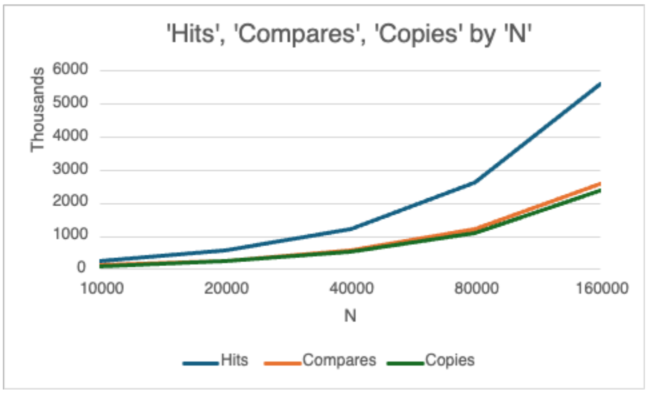
Program Structures and Algorithms
Spring 2024

NAME: Saurabh Srivastava
NUID: 002895225
GITHUB LINK: https://github.com/ssaurabh760/INFO6205

Merge Sort

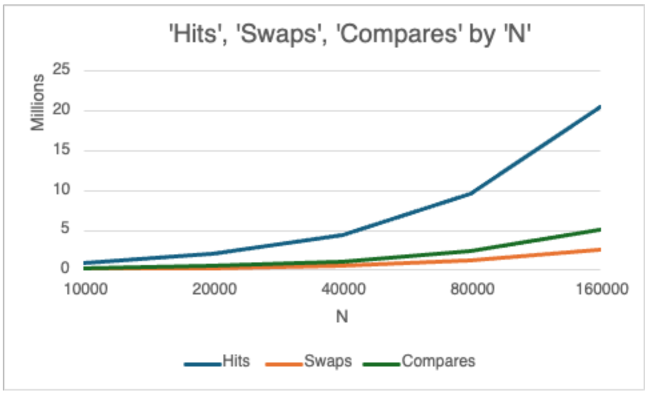
N	Time(instrument true)	Time(instrument false)	Hits	Swaps	Compares	Copies	Normalized Time
10000	3.57	3.2	269838	9792	121489	110000	6.2
20000	8.68	8.25	579407	19472	262980	240000	5.63
40000	17.09	19.42	1239012	39002	566021	520000	5.14
80000	35.06	32.10	2637919	77960	1211947	1120000	4.92
160000	75.82	71.96	5596887	156243	2584092	2400000	4.98



Merge Sort demonstrates consistent and efficient performance across varying dataset sizes, with a normalized execution time per run showing a logarithmic increase as the dataset size grows. It performs a moderate number of comparisons and swaps while exhibiting a relatively high number of array accesses and copies. MergeSort's stable performance and balanced use of resources make it a reliable choice for sorting tasks, particularly for large datasets where its efficient divide-and-conquer approach shines.

Heap Sort

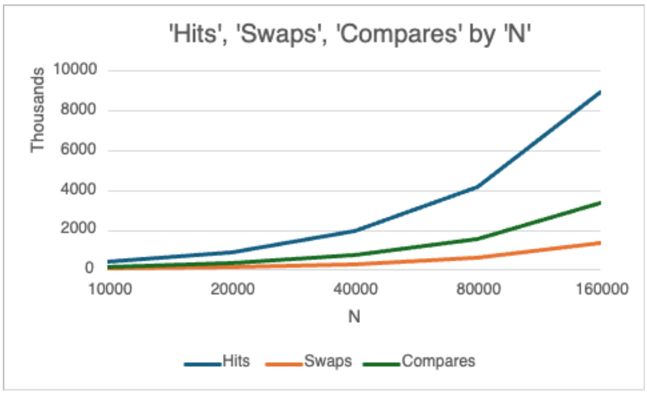
N	Time(instrument true)	Time(instrument false)	Hits	Swaps	Compares	Copies	Normalized Time
10000	375.08	14.07	967615	124213	235381	0	527.72
20000	1685.75	7.22	2095316	268437	510783	0	1093.61
40000	7198.89	16.82	4510298	576824	1101502	0	2166.48
80000	30085.33	31.69	9660362	1233586	2363010	0	4222.11
160000	127714.24	78.08	20600062	2627098	5045835	0	8396.02



HeapSort exhibits higher execution times compared to MergeSort, especially as the dataset size increases, with a normalized execution time per run increasing linearly with the dataset size. Despite performing a substantial number of swaps, HeapSort benefits from minimal array accesses and copies, indicating efficient memory usage. However, its relatively high number of comparisons suggests potential inefficiencies compared to MergeSort, particularly for larger datasets. HeapSort's strength lies in its in-place sorting and consistent performance across different input scenarios, making it suitable for applications where memory usage is a concern.

Quick Sort Dual Pivot

N	Time(instrument true)	Time(instrument false)	Hits	Swaps	Compares	Copies	Normalized Time
10000	278.73	4.57	426214	66906	155872	0	392.17
20000	1246.38	6.81	910696	139482	347221	0	808.57
40000	5175.97	7.97	1986941	309558	737642	0	1577.69
80000	17962.5	17.32	4161422	638036	1587247	0	2520.82
160000	82542.91	44.98	8947694	1377226	3394603	0	5426.43



Quick Sort Dual Pivot showcases competitive performance, with execution times falling between MergeSort and HeapSort. Its normalized execution time per run increases logarithmically with dataset size, similar to MergeSort, indicating efficient scalability.

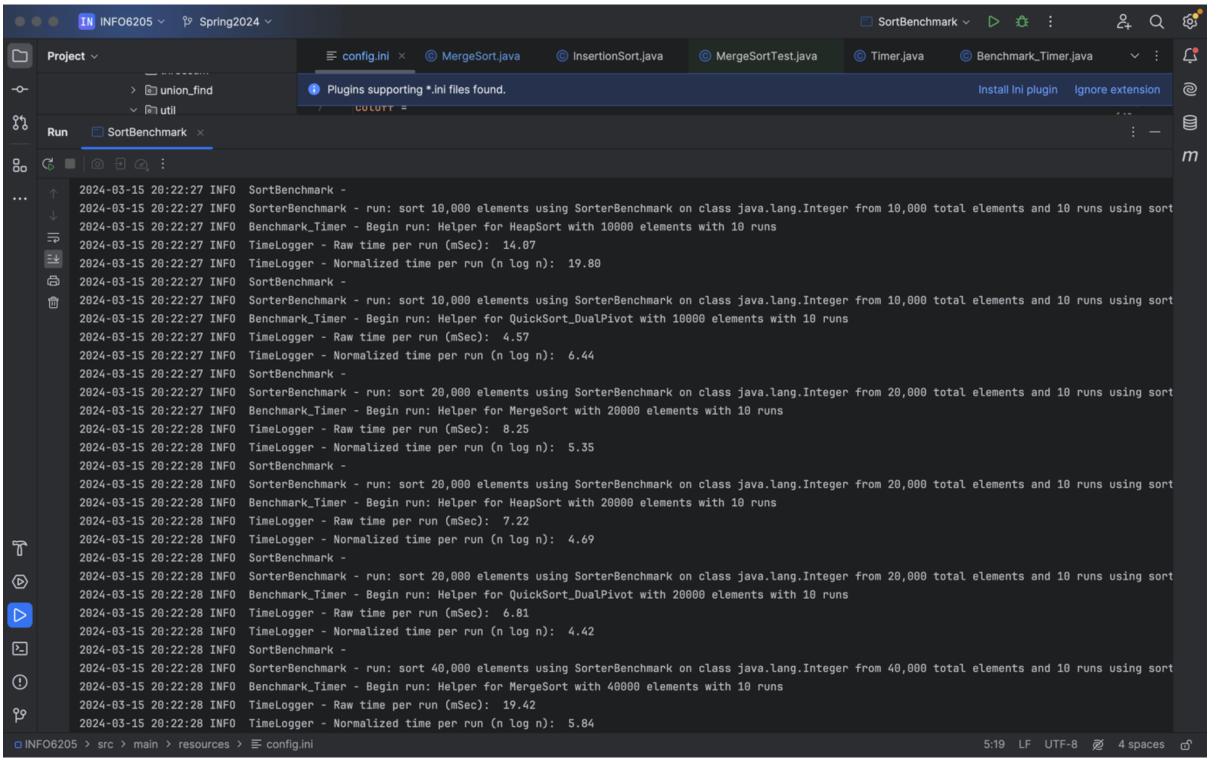
In general, the number of comparisons and the number of swaps tend to be the most indicative factors for predicting total execution time in sorting algorithms. Here's why:

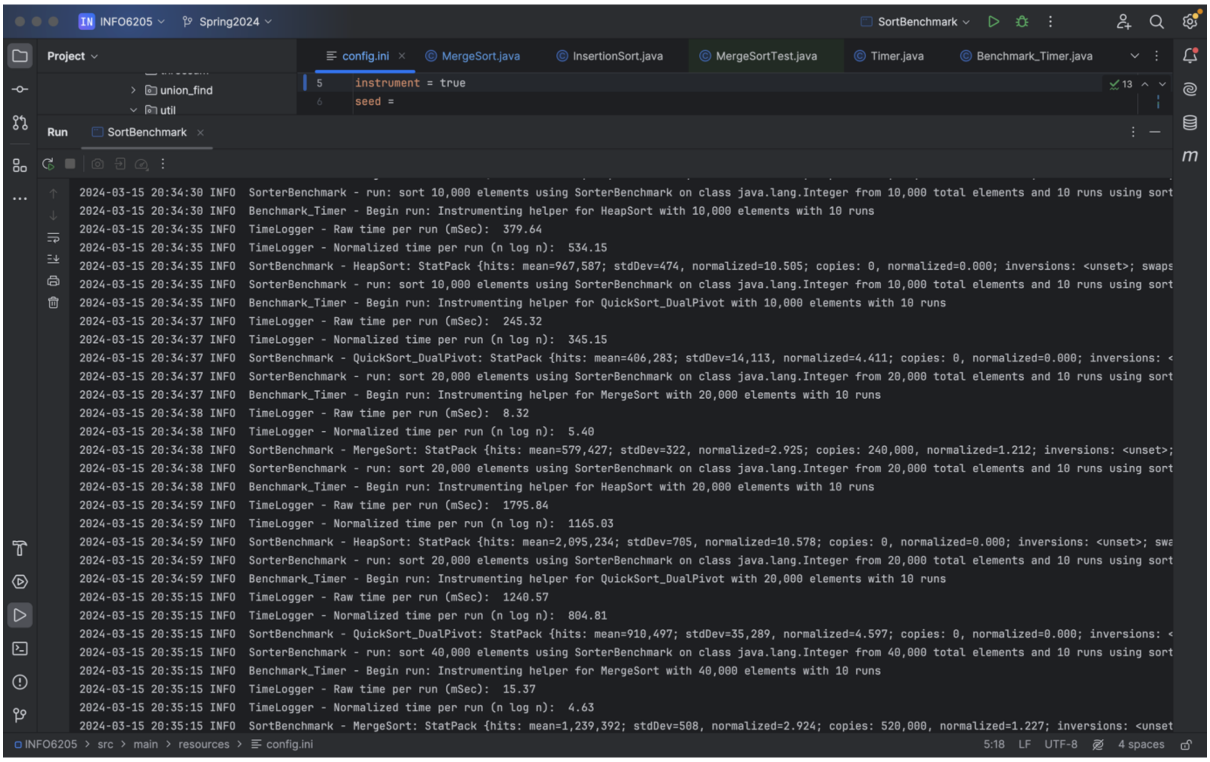
Comparisons: Sorting algorithms primarily operate by comparing elements to establish their relative order. Algorithms with fewer comparisons generally exhibit better performance, especially for large datasets, as each comparison involves evaluating elements and can become a significant overhead.

Swaps: Swapping elements is another fundamental operation in sorting, particularly in algorithms like bubble sort, insertion sort, and quicksort. Swapping becomes more costly as the size of the dataset increases. Algorithms with fewer swaps or more efficient swapping strategies tend to perform better.

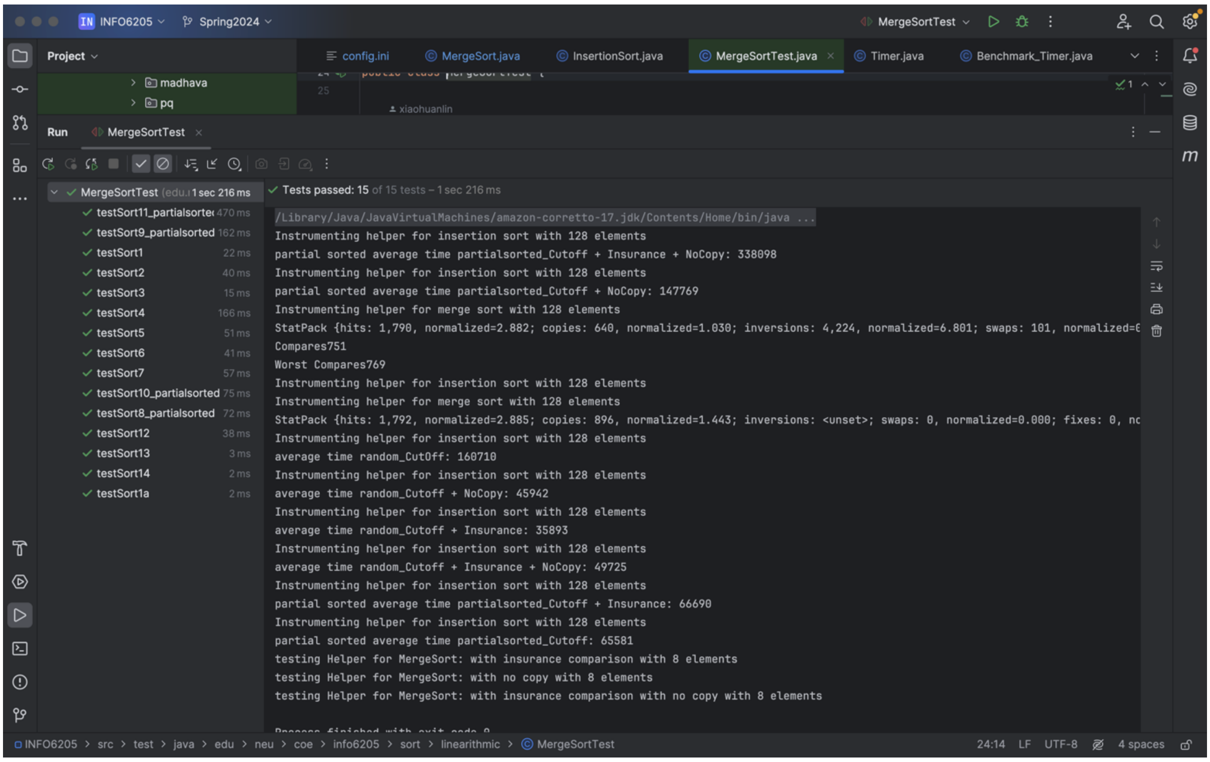
While hits (array accesses), copies, and fixes provide valuable insights into algorithm behaviour, they may not always directly correlate with execution time. Hits and copies can be influenced by factors like data layout in memory and the specific implementation of the algorithm. Fixes are algorithm-dependent and may not be consistently indicative of performance across different datasets.

Therefore, in this case, the best predictor for total execution time is likely to be a combination of comparisons and swaps. By considering both factors, we can assess the overall efficiency and performance of sorting algorithms accurately.





INFO6205
Spring2024
SortBenchmark
Project
config.ini
MergeSort.java
InsertionSort.java
MergeSortTest.java
Timer.java
Benchmark_Timer.java
union_find
util
instrument = true
seed =
Run
SortBenchmark
2024-03-15 20:34:30 INFO SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.Integer from 10,000 total elements and 10 runs using sort
2024-03-15 20:34:30 INFO Benchmark_Timer - Begin run: Instrumenting helper for HeapSort with 10,000 elements with 10 runs
2024-03-15 20:34:35 INFO TimeLogger - Raw time per run (mSec): 379.64
2024-03-15 20:34:35 INFO TimeLogger - Normalized time per run (n log n): 534.15
2024-03-15 20:34:35 INFO SortBenchmark - HeapSort: StatPack {hits: mean=967,587; stdDev=474, normalized=10.505; copies: 0, normalized=0.000; inversions: <unset>; swaps
2024-03-15 20:34:35 INFO SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.Integer from 10,000 total elements and 10 runs using sort
2024-03-15 20:34:35 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort_DualPivot with 10,000 elements with 10 runs
2024-03-15 20:34:37 INFO TimeLogger - Raw time per run (mSec): 245.32
2024-03-15 20:34:37 INFO TimeLogger - Normalized time per run (n log n): 345.15
2024-03-15 20:34:37 INFO SortBenchmark - QuickSort_DualPivot: StatPack {hits: mean=406,283; stdDev=14,113, normalized=4.411; copies: 0, normalized=0.000; inversions: <
2024-03-15 20:34:37 INFO SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.Integer from 20,000 total elements and 10 runs using sort
2024-03-15 20:34:37 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort with 20,000 elements with 10 runs
2024-03-15 20:34:38 INFO TimeLogger - Raw time per run (mSec): 8.32
2024-03-15 20:34:38 INFO TimeLogger - Normalized time per run (n log n): 5.40
2024-03-15 20:34:38 INFO SortBenchmark - MergeSort: StatPack {hits: mean=579,427; stdDev=322, normalized=2.925; copies: 240,000, normalized=1.212; inversions: <unset>;
2024-03-15 20:34:38 INFO SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.Integer from 20,000 total elements and 10 runs using sort
2024-03-15 20:34:38 INFO Benchmark_Timer - Begin run: Instrumenting helper for HeapSort with 20,000 elements with 10 runs
2024-03-15 20:34:59 INFO TimeLogger - Raw time per run (mSec): 1795.84
2024-03-15 20:34:59 INFO TimeLogger - Normalized time per run (n log n): 1165.03
2024-03-15 20:34:59 INFO SortBenchmark - HeapSort: StatPack {hits: mean=2,095,234; stdDev=705, normalized=10.578; copies: 0, normalized=0.000; inversions: <unset>; swa
2024-03-15 20:34:59 INFO SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.Integer from 20,000 total elements and 10 runs using sort
2024-03-15 20:34:59 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort_DualPivot with 20,000 elements with 10 runs
2024-03-15 20:35:15 INFO TimeLogger - Raw time per run (mSec): 1240.57
2024-03-15 20:35:15 INFO TimeLogger - Normalized time per run (n log n): 804.81
2024-03-15 20:35:15 INFO SortBenchmark - QuickSort_DualPivot: StatPack {hits: mean=910,497; stdDev=35,289, normalized=4.597; copies: 0, normalized=0.000; inversions: <
2024-03-15 20:35:15 INFO SorterBenchmark - run: sort 40,000 elements using SorterBenchmark on class java.lang.Integer from 40,000 total elements and 10 runs using sort
2024-03-15 20:35:15 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort with 40,000 elements with 10 runs
2024-03-15 20:35:15 INFO TimeLogger - Raw time per run (mSec): 15.37
2024-03-15 20:35:15 INFO TimeLogger - Normalized time per run (n log n): 4.63
2024-03-15 20:35:15 INFO SortBenchmark - MergeSort: StatPack {hits: mean=1,239,392; stdDev=508, normalized=2.924; copies: 520,000, normalized=1.227; inversions: <unset
INFO6205 > src > main > resources > config.ini
5:18 LF UTF-8 4 spaces



INFO6205
Spring2024
MergeSortTest
Project
config.ini
MergeSort.java
InsertionSort.java
MergeSortTest.java
Timer.java
Benchmark_Timer.java
madhava
pq
xiaohuanlin
Run
MergeSortTest
MergeSortTest (edu.i 1 sec 216 ms
testSort11_partialsorte 470 ms
testSort9_partialsorted 162 ms
testSort1 22 ms
testSort2 40 ms
testSort3 15 ms
testSort4 166 ms
testSort5 51 ms
testSort6 41 ms
testSort7 57 ms
testSort10_partialsorted 75 ms
testSort8_partialsorted 72 ms
testSort12 38 ms
testSort13 3 ms
testSort14 2 ms
testSort1a 2 ms
Tests passed: 15 of 15 tests – 1 sec 216 ms
/Library/Java/JavaVirtualMachines/amazon-corretto-17.jdk/Contents/Home/bin/java ...
Instrumenting helper for insertion sort with 128 elements
partial sorted average time partialsorted_Cutoff + Insurance + NoCopy: 338098
Instrumenting helper for insertion sort with 128 elements
partial sorted average time partialsorted_Cutoff + NoCopy: 147769
Instrumenting helper for merge sort with 128 elements
StatPack {hits: 1,790, normalized=2.882; copies: 640, normalized=1.030; inversions: 4,224, normalized=6.801; swaps: 101, normalized=6
Compares751
Worst Compares769
Instrumenting helper for insertion sort with 128 elements
Instrumenting helper for merge sort with 128 elements
StatPack {hits: 1,792, normalized=2.885; copies: 896, normalized=1.443; inversions: <unset>; swaps: 0, normalized=0.000; fixes: 0, nc
Instrumenting helper for insertion sort with 128 elements
average time random_CutOff: 160710
Instrumenting helper for insertion sort with 128 elements
average time random_Cutoff + NoCopy: 45942
Instrumenting helper for insertion sort with 128 elements
average time random_Cutoff + Insurance: 35893
Instrumenting helper for insertion sort with 128 elements
average time random_Cutoff + Insurance + NoCopy: 49725
Instrumenting helper for insertion sort with 128 elements
partial sorted average time partialsorted_Cutoff + Insurance: 66690
Instrumenting helper for insertion sort with 128 elements
partial sorted average time partialsorted_Cutoff: 65581
testing Helper for MergeSort: with insurance comparison with 8 elements
testing Helper for MergeSort: with no copy with 8 elements
testing Helper for MergeSort: with insurance comparison with no copy with 8 elements
INFO6205 > src > test > java > edu > neu > coe > info6205 > sort > linearithmic > MergeSortTest
24:14 LF UTF-8 4 spaces