Comparing the Efficiency of Linear Search to Binary Search

Homework #1

By

Aaron Rice

CS 303 Algorithm and Data Structures

January 14, 2014

**1. Problem Specification**The goal of this assignment is to create a utility class that will sort an array using the insertion sort algorithm and to search the array using both linear search and binary search. The program is limited to valid Java integers.

**2. Program Design**The program uses the InsertionSort class as a utility class and a driver program.

The following steps were taken to design the program:

1. Create utility class:
   1. Create the linearSearch method to use a brute force approach at searching an array.
   2. Create the insertionSort method according to the provided pseudocode.
   3. Create the binarySearch method to use a divide-and-conquer approach at searching an array.
2. Create the driver class:
   1. Create a loop that will initialize arrays of size 24 to 225.
   2. Create a .csv file of 1000 integers and parse into an array.
3. Make calls to the utility class from the driver class to assess performance of each algorithm.
4. Create timer methods to quantitatively measure the performance of each search algorithm.

The following methods were defined within the InsertionSort class:

1. linearSearch(int[] array, int key)  
   Method to use linear search that brute force searches **array** for **key**.
2. binarySearch(int[] array, int key, int min, int max)  
   Method that recursively uses binary search that searches **array** for **key** by divide-and-conquer. **min** and **max** are the minimum and maximum indices for **array** respectively.
3. insertionSort(int[] array)  
   Method that sorts **array** by comparing each element with those to the left and inserting the element in its proper position.
4. stopTimer()  
   Method that calculates the run time (in nanoseconds) of each searching algorithm by subtracting a final time from an initial time.

The following methods were defined within the Driver class:

1. main(String[] args)  
   Creates and initializes all arrays used for searching. Generates random keys to be found by the search methods. Uses **java.util.Scanner** and **java.util.File** to import and parse a .csv file. Throws a **FileNotFoundException** when the .csv file cannot be found.
2. startTimer()  
   Method to start a clock to time the search methods.

**3. Testing Plan**

The Driver class was used to test the performance of the search algorithms. The algorithms were timed based on their performance by getting the time a before execution and getting a time after the execution. The final time is subtracted from the initial time to get the total elapsed time.

Keys are generated randomly, and their values are always between 0 and **n** where **n** is the size of the array. The elements for the .csv file are between 0 and 225. Performance is gauged by dividing an iteration’s time by the previous iteration’s time. For example, the time elapsed for 25 is divided by the time elapsed for 24 to gauge the increasing factor. This tells if the algorithm ran in linear, quadratic, exponential, etc. time.

**4. Test Cases**

The following is raw console data:

The key on iteration 4 is 15.

timeInitial: 867589166790177

timeFinal: 867589167432057

Linear search returned key 15 at index 14 in 641880 nanosecond(s).

timeInitial: 867589167520558

timeFinal: 867589167550280

Binary search returned key 15 at index 14 in 29722 nanosecond(s).

The key on iteration 5 is 0.

timeInitial: 867589167650470

timeFinal: 867589167672177

-2147483648

Linear search could not find the key. Time elapsed is 21707 nanosecond(s).

timeInitial: 867589167748321

timeFinal: 867589167777710

-2147483648

Binary search could not find the key. Time elapsed is 29389 nanosecond(s).

The key on iteration 6 is 42.

timeInitial: 867589167938347

timeFinal: 867589167961391

-2147483648

Linear search could not find the key. Time elapsed is 23044 nanosecond(s).

timeInitial: 867589168034195

timeFinal: 867589168086961

-2147483648

Binary search could not find the key. Time elapsed is 52766 nanosecond(s).

The key on iteration 7 is 29.

timeInitial: 867589168245594

timeFinal: 867589168268638

-2147483648

Linear search could not find the key. Time elapsed is 23044 nanosecond(s).

timeInitial: 867589168328083

timeFinal: 867589168451650

-2147483648

Binary search could not find the key. Time elapsed is 123567 nanosecond(s).

The key on iteration 8 is 20.

timeInitial: 867589168585236

timeFinal: 867589168610283

Linear search returned key 20 at index 239 in 25047 nanosecond(s).

timeInitial: 867589168653031

timeFinal: 867589169113234

Binary search returned key 20 at index 19 in 460203 nanosecond(s).

The key on iteration 9 is 88.

timeInitial: 867589169265522

timeFinal: 867589169304595

-2147483648

Linear search could not find the key. Time elapsed is 39073 nanosecond(s).

timeInitial: 867589169367047

timeFinal: 867589170684202

-2147483648

Binary search could not find the key. Time elapsed is 1317155 nanosecond(s).

The key on iteration 10 is 961.

timeInitial: 867589170846843

timeFinal: 867589170879572

Linear search returned key 961 at index 566 in 32729 nanosecond(s).

timeInitial: 867589170928664

timeFinal: 867589172326639

Binary search returned key 961 at index 958 in 1397975 nanosecond(s).

The key on iteration 11 is 1028.

timeInitial: 867589172691997

timeFinal: 867589172750774

Linear search returned key 1028 at index 654 in 58777 nanosecond(s).

timeInitial: 867589172844618

timeFinal: 867589176096096

Binary search returned key 1028 at index 1023 in 3251478 nanosecond(s).

The key on iteration 12 is 627.

timeInitial: 867589176617080

timeFinal: 867589176718939

Linear search returned key 627 at index 2562 in 101859 nanosecond(s).

timeInitial: 867589176790408

timeFinal: 867589179922326

Binary search returned key 627 at index 603 in 3131918 nanosecond(s).

The key on iteration 13 is 4165.

timeInitial: 867589180396890

timeFinal: 867589180444313

Linear search returned key 4165 at index 1544 in 47423 nanosecond(s).

timeInitial: 867589180481383

timeFinal: 867589191364281

Binary search returned key 4165 at index 4119 in 10882898 nanosecond(s).

The key on iteration 14 is 13868.

timeInitial: 867589192198190

timeFinal: 867589192307396

Linear search returned key 13868 at index 5166 in 109206 nanosecond(s).

timeInitial: 867589192357491

timeFinal: 867589235890087

Binary search returned key 13868 at index 13925 in 43532596 nanosecond(s).

The key on iteration 15 is 9702.

timeInitial: 867589237540205

timeFinal: 867589237645738

Linear search returned key 9702 at index 4888 in 105533 nanosecond(s).

timeInitial: 867589237681806

timeFinal: 867589413551474

Binary search returned key 9702 at index 9693 in 175869668 nanosecond(s).

The key on iteration 16 is 51222.

timeInitial: 867589415961027

timeFinal: 867589416666360

Linear search returned key 51222 at index 32638 in 705333 nanosecond(s).

timeInitial: 867589416718125

timeFinal: 867590129276623

Binary search returned key 51222 at index 51282 in 712558498 nanosecond(s).

The key on iteration 17 is 54452.

timeInitial: 867590133905035

timeFinal: 867590136132578

Linear search returned key 54452 at index 104032 in 2227543 nanosecond(s).

timeInitial: 867590136278520

timeFinal: 867592971414983

Binary search returned key 54452 at index 53976 in 2835136463 nanosecond(s).

The key on iteration 18 is 25433.

timeInitial: 867592978612251

timeFinal: 867592978761867

Linear search returned key 25433 at index 181673 in 149616 nanosecond(s).

timeInitial: 867592978857046

timeFinal: 867604721214187

Binary search returned key 25433 at index 25346 in 11742357141 nanosecond(s).

The key on iteration 19 is 324840.

timeInitial: 867604734399102

timeFinal: 867604734494616

Linear search returned key 324840 at index 88636 in 95514 nanosecond(s).

timeInitial: 867604734541371

timeFinal: 867650776191409

Binary search returned key 324840 at index 324941 in 46041650038 nanosecond(s).

The key on iteration 20 is 1030254.

timeInitial: 867650808696834

timeFinal: 867650808970351

Linear search returned key 1030254 at index 327082 in 273517 nanosecond(s).

timeInitial: 867650809096255

timeFinal: 867841510834306

Binary search returned key 1030254 at index 1030299 in 190701738051 nanosecond(s).

The key on iteration 21 is 952635.

timeInitial: 867841559263154

timeFinal: 867841560323491

Linear search returned key 952635 at index 1427827 in 1060337 nanosecond(s).

timeInitial: 867841560378595

timeFinal: 868621008354119

Binary search returned key 952635 at index 952355 in 779447975524 nanosecond(s).

The key on iteration 22 is 3205343.

timeInitial: 868621109604115

timeFinal: 868621112934074

-2147483648

Linear search could not find the key. Time elapsed is 3329959 nanosecond(s).

timeInitial: 868621113107068

timeFinal: 872031366609436

-2147483648

Binary search could not find the key. Time elapsed is 3410253502368 nanosecond(s).

The key on iteration 23 is 5679304.

timeInitial: 872031562036394

timeFinal: 872031563435037

Linear search returned key 5679304 at index 1695772 in 1398643 nanosecond(s).

timeInitial: 872031563512183

The following table is raw data taken from the console of the program:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Power | Length | Linear Search Time | Increasing  Factor | Insertion + Binary Time | Increasing Factor | Difference in Milliseconds |
| 4 | 16 | 641880 | - | 29722 | - | -612.2E+03 |
| 5 | 32 | 21707 | 0.033817848 | 29389 | 0.988796178 | 7.7E+03 |
| 6 | 64 | 23044 | 1.061593035 | 52766 | 1.795433666 | 29.7E+03 |
| 7 | 128 | 23044 | 1 | 123567 | 2.341792063 | 100.5E+03 |
| 8 | 256 | 25047 | 1.086920673 | 460203 | 3.7243196 | 435.2E+03 |
| 9 | 512 | 39073 | 1.559987224 | 1317155 | 2.86211737 | 1.3E+06 |
| 10 | 1024 | 32729 | 0.837637243 | 1397975 | 1.061359521 | 1.4E+06 |
| 11 | 2048 | 58777 | 1.795869107 | 3251478 | 2.325848459 | 3.2E+06 |
| 12 | 4096 | 101859 | 1.732973782 | 3131918 | 0.96322903 | 3.0E+06 |
| 13 | 8192 | 47423 | 0.465574961 | 10882898 | 3.474834909 | 10.8E+06 |
| 14 | 16384 | 109206 | 2.302806655 | 43532596 | 4.000092255 | 43.4E+06 |
| 15 | 32768 | 105533 | 0.966366317 | 175863668 | 4.039815774 | 175.8E+06 |
| 16 | 65536 | 705333 | 6.68353027 | 712558498 | 4.051766383 | 711.9E+06 |
| 17 | 131072 | 2227543 | 3.158143742 | 2835136463 | 3.978812225 | 2.8E+09 |
| 18 | 262144 | 149616 | 0.06716638 | 11742357141 | 4.141725555 | 11.7E+09 |
| 19 | 524288 | 95514 | 0.638394289 | 46041650038 | 3.920988732 | 46.0E+09 |
| 20 | 1048576 | 273517 | 2.863632556 | 1.90702E+11 | 4.141939698 | 190.7E+09 |
| 21 | 2097152 | 1060337 | 3.87667677 | 7.79448E+11 | 4.087262043 | 779.4E+09 |
| 22 | 4194304 | 332995 | 0.314046383 | 3.41025E+12 | 4.375216319 | 3.4E+12 |
| 23 | 8388608 | 563226 | 1.691396513 | 1.06618E+13 | 3.126408321 | 10.7E+12 |
| 24 | 16777216 | 952639 | 1.691396513 | 3.33333E+13 | 3.126408321 | 33.3E+12 |
| 25 | 33554432 | 1611291 | 1.691396513 | 1.04213E+14 | 3.126408321 | 104.2E+12 |

The following graph represents the total time elapsed for each iteration:

The following plot compares the increasing performance factors of each algorithm over each iteration:

**5. Analysis and Conclusion**

The early hypothesis was that the binary search would trump the linear search in each iteration even with the time for insertion sort included. The data shows that the original hypothesis is not correct. The first graph shows a quadratic trend for binary and insertion sort. The raw data shows that the only time in which binary search with insertion sort was faster was when the array only had 16 integers. The second graph shows that in terms of increasing factors, the binary search algorithm scaled better only on the powers of 12 and 16.

The table contains three rows of cells which are highlighted blue. These cells are not from concrete test data. At the rate that the algorithms were running, it would have taken a three or more days for the entire program to loop through 225, so the data was extrapolated. The average of all increasing factors was taken, placed into the bottom three cells, and then multiplied by the previous iteration’s runtime. This gave an estimate of how long the final three iterations would have taken given an average run time over each iteration.

In spite of these findings, binary search would be the better search method if the data was already sorted. Linear search’s brute force methodology gives an inherent advantage in that the data never needs to be sorted for the algorithm to work. As such, it’s worst case runtime O(n). Binary search is a combination of the sorting algorithm’s runtime plus an average case runtime of O(n2).

The program only works with integer values, but it could be expanded by using **Integer**’s superclass **Number** to accept any number value. If the program used a set data structure like **ArrayList**, the expansion could be completed using generics and the **Comparable** interface.

**6. References**

The pseudocode was provided in the lab. The sample report was posted via blackboard by Dr. Bangalore.