Comp 2210 Empirical Analysis Assignment

Theo Zinner

09/27/18

**Abstract**

We identified the big-Oh time complexity of five methods using the number 903502834. Timing Data was gathered for each method based on its recorded time and stability. Analysis of this data has determined what sort each method is: 1 is selection sort, 2 is merge sort, 3 is random quicksort, 4 is insertion sort, and 5 is quick sort.

1. Problem Overview

The assignment required us to empirically discover which type of sort was being used in 5 different sort methods in the provided resource jar file. Each of these methods sorts an array with a different sorting algorithm. Each algorithm has a time complexity T(N) that is O(N^k) for some k > 1. Because of this we can calculate the running for some N and 2N. Then we can divide time for 2N by N to get the ration R.

|  |  |  |
| --- | --- | --- |
| **N** | **T(N)** | **R** |
| N0 | T(N0) | - |
| N1 | T(N1) | T(N1)/T(N0) |
| N2 | T(N2) | T(N2)/T(N1) |
| N3 | T(N3) | T(N3)/T(N2) |
| Nm | T(N4) | T(NM)/T(NM-1) |

R will eventually converge to 2^K where K is the time complexity. This will be used to find the time complexity of each sort method given.

Each sorting method will either be stable or not stable. For a sort algorithm to be stable the identical items must remain in the order they first appeared. This will be used to test for stability.

Sort algorithms such as quick sort and insertion sort have varied time complexity based on how the array is arranged before being sorted. This can be used to test for such methods.

2.1 Experimental materials

Two Java classes were used in these tests. First SortingLabClient was used to create arrays. These arrays are sorted and the processing time is recorded. The class TwoNumber had two integer parameters. It created an object that had 2 integer fields. The first field is compared and the second is not. This second field is for testing stability.

2.2 Collecting data

SortingLabClient is used to create arrays and record the time while sorting them, and finally printing the results. It then sorts a second time to check for sorts such as non-randomized quicksort and insertion sort. It finally creates an array of type TwoNumber(example bellow) and sorts its. This is done to check if the method is stable. The first field will be set to 1 or 2 alternating and the second will be from 0 to N. It is then sorted according to the first field. The chances of the sort being unstable yet appearing in the proper order is 1/N!. For an array of size 10 this would be 1/3628800. Assuming N is large enough this is not of concern.

TwoNumber Example

public class twoNumber implements Comparable<twoNumber> {  
 private int first;  
 private int second;  
 twoNumber(int firstIn, int secondIn) {  
 first = firstIn; //compares this only  
 second = secondIn;

SortingLabClient Example

int M = 128001; // max capacity for array  
int N = 1000; // initial size of array  
double start;  
double elapsedTime;  
double eTimeLast = 0;   
System.out.print("sort\n");  
for (; N < M; N \*= 2) {  
 Integer[] ai = getIntegerArray(N, Integer.MAX\_VALUE);  
 start = System.nanoTime(); //time start  
 sli.sort5(ai); //sort happens here  
 elapsedTime = (System.nanoTime() - start) / 1\_000\_000\_000d; //time stop  
 System.out.print(N + "\t"); //print N  
 System.out.printf("%4.3f", elapsedTime); //print time  
 if(eTimeLast == 0) {  
 System.out.print("\tN/A\n"); //Print N/A}  
 else {  
 System.out.printf("\t%4.3f\n", elapsedTime / eTimeLast); //print ratio  
 }  
 eTimeLast = elapsedTime;

2.3 Analyzing running time data

Analysis of the data produced solutions that can be compared to these sort methods.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sort Type | Selection sort | Insertion sort | Merge sort | Quick sort | Randomized quick Sort |
| Best case O | N^2 | N | NlogN | NlogN | NlogN |
| Average case O | N^2 | N^2 | NlogN | NlogN | NlogN |
| Worst case O | N^2 | N^2 | NlogN | N^2 | NlogN\* |
| Stability | Unstable | Stable | Stable | Unstable | Unstable |

\*if the pivot in randomized quicksort happened to be the smallest in the array it is possible that it could be N^2 however this is extremely unlikely.

3 Data Collection Analysis

Data was gather by SortingLabClient. The computer on which it was run is described below:

* Computer: CustomPC 3.4GHz Intel i5(Haswell), 8.00 GB RAM 2133 MHZ
* Operating System: Windows 10 Build 17134
* Java 1.8.0\_151
  + Javac -1.8.0
  + **System**.**nanoTime**() used to measure elapsed time

3.1 Timing Data

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Sort 1** |  |  | **Stability Check** | | **Sort 2** |  |  | **Stability Check** | |
| **N** | **Time** | **Ratio** | **1** | 0 | **N** | **Time** | **Ratio** | 1 | 0 |
| 1000 | 0.008 | N/A | 1 | 2 | 100000 | 0.17 | N/A | 1 | 2 |
| 2000 | 0.039 | 4.777 | 1 | 4 | 200000 | 0.096 | 0.562 | 1 | 4 |
| 4000 | 0.031 | 0.793 | 1 | 6 | 400000 | 0.185 | 1.939 | 1 | 6 |
| 8000 | 0.079 | 2.522 | 1 | 8 | 800000 | 0.412 | 2.223 | 1 | 8 |
| 16000 | 0.312 | 3.967 | 1 | 10 | 1600000 | 0.92 | 2.234 | 1 | 10 |
| 32000 | 1.281 | 4.108 | 1 | 12 | 3200000 | 2.116 | 2.3 | 1 | 12 |
| 64000 | 5.362 | 4.185 | 1 | 14 | 6400000 | 4.733 | 2.237 | 1 | 14 |
| 128000 | 19.189 | 3.579 | 1 | 16 | 12800000 | 10.401 | 2.197 | 1 | 16 |
| **Sort 1 on Sorted Array** | | | 1 | 18 | **Sort 2 on Sorted Array** | | | 1 | 18 |
| **N** | **Time** | **Ratio** | 2 | 5 | **N** | **Time** | **Ratio** | 2 | 1 |
| 1000 | 0.001 | N/A | 2 | 11 | 100000 | 0.018 | N/A | 2 | 3 |
| 2000 | 0.004 | 3.983 | 2 | 3 | 200000 | 0.067 | 3.648 | 2 | 5 |
| 4000 | 0.018 | 4.355 | 2 | 13 | 400000 | 0.167 | 2.493 | 2 | 7 |
| 8000 | 0.074 | 4.003 | 2 | 7 | 800000 | 0.416 | 2.488 | 2 | 9 |
| 16000 | 0.279 | 3.775 | 2 | 15 | 1600000 | 0.959 | 2.304 | 2 | 11 |
| 32000 | 1.304 | 4.671 | 2 | 1 | 3200000 | 2.198 | 2.293 | 2 | 13 |
| 64000 | 5.645 | 4.329 | 2 | 17 | 6400000 | 5.002 | 2.275 | 2 | 15 |
| 128000 | 22.888 | 4.055 | 2 | 19 | 12800000 | 11.267 | 2.252 | 2 | 17 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | |  |  | | |  | |  | |  | |  |  | | |
| **Sort 3** |  | |  | | | **Stability Check** | | | **Sort 4** | |  | |  | | | **Stability Check** | | |
| **N** | **Time** | | **Ratio** | | | 1 | 4 | | **N** | | **Time** | | **Ratio** | | | 1 | 0 | |
| 100000 | 0.062 | | N/A | | | 1 | 2 | | 1000 | | 0.011 | | N/A | | | 1 | 2 | |
| 200000 | 0.056 | | 0.913 | | | 1 | 13 | | 2000 | | 0.02 | | 1.785 | | | 1 | 4 | |
| 400000 | 0.131 | | 2.32 | | | 1 | 0 | | 4000 | | 0.025 | | 1.255 | | | 1 | 6 | |
| 800000 | 0.31 | | 2.374 | | | 1 | 18 | | 8000 | | 0.078 | | 3.15 | | | 1 | 8 | |
| 1600000 | 0.75 | | 2.419 | | | 1 | 6 | | 16000 | | 0.419 | | 5.369 | | | 1 | 10 | |
| 3200000 | 1.958 | | 2.609 | | | 1 | 12 | | 32000 | | 1.426 | | 3.4 | | | 1 | 12 | |
| 6400000 | 5.101 | | 2.605 | | | 1 | 16 | | 64000 | | 8.098 | | 5.678 | | | 1 | 14 | |
| 12800000 | 12.507 | | 2.452 | | | 1 | 10 | | 128000 | | 32.891 | | 4.062 | | | 1 | 16 | |
| **Sort 3 on Sorted Array** | | | | | | 1 | 8 | | **Sort 4 on Sorted Array** | | | | | | | 1 | 18 | |
| **N** | **Time** | | **Ratio** | | | 2 | 5 | | **N** | | **Time** | | **Ratio** | | | 2 | 1 | |
| 100000 | 0.019 | | N/A | | | 2 | 15 | | 100000 | | 0.001 | | N/A | | | 2 | 3 | |
| 200000 | 0.048 | | 2.471 | | | 2 | 7 | | 200000 | | 0.001 | | 2.278 | | | 2 | 5 | |
| 400000 | 0.123 | | 2.551 | | | 2 | 19 | | 400000 | | 0.001 | | 1.677 | | | 2 | 7 | |
| 800000 | 0.309 | | 2.511 | | | 2 | 17 | | 800000 | | 0.002 | | 1.996 | | | 2 | 9 | |
| 1600000 | 0.765 | | 2.48 | | | 2 | 13 | | 1600000 | | 0.005 | | 2.058 | | | 2 | 11 | |
| 3200000 | 1.675 | | 2.188 | | | 2 | 3 | | 3200000 | | 0.010 | | 2.069 | | | 2 | 13 | |
| 6400000 | 3.739 | | 2.233 | | | 2 | 9 | | 6400000 | | 0.021 | | 1.998 | | | 2 | 15 | |
| 12800000 | 8.909 | | 2.383 | | | 2 | 1 | | 12800000 | | 0.042 | | 2.019 | | | 2 | 17 | |
|  |  | |  | | |  |  | |  | |  | |  | | |  |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sort 5** |  |  | **Stability Check** | |
| **N** | **Time** | **Ratio** | **1** | 12 |
| 100000 | 0.034 | N/A | 1 | 14 |
| 200000 | 0.029 | 0.848 | 1 | 10 |
| 400000 | 0.062 | 2.128 | 1 | 8 |
| 800000 | 0.142 | 2.281 | 1 | 18 |
| 1600000 | 0.334 | 2.355 | 1 | 16 |
| 3200000 | 0.800 | 2.399 | 1 | 0 |
| 6400000 | 1.910 | 2.386 | 1 | 2 |
| 12800000 | 4.687 | 2.454 | 1 | 4 |
| **Sort 5 on Sorted array** | | | 1 | 6 |
| **N** | **Time** | **Ratio** | 2 | 17 |
| 1000 | 0.001 | N/A | 2 | 15 |
| 2000 | 0.002 | 3.410 | 2 | 19 |
| 4000 | 0.009 | 4.050 | 2 | 1 |
| 8000 | 0.035 | 3.958 | 2 | 7 |
| 16000 | 0.144 | 4.117 | 2 | 9 |
| 32000 | 0.613 | 4.250 | 2 | 5 |
| 64000 | 2.395 | 3.905 | 2 | 3 |
| 128000 | 10.485 | 4.377 | 2 | 13 |
|  |  |  |  |  |

4 Interpretation

Timing and stability data were collected and compared according to the table in section 2.3.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **---** | **Sort 1** | **Sort 2** | **Sort 3** | **Sort 4** | **Sort 5** |
| **Best O** | N^2 | NlogN | NlogN | N | NlogN |
| **Average O** | N^2 | NlogN | NlogN | N^2 | NlogN |
| **Worst O** | N^2 | NlogN | NlogN | N^2 | N^2 |
| **Stability** | Not Stable | Stable | Not Stable | Stable | Not Stable |
| **Sort Type** | Selection Sort | Merge Sort | Random Quick Sort | Insertion Sort | Quick Sort |