. Binary checks: 8, linear checks: 237
Test 4. Binary checks: 8, linear checks: 471
Test 5. Binary checks: 7, linear checks: 991
Test 6. Binary checks: 10, linear checks: 900
Test 7. Binary checks: 10, linear checks: 771
Test 8. Binary checks: 10, linear checks: 630
Test 9. Binary checks: 10, linear checks: 544
Test 10. Binary checks: 10, linear checks: 198
Test 11. Binary checks: 9, linear checks: 48
Test 12. Binary checks: 9, linear checks: 223
Test 13. Binary checks: 8, linear checks: 682
Test 14. Binary checks: 10, linear checks: 16
Test 15. Binary checks: 8, linear checks: 424
Test 16. Binary checks: 10, linear checks: 888
Test 17. Binary checks: 8, linear checks: 299
Test 18. Binary checks: 10, linear checks: 996
```

Joshua Nelson
CSCE 146

1146 not found.
Test 19. Binary checks: 10, linear checks: 1000
Test 20. Binary checks: 10, linear checks: 261

Problems Encountered

For this problem, I had to come up with a way to generate numbers that were usually, but not always in the list. The naive solution, Random.nextInt(), would have had a probability of 1 in 2^31 – 1001 = 2147482647 that the integer was in the list. Instead, I generated a number between 0 and 1200; if it was less than 1000, I searched for the number at that index; if not, I searched for the generated number.

I also had difficulty ensuring the binary search would never take more than $\log_2(n)$ time. I consistently used two more searches than necessary. This turned out to be because my break condition was `current == previous` when it should have been `current == min`; more simply put, I made the lowest index found min when it should have been lowest + 1.

Conclusions and Discussion

I used an iterative method for both linear and binary search. It would have been fairly simple to implement either way, but since the recursive version was given in class, I decided to try iteration. I found this lab extremely easy to complete.

Additional Questions

1. What is the Big Oh complexity for linear search? Binary search?
Linear search: O(n)

Binary search: $O(\log_2(n))$

2. Plugging in the size of the data in this test into those complexities, did your tests validate those assumptions? Why?
Yes. In the worst case (item not in list), the linear search would take time equal to the length of the list, while binary search would take time equal to $\text{floor}(\log_2(\text{length}))$.