Assignment 2 - LaTeX Write Up

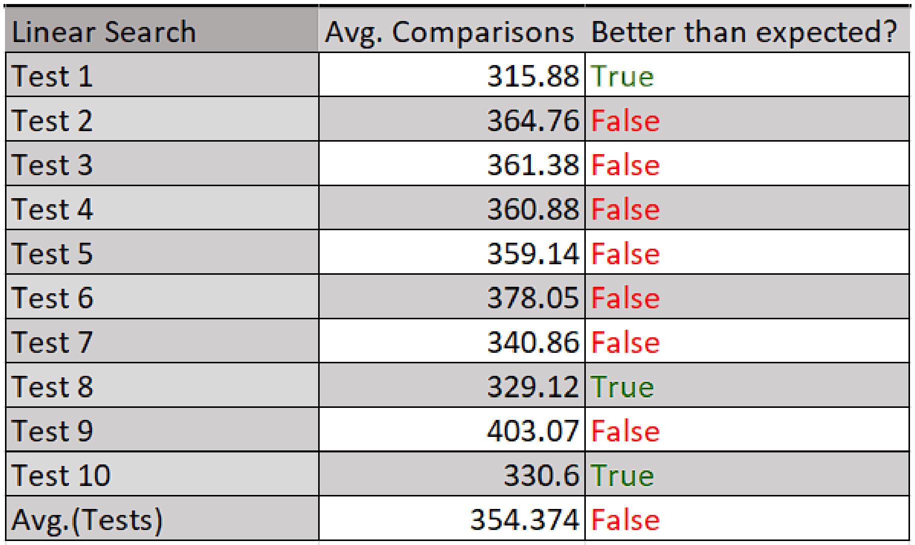
Connor Fleischman

November 1, 2024

1 Assignment Results

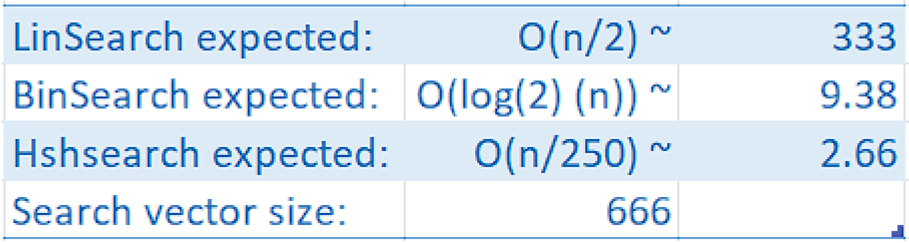
Assignment 02 was to perform different searches on the magic items text file provided in Assignment 01 for 42 random keys, picked from magic items.

The first search to be performed was a linear search. This searching method takes a key and sequentially picks through each magic item, comparing the current item to the key, until the key is found. After implementing this in C++ and ensuring that my algorithm is correct, 10 tests were performed and their data recorded below.



Linear Search	Avg. Comparisons	Better than expected?
Test 1	315.88	True
Test 2	364.76	False
Test 3	361.38	False
Test 4	360.88	False
Test 5	359.14	False
Test 6	378.05	False
Test 7	340.86	False
Test 8	329.12	True
Test 9	403.07	False
Test 10	330.6	True
Avg.(Tests)	354.374	False

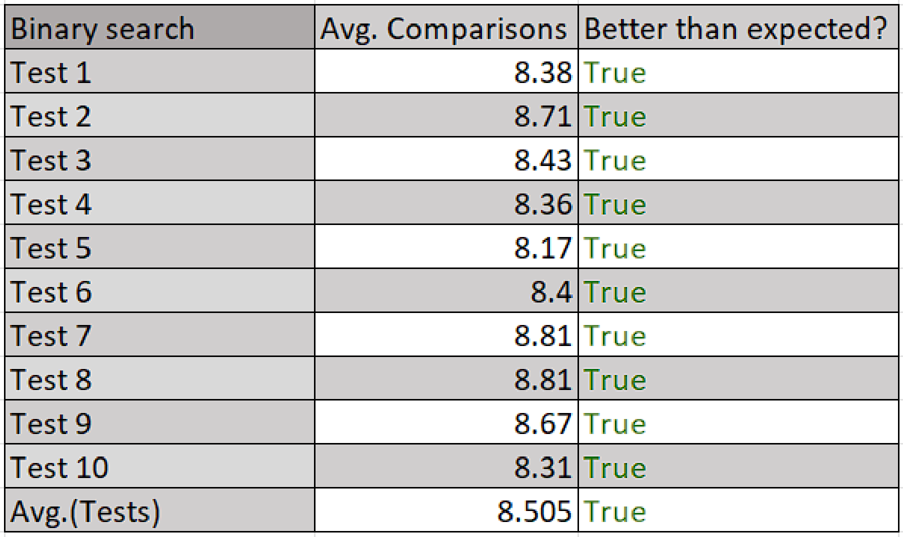
This table diagrams the average time each test took to find all 42 keys in magic items. It also depicts how efficient the algorithm was. Before we continue, it is important clarify a few prerequisites.



LinSearch expected:	O(n/2) ~	333
BinSearch expected:	O(log(2) (n)) ~	9.38
Hshsearch expected:	O(n/250) ~	2.66
Search vector size:	666	

With these in mind, the ”Better than expected?” column now has a basis to compare to. So over the performed tests, the data shows that only 30% of all searches for the 42 keys performed were more efficient than the expected value. Not an amazing look for my code, however only 10 tests were performed, who’s to say that if 100 tests took place it wouldn’t spread to 50%?

A binary search was the next task to conquer. This search takes a key and the middle value of the sorted magic items. Compares if the middle item is or is not the key, if it is not, then it compares if the middle item is larger or smaller than the key. Finally if the key is larger than the middle item, the search is ran again on the half greater than the middle. Otherwise the key is smaller and the search is ran on the half smaller than the key.



Binary search	Avg. Comparisons	Better than expected?
Test 1	8.38	True
Test 2	8.71	True
Test 3	8.43	True
Test 4	8.36	True
Test 5	8.17	True
Test 6	8.4	True
Test 7	8.81	True
Test 8	8.81	True
Test 9	8.67	True
Test 10	8.31	True
Avg.(Tests)	8.505	True

The same 10 tests were performed for binary search with each test’s keys being the same for each numbered test. Meaning, the keys used for test one in linear search were the same keys used for binary search (and for hash searching). As shown above, the binary searching algorithm I implemented was quite efficient. But the same logic applied to linear search applies here, although 100% of tests were more efficient than the average case, does not mean that with more testing it can't be inefficient.

The last search assigned was to take the magic items, create a hash table of 250 buckets, populate it with the items, and search for the keys using the hash table.

2 Prerequisites

From Assignment 01 I was able to use my own code to load magic items into a vector and sort it, which saved me time.

3 Insertion Sort

The second sorting method was Insertion sort which takes the list and compares each element one by one. Meaning the first element is compared to the second, if its greater than the second, swap, then go to the third element, if its less than the second swap, if its less than the first, swap again. This continues until the array is sorted

```
vector<string> insertionSort() // Defines the insertionSort method
{
   int n = lines.size();       // n = the length of lines vector
   for (int i = 1; i < n; i++) // For every item i in the vector other than the first
   {
      int j = i;                      // Let j = i
      while (lines[j - 1] > lines[j]) // While the previous element is larger than the current
      {
         swap(lines[j], lines[j - 1]); // Swap the positions of the current and previous elements
         j--;                          // Decrement j
      }
   }
```

```
    return lines; // Returns the sorted array
}
```

Record your results in a table in a LaTeX document along with your code listings and documentation. Note the asymptotic running time of each search and explain why it is that way.

Add these results to your LaTeX document, including the asymptotic running time of hashing with chaining and explain why it is that way.

4 Final Thoughts

text