|  |  |
| --- | --- |
|  |  |
| Algorithm Efficiency & Sorting |  |
|  |  |
|  | 11/28/2022Data Structures |
|  | Gage CarpenterGary Thompson |

# Table Of Contents

Abstract…………………………………………………………………………………………………………………………….Page 3

Keywords…………………………………………………………………………………………………………………………..Page 3

Introduction………………………………………………………………………………………………………………………Page 3

Classification……………………………………………………………………………………………………………………..Page 3

Experimental Data……………………………………………………………………………………………………………..Page 4

Comparison-Based Sorting Algorithms………………………………………………………………………………..Page 4

Instrumentation of time complexity…………………………………………………………………………………….Page 4

Quick Sort…………………………………………………………………………………………………………………………Page 4

Merge Sort………………………………………………………………………………………………………………………..Page 4

Heap Sort………………………………………………………………………………………………………………………….Page 5

Bubble Sort……………………………………………………………………………………………………………………….Page 5

Insertion Sort…………………………………………………………………………………………………………………….Page 5

Instrumentation of stability………………………………………………………………………………………………..Page 6

Experiment……………………………………………………………………………………………………………………….Page 7

Results……………………………………………………………………………………………………………………………..Page 7

Conclusion………………………………………………………………………………………………………………………..Page 8

Bibliography……………………………………………………………………………………………………………………..Page 9

### Algorithm Efficiency & Sorting

#### Gage Carpenter

#### Computer Science, West Virginia University - Parkersburg

**Abstract:** This paper delves into the defining characteristics of major sort algorithms and explores their strengths and weaknesses. Although space complexity, parallelism, adaptability, and recursion can be paramount to a result set, The defining characteristics explored in this study were time complexity and stability. The time complexity of an algorithm is expressed through Big-O notation and determined by the number of elementary operations performed on a data set while the stability of algorithms is determined through object identity. After thorough experimentation of various common comparison-based sorting algorithms, the best algorithms for sorted data sets are Insertion and Bubble sort. The best algorithm for unsorted data is Quick sort.

**Keywords:** Quick sort, Merge sort, Heap sort, Bubble sort, Insertion sort, Comparison, Time Complexity, Stability

1. **INTRODUCTION**

A documented series of steps that leads to the transformation of some data could be considered an algorithm. A sorting algorithm is a method for arranging a collection of items in a particular order. Sorting algorithms, classified as comparison-based sorts, are based on the comparison of two elements at a time. The comparison of two elements is done by using a comparison operator or method which can be sorted by either relational or user-defined criteria.

While differing algorithms can all be expressed using Big-O notation, each algorithm attempts to solve sorting problems using different implementations. Analysis of sorting algorithms conveys differences between algorithms and the problems they solve.

The purpose of this paper is to determine the defining characteristics of major sort algorithms and elucidate their strengths and weaknesses. The major sort algorithms used are Quick Sort, Merge Sort, Heap Sort, Bubble Sort, and Insertion Sort. The primary characteristics explored are time complexity and stability.

1. **CLASSIFICATION**

**Time Complexity:** describes the limiting behavior of a function when the argument tends towards a particular value or infinity. Time complexity is conveyed through Big-O notation which is a mathematical display of the performance of a given algorithm. Time complexity is not machine specific due to how time complexity is derived. Time complexity is derived by the measurement of an elementary operation. Elementary operations are those that occur most often while maintaining a constant execution. Elementary operations mustn’t have degrees of variability while performing their calculation (Nilsson). Each examination of the algorithm produces a plot point. After multiple consistent tests of varying magnitude, the collection of elementary operations should be plotted on a cartesian plane. The cartesian plane displays the magnitude of the experimental data tested on the X axis while also displaying the elementary operations performed on that experimental data, on the Y axis. Once the algorithm has been tested extensively with varying magnitudes of experimental data, the best fit curve can be produced through regression analysis, which will yield that algorithm's growth rate function. Using calculus’s approach, function dominance is determined from the best-fit curves growth function which results in the algorithm's big O notation.

**Stability:** algorithms that preserve the relative order of elements with equal keys while allowing the relative sequence of elements that have the same sort key to be maintained are known as stable (Woltmann, 2022). Stable sorting is determined through an operation that yields a binary outcome. The binary outcome can be either true or false. If the binary outcome is true, then the algorithm is stable due to it passing all object identity tests. If the binary outcome is false, then the algorithm is unstable due to it failing one or more object identity tests. Object identity refers to the fact that two objects are the same if they are the same object. In other words, two objects are the same if they are the same instance of the same class. A stable sorting algorithm should result in a lexicographically sorted result set.

1. **EXPERIMENTAL DATA**

To assess the validity and efficacy of the measurements that will be imposed on sorting algorithms, the experimental data sets to be sorted remain constant across all algorithms. All algorithms will be tested on a total of 200 data sets. These 200 data sets consist of randomized alphanumeric characters 10 characters long. The magnitude of these data sets ranges from 500 to 100,000 elements. Each data set is a multiple of 500 elements till 100,000 elements are reached. All 200 data sets will be run 4 times through the given algorithm with varying degrees of lexicographic pre-ordering. The types of pre-ordering imposed are unsorted, partially unsorted, partially sorted, and sorted.

* **Unsorted** - he data set is not sorted.
* **Partially** **Unsorted** - The data set will have every second element out of order.
* **Partially** **Sorted** - The data set will have every fourth element out of order.
* **Sorted** - The data set is sorted in ascending order.

1. **COMPARISON-BASED SORTING ALGORITHMS & INSTRUMENTATION OF TIME COMPLEXITY**

To determine the time complexity of sorting algorithms, the measurement should only be dependent on an algorithm and its input. Time complexity cannot be calculated through clock speeds, but rather the sum of an elementary operation(s) that occur in each algorithm. An elementary operation must have two characteristics. There can't be any other operations that are performed more frequently as the size of experimental data increases. The time to execute an elementary operation must be constant. Each algorithm will be instrumented to increment where the most operations occur. Fewer increments will result in lower time complexity and more increments will result in higher time complexity. Five sorting-based algorithms to be instrumented and analyzed are Quick Sort, Merge Sort, Heap Sort, Bubble Sort, and Insertion Sort

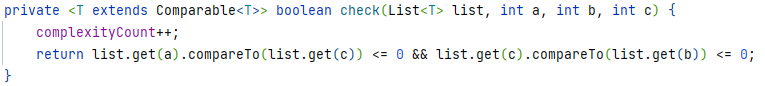
**Quick Sort:** quick sort is a divide-and-conquer algorithm that picks an element as a pivot and partitions the given array around the picked pivot. The array is divided into two parts based on the pivot. The elements less than the pivot are placed before the pivot and the elements greater than the pivot are placed after the pivot. The quick sort algorithm is applied recursively to the two sub-arrays until the sub-arrays are sorted.

A picture containing text, electronics

Description automatically generated

(Prichard & Carrano, 2014)

Instrumentation of quick sort will occur where index a is less than or equal to the element at index c  if the element at index c is less than or equal to the element at index b. The complexity of this portion is O(1) which is an elementary constant and this operation also occurs the most frequently in the quick sort algorithm.



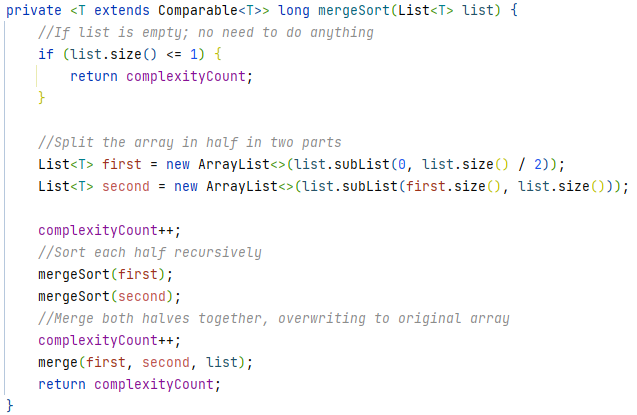
**Merge Sort:** Merge sort is a divide-and-conquer algorithm. The divide step is the first step in the merge sort algorithm. It divides the array into two halves. The divide step is performed recursively until the array is divided into single elements. The conquer step is the second step in the merge sort algorithm. It sorts the array by comparing the elements in the array and merging them into a single sorted array.

Diagram

Description automatically generated

(Prichard & Carrano, 2014)

Instrumentation of merge sort will occur where the algorithm sorts each half recursively and then merges both halves. Other divide-and-conquer algorithms are instrumented in a similar manner.



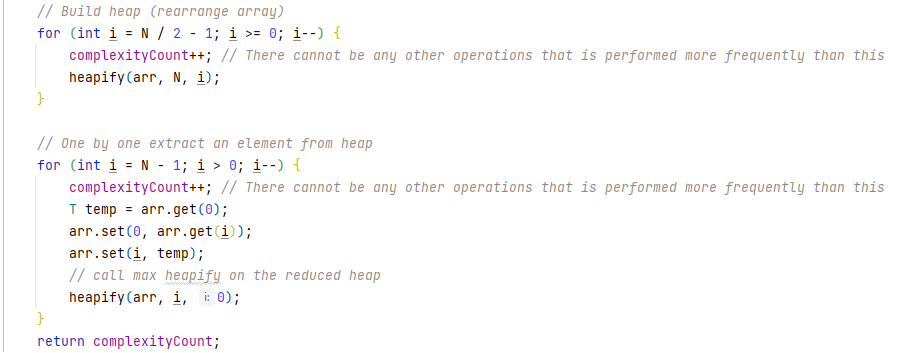
**Heap sort:**  Heap sort is a comparison-based sorting technique based on the Binary Heap data structure. It is like the selection sort where we first find the maximum element and place the maximum element at the end. We repeat the same process for the remaining elements. this sorts the array because it uses a heap data structure to sort the array. The heap data structure is a complete binary tree that satisfies the heap property. The heap property is that the value of each node is greater than or equal to the value of its parent, with the maximum value in the root node.

Diagram

Description automatically generated

(Prichard & Carrano, 2014)

Instrumentation of heap sort will occur where the heap is rearranged and where the heap extracts an element one by one. Building and extracting from the heap occur most often and occur N times.



**Bubble sort:** Bubble sort is a simple sorting algorithm where each pair of adjacent elements are compared, and the elements are swapped if they are not in lexicographic order. This sorts the array because it repeatedly steps through the list to be sorted, compares each pair of adjacent items, and swaps them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted.

A picture containing text, electronics, black, keyboard

Description automatically generated

(Prichard & Carrano, 2014)

Instrumentation of bubble sort is done inside the conditional that swaps experimental data if necessary. Although the inside of the conditional is not the most commonly occurring operation, the conditional itself is. This is the operation where experimental data is sorted. This means that a fully sorted list should only require one pass through the experimental data before completing.

Text

Description automatically generated

**Insertion Sort:**Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed in the correct position in the sorted part. This process sorts the array because it builds the final sorted array one item at a time.

Table

Description automatically generated with low confidence

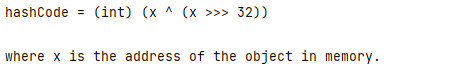
(Prichard & Carrano, 2014)

Instrumentation of insertion sort occurs when one piece of experimental data is being compared to another piece of experimental data.Text

Description automatically generated

1. **INSTRUMENTATION OF STABLITY**

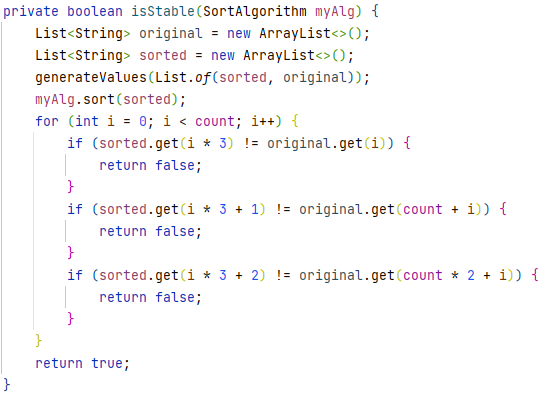
To determine the stability of comparison-based sorting algorithms, alphanumeric characters will be used. Alphanumeric characters all possess what is known as object identity. Object identity is a property of an object that is unique and unchanging. Part of an object’s identity refers to The hash code of an object, which is a unique integer value that is assigned to an object when it is created. The hash code of an object is used to identify the object in a hash table. Hash codes are calculated with the following formula.

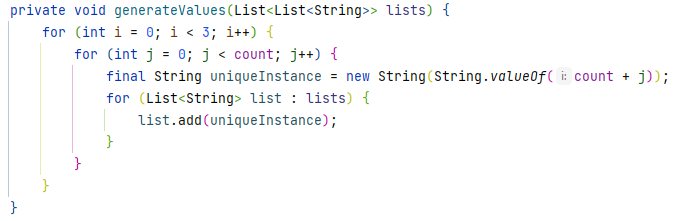


Hash codes are used by the compiler to determine whether two objects are equal.

With the creation of objects that were initialized with duplicate values, only an algorithm that properly handles duplicates can be used to find the correct object. Fortunately, some programming languages offer stable algorithms. These can be used as a baseline. Once the baseline has been established, the baseline can be compared with the algorithm in question and if the two algorithms are not the same, the algorithm in question is not stable. The algorithm in question is not stable if it does not produce the same results as the baseline algorithm.

To perform object identity evaluation, an operation was developed outside of the given algorithm which checks if the algorithm is stable by sorting a list of alphanumeric characters with overlapping values, and then checking if the sorted list is the same as the original list that was sorted under the stable baseline algorithm.



To prevent false positives from occurring, the experimental data used to calculate time complexity is not the same as the experimental data used to measure stability. Experimental data used to measure stability is generated through an algorithm that makes objects in sets of threes, all three objects possess the same value but different object identities. These objects are the ones that are used for stability testing.

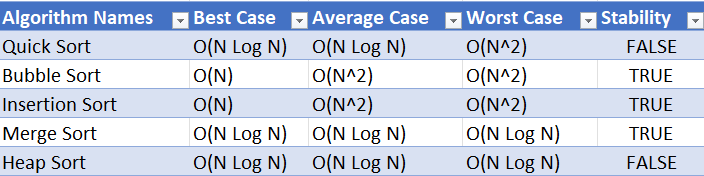
1. EXPERIMENT

At the inception of the experimental program’s driver operations, an input and output directory is planted on the host device. Once created, the operation proceeds to make two hundred JSON files. These JSON files are what will hold the experimental data. All two hundred JSON files have randomized alphanumeric characters serialized to their location if they are empty. JSON is used as a caching mechanism to run additional experiments on a given data set if needed. Once the generation of data has been completed, these experimental seed data JSON files are then deserialized from JSON into collections of alphanumeric characters and partitioned by their native directory. All two hundred collections of deserialized JSON Objects are then run through the four different levels of varying lexicographic order. This results in a scale of eight hundred different collections of data. Of the eight hundred different experimental data sets, each experimental data set is run through all the algorithms. Each sorting algorithm was assigned an output object where that algorithm's name, time complexity, stability, lexicographic order, and experimental data size were all output to the output directory in CSV format.



1. RESULTS

Of the algorithms tested, all showed variations in performance based on either the size of experimental data or the degree of lexicographic order. The Most significant increase in complexity that could be empirically observed was bubble sort from the conversion of full lexicographic order to no degree of order. Bubble sort performs best when the experimental data set is already sorted. Insertion sort also performed best when the data is already sorted. However, Quick sort performed noticeably worse on fully lexicographic experimental data. This could be contrived from the other algorithms' ability to stop after one pass of the experimental data set if nothing needs to be sorted whereas quick sort does not possess this attribute. However, Quick sort performed the best on data sets that had no degree of order.



1. CONCLUSION

After thorough experimentation of various common comparison-based sorting algorithms with multiple consistent tests of varying magnitude, it was proven that Big O notation can be quantitatively measured and compared

 by counting the number of elementary operations and finding their best fit curve.  The best algorithms for sorted data sets are Insertion and Bubble sort. The best algorithm for unsorted data sets is quicksort. Lastly, the best all-around algorithms were merge and heap sort. Using object identity to test stability was also proven to hold merit in determining the stability of comparison-based sorting algorithms. Of all the algorithms tested, Quick Sort and Heap Sort were the only unstable algorithms measured. Combining the values of both stability and time complexity, merge sort was the best algorithm.

#### **Algorithm Efficiency & Sorting Bibliography** Gage Carpenter

#### Computer Science, West Virginia University - Parkersburg

November 11th, 2022

References

[1]“Bubble sort algorithm,” *GeeksforGeeks*, 11-Nov-2022. [Online]. Available: https://www.geeksforgeeks.org/bubble-sort/. [Accessed: 06-Dec-2022].

[2]“Heap sort,” *GeeksforGeeks*, 22-Sep-2022. [Online]. Available: https://www.geeksforgeeks.org/heap-sort/. [Accessed: 06-Dec-2022].

[3]“Insertion sort,” *GeeksforGeeks*, 18-Oct-2022. [Online]. Available: https://www.geeksforgeeks.org/insertion-sort/. [Accessed: 06-Dec-2022].

[4]J. J. Prichard and F. M. Carrano, *Data Abstraction & problem solving with Java Walls and mirrors*. Burnaby, B.C.: University of Simon Fraser Library, 2014.

[5]“Merge sort algorithm,” *GeeksforGeeks*, 15-Nov-2022. [Online]. Available: https://www.geeksforgeeks.org/merge-sort/. [Accessed: 06-Dec-2022].

[6]“Quicksort,” *GeeksforGeeks*, 27-Sep-2022. [Online]. Available: https://www.geeksforgeeks.org/quick-sort/. [Accessed: 06-Dec-2022].

[7]S. Nilsson, “How to analyze time complexity: Count your steps,” *· YourBasic*. [Online]. Available: https://yourbasic.org/algorithms/time-complexity-explained/. [Accessed: 06-Dec-2022].

[8]S. Woltmann, “Sorting algorithms [ultimate guide],” *HappyCoders.eu*, 19-Jul-2022. [Online]. Available: https://www.happycoders.eu/algorithms/sorting-algorithms/. [Accessed: 06-Dec-2022].