3. Performance Benchmarking

We measured each algorithm’s runtimes on random array sizes of 220 to 226 for both integers and double-precision floats. Every size-type-algorithm combination was run **5 times** and the **average** elapsed runtime in both nanoseconds(avg\_ns) and miliseconds(avg\_ms). To isolate pure sorting times, we recorded the timestamp before calling runSort(...)and again immediately after it returns. The difference between the two timestamps is the sorting duration itself excluding any time spent generating data, copying arrays, or writing results. **Figure 1** plots these integer results on a log–log scale. The full dataset is available in **Appendix A** (benchmark\_results.c).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Size (n) | QuadHeap (ms) | 3-Way Merge (ms) | QuickSort (ms) | TimSort (ms) |
| 1048576 | 110.836 | 68.608 | 84.513 | 82.263 |
| 2097152 | 251.005 | 160.116 | 145.745 | 170.523 |
| 4194304 | 490.033 | 322.238 | 316.559 | 352.007 |
| 8388608 | 1161.146 | 604.538 | 643.941 | 757.589 |
| 16777216 | 3039.502 | 1465.003 | 1325.303 | 1595.197 |
| 33554432 | 7157.719 | 2892.172 | 2722.626 | 3294.448 |
| 67108864 | 16705.435 | 5852.513 | 5622.725 | 6969.285 |

*Table 1: key timings at each size*

**Figure 1** plots these integer results on a log–log scale.

A graph with colorful lines and numbers

AI-generated content may be incorrect.

Figure 1: integer results on a log–log scale

**Figure 2** shows the analogous curves for double-precision inputs; the shape and ordering are essentially identical, with each algorithm incurring an extra ~10–15% overhead due to floating-point operations.

A graph with numbers and a line

AI-generated content may be incorrect.

Figure 2: double‐precision results

4. Performance Analysis and Discussion

a) **Asymptotic Behavior:** All four algorithms have slopes near 1.05 – 1.15 on the log-log plot which is consistent with their O(nlogn) time complexity.

b) **QuickSort:** Despite theoretically being the worst case ( O(n2)), QuickSort leads overall as random pivot selection yields excellent average performance with it. QuickSort is the fastest for every tested size — for example, at 2²⁰ (1 048 576 elements) it runs in **84.5 ms** versus Timsort’s **82.3 ms**, and at 2²⁶ (67 108 864 elements) it runs in **5.62 s** versus **7.00 s**.

c) **3-Way Merge:** Merge sort’s three-way split reduces recursion depth and offers stable performance: at 224 (~16.8 M Elements), **QuickSort** averaged **1325 ms** while **3-Way Merge** averaged **1465 ms** i.e., **~10.5%** **slower**. Additionally, at 226 (~67 M Elements), **QuickSort** averaged **5623 ms** and **3-Way Merge** averaged **5853 ms**, only **~4.1% slower**.

d) **Timsort**: While actually designed for real-world partially-ordered data, our random tests show **Timsort** trailing **QuickSort** by ~20% at small sizes (220 and 221) and narrows the gap for larger n. For instance, at 220, **QuickSort** takes **84.5 ms** while **Timsort** takes **82.3 ms**, a difference of **20%** in our random-data tests. Additionally, at 221 (larger n), **QuickSort** takes **145.7 ms** while **Timsort** takes **170.5 ms**, roughly **17% slower**.

e) **Quad-heap**: The quad-heap requires more comparisons and swaps per sift-down since it is a four-ary heap (so we’re examining four children instead of two). That extra work results in the slowest performance across all sizes: at 220 it takes about 110.8 ms while at 226 it balloons to 16.7s, making it the slowest of the four sorts.

f) **Integer vs Double-Precision**: Across all four algorithms, sorting double[] arrays was consistently about 10-12% slower than int[] arrays. For example:

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | int[] | double[] | Speed Rate |
| QuickSort | 84.5 ms | 93.5 ms | ~10.7% slower |
| 3-Way Merge | 68.6 ms | 76.1 ms | ~10.9% slower |
| TimSort | 82.3 ms | 91.0 ms | ~10.6% slower |
| QuadHeap | 110.8 ms | 122.6 ms | ~10.7% slower |

5. Benchmarking Limits

Although the specification called for testing up to 230, our hardware (16 GB RAM, quad-core CPU) imposed practical limits: at 227, average runtimes exceeded 37s for the slowest algorithms and memory use became critical. At sizes >= 228, we encountered an OutOfMemoryErrors and incredibly long single-trial durations (>1 min). Below is a sample of the timing output for n = 227, which we interrupted after confirming excessive durations:

QuadHeap int 2^27 → avg 37528533µs (37528.533ms)

ThreeWayMerge int 2^27 → avg 13694843µs (13694.843ms)

QuickSort int 2^27 → avg 11521432µs (11521.432ms)

TimSort int 2^27 → avg 14562069µs (14562.069ms)

^C (interrupted)

As these single-trial times range from **~11s** (**QuickSort**) up to **~37s** (**QuadHeap**) and began to exhaust the available heap memory, we chose to cap our benchmark at 226.