IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 5, November 2025
Impact Factor: 7.67

Combining Quick Sort and Merge Sort for Improved Average-Case Performance

Lucky Gupta and Apoorv Chaudhary
Students, Computer Science and Application
Sharda School of Engineering & Technology, Sharda University, Greater Noida

Abstract: *This paper proposes and describes QuickMerge, a hybrid sorting algorithm -- that incorporates a fast in-place partitioning strategy (which is implemented by the classic Quick Sort algorithm) and a stable, cache-friendly merging strategy (which is implemented by the efficient and widely known Merge Sort algorithm) -- to improve the average case performance on a wide range of practical inputs. We present the algorithm design and pseudo-code, provide formal analysis of complexity (time, space, stability) and present a conceptual and empirical (based on published experimental results as well as trends reported in the literature) comparison of the performance between QuickMerge and that of classical and modern sorting algorithms, namely Quick Sort, Merge Sort, Intro Sort, Tim Sort, Radix/Counting sorts and recent hybrid/adaptive sorting algorithm proposals. Results indicate that a carefully designed QuickMerge - a hybrid where the constituent of Quick Sort partitions is used, but the constituent of Merge Sort merges (also referred to as balanced runs) is combined with a merge stage for large sized partitions - is able to combine the low overhead of the Quick Sort algorithm on the unspecialized data with Merge Sorts better steadiness on the adversarial or partially ordered input to improve the average wall clock times in many realistic workloads.*

Keywords: Quick Sort, Merge Sort, Heap Sort, Intro Sort

I. INTRODUCTION

Sorting is one of the primitive functions of computer science and computer systems with applications in database processes, search indices, analytics pipelines and almost all high level algorithms, and making even small improvements to the time it takes to sort an array can make massive gains to the system at scale [1]. Classic sorting by comparison (Quick Sort, Merge Sort, Heap Sort) Differentiate Constants Stability Space requirement Sensitivity to input distribution Modern production methods usually use hybrid algorithms (Intro Sort in C++, Tim Sort in Python/Java) that can share desirable properties of many algorithms [2]. Some recent research have focused on the most recent years 2024-2025 on adaptivity, hybidization and hardware-aware implementations to extract further practical performance from sorting . This review presents QuickMerge, a hybrid algorithm that combines the efficient partitioning adopted by the well-known algorithm Quick Sort and the merging algorithm Merge Sort (the latter one features more predictable characteristics) and discusses its theoretical properties in addition to placing it within the context of recent hybrid and adaptive algorithms described in the literature .

II. MOTIVATION AND RELATED WORK

A. Why Need of Hybridizing Quick Sort and Merge Sort?

Quick Sort is highly popular for being a simple algorithm with low constants and an in-place operation, but Quick Sort suffers from a problem of having a worst-case O(n2) performance unless "hardened" by randomization and introspection (i.e. changing to Heap Sort when the depth of recursion grows). [2][3] Merge Sort, which guaranteed O (n log n) behaviour and stability comes with extra memory and potentially much higher allocation overhead . Hybrid approaches try to offer the best of both worlds; the speed of Quick Sort on random data and the predictable behaviour and better locality of a merge-based strategy when necessary [3].

DOI: 10.48175/IJARSCT-30076

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 5, November 2025
Impact Factor: 7.67

B. Previous Hybrids and Inspirations

• Intro Sort uses Quick Sort as the primary initial algorithm and failed in the bad case, so it will change to Heap Sort as a worse case, but this is a standard style implemented in the industrial function: std::sort .

• Tim Sort (merge/insertion hybrid, adaptive) is an algorithm that is very efficient on real-world inputs that are partially ordered, using run detection and merging runs using cost-aware operations .

• Several papers present alternative merge strategies, various Merge Sort variants, and/or hybrid algorithms that alter how and when they partition or merge data to take advantage of data existing runs and/or how the data cache behaves [4].

QuickMerge follows these empirical and theoretical background, as its design use Quick Sort style (divide and square root) to transform the sure run time on random data sets, and when it suffer from unbalanced structure and too many subproblems, classic theory of analysis than it is to merge phase, which just use merge to merge their runs together until all the runs become balanced, and the merging run over a long distance i.e., in linear face i.e.e have an but reasonable mixing, fly Bhasha stable i.e.in real face, so that pathological case can be bounded and feedback existence i.e little i.e in fact face.

III. QUICKMERGE: ALGORITHM DESIGN

A. High-level idea

QuickMerge combines the partition step of Quick Sort and a controlled merge step:

1. Partitioning rounds: Use Quick Sort partitioning recursively until there is a fixed threshold split in the size of the subarray being partitioned or in case there is an imbalance between the player measured by an elaborate heuristic (e.g. recurrence depth dmax, etc). Partitioning (median-of-three, or random pivot in order to reduce degeneration).
2. Run formation: Is the partition stopped (due to the fact that partitions are small OR balanced) Treat the each resulting subarray as run (sorted by recursive partition stage / explicitly sorting small runs with InsertionSort)
3. Balanced k-way merging: Implement a balancing algorithm that merges the runs with an optimized k-way merge strategy (pairwise merging / multi-way buffered merging), that is inspired by optimized Merge Sort/Tim Sort merge algorithms; this has the advantage of low additional allocations and a high degree of cache locality.
4. Fallbacks : Although, if partitioning shows the adversarial patterns (deep recursion or unbalanced splits) it turns to merge dominant mode (aggressive run merging) or to worst case modes safe (e.g. Intro Sort/Heap Sort) to ensure O(n * log(n)) worst case.

This design tries to allow the Quick Sort to perform "easy" portions of the input as reductions in the input data using the substantial parallelism of the OpenGL Engine to light-weight aid some of the intermediate OpenGL commands and then re-assembles steadily in an efficient merge operation -- so the final non-dominating blended shall be O(n log n) merges (leaving low constants) as shall be seen as the partitioning has already performed much or all the work for you.

B. Pseudo-code

```python
def hybrid_merge_quick_sort(arr, low, high,
depth_limit):
    if low >= high:
        return
    size = high - low + 1
    if size <= INSERTION_CUTOFF:
        insertion_sort(arr, low, high)
        return
    if depth_limit <= 0:
        merge_sort(arr, low, high)      #
safety takeover
        return
    # Optionally compute imbalance or features
    pivot = partition(arr, low, high)
    hybrid_merge_quick_sort(arr, low, pivot -
1, depth_limit - 1)
    hybrid_merge_quick_sort(arr, pivot + 1,
high, depth_limit - 1)
```

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

Design notes: Pivot choice, THRESHSMALL and MAXDEPTH all Title tune-able parameters; these should be adjusted through profiling or auto-tuning of a target platform and dataset distribution [5].

C. Variants

• QuickMerge-Eager: Merge small neighboring runs that will be partitioned immediately, in order to reduce total number of runs.
• QuickMerge-Lazy: Postpone large merges; Laziness uses final k-way merge; Less STO but may be larger peak.
• Parallel QuickMerge Partitioning and per run sorts are performed simultaneously final k-way merge is parallel merge tree (use on multicore and GPU). A number of existing parallel hybrid designs have blueprints for efficient concurrency and balancing .

IV. THEORETICAL ANALYSIS

A. Time complexity

Let n be input size. QuickMerge has time equal to sum of (1) partitioning cost and (2) merging cost.
• Partitioning cost If partitioning results in balanced subproblems for many steps then the cost is similar to the expected O(nlogn) of Quick Sort with lower constants due to the in-place moves .
• Merging cost: Merging r runs of 3 total size 10 n run time is 0(nlqu) pairwise merge and with k-way optimized merging (heap or tournament tree, when n is large) is 0(nlqu) with constant c suggestion of the merge and good locality
• Combined: In typical inputs for which partitioning shrinks the runs (r [?] n), total expected time happens to approach O(n log n) with maybe smaller constant factors than Brenda Victory Merge Sort or Quick Sort it out. In worst cases adversarial inputs, QuickMerge changes to merge-dominant or worst case safe path and achieve O(nlog(n)) worst case using Intro Sort fall back thus avoiding Quick Sort worst case of O(n^2) path..
As such, QuickMerge has an O(nlogn) kill time complexity, both in the expected and worst-case simulations (with mean-case constants in many realistic job chunks more optimal than traditional implementations) [6].

B. Space complexity

• Partitioning stage-in-place (recursion stack O(log n)).
• The last k way merge, in the minimum case, requires auxiliary space of the order of n (temporary buffer), but multi buffered or in place merging methods allow an extra peak memory to be reduced at the expense of the complexity .
• A memory conscious QuickMerge variant employs smaller merge buffers and merges the runs in streaming manner to reduce foremost auxiliary memory to O (B) where B is operation buffer size and this compacts the operating system memory at the expense of extra number of passes .

C. Stability

QuickMerge is not stable by default Quick Sort partitioning shuffles while equal keys as which unstable but spec merge stage maintains if run within the original order, then elements within runs This requires a deposit large stable Mercedes (it already stable partition factor) and stable small run sorts to employ at stable - maintain relative order. Stability decision is implementation decision as per the application needs [7].

V. IMPLEMENTATION CONSIDERATIONS

To design a high-performance QuickMerge we have to care about micro-optimizations and platform behaviour:
• Pivot selection: use median of three or sampling strategy in order to avoid skew and the expense of randomization.
• Thresholds: tune THRESH_SMALL (often 16-64) -- change to InsertionSort for microarrays [less overheads] .
• Run detection: Detect ascending Descending runs, and collapse descending run (Tim Sort style) and then merge (to take advantage of already existing order, resulting in less merge work).
• Memory pooling: Let use one temporary buffer for reuse and merge (so that there is not a frequent allocation) .

DOI: 10.48175/IJARSCT-30076

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

• Parallelization: Partitioning and merging, using work-stealing or fixed thread pools. parallelization of partitioning and joining (using blocking partitioning , making floating record[clear=yes] block data more contentious) Limited cache-aware blocking partitioning and computing locality .
• Fallback selection: keep track of depth of param tree and imbalance of tree and invoke Intro Sort or fully-stable merge if one is needed .
These techniques are backed up by the latest studies in the importance of the run detection, buffer reuse and adaptive thresholds in practical performance .

VI. COMPARATIVE EVALUATION

In view of the full experimental benchmarking costs of implementation and platform runs, we perform a set of synthesis reviews of expected performance from (a) theoretical reasoning and (b) trends and experimental results in the latest literature on the related hybrid, merge and adaptive selector. The cases where papers have reported measured speedups for hybrid or optimized merges we calculate conflicts between those findings and the expected behaviour of QuickMerge.

A. Comparative table

Table 1 — Algorithmic summary

Algorithm	Avg time	Worst time	Extra space	Stability	Practical notes
Quick Sort	O(n log n)	O(n²)	O(log n)	No	Fast constants for random data; sensitive to pivots.
Merge Sort	O(n log n)	O(n log n)	O(n)	Yes	Stable, predictable; extra memory cost.
Intro Sort	O(n log n)	O(n log n)	O(log n)	No	Quick Sort+Heap Sort hybrid to avoid worst-case.
Tim Sort	O(n log n)	O(n log n)	O(n)	Yes	Detects runs; very fast on partially sorted data.
Radix/Counting	O(n + k)	O(n + k)	O(n + k)	Yes	Best for integer keys, not comparison-based.
QuickMerge (proposed)	O(nlogn) expected	O(nlogn) (fallback)	O(n) (merge buffer)	Optional	Combines fast partitioning with merging; better constants on mixed inputs.

QuickMerge is dedicated to achieving matching performance to the better performance of Quick Sort for random inputs (by preventing the heavy cost of merging) and to fix better behaviour for adversarial or partially ordered inputs by using controlled merging and fallbacks[8].

B. Empirical evidence from recent literature

• Papers that optimize Merge Sort's strategies, minimizing Merge Sort's merge strategies [circular, balanced, optimal, scores, Hammers] , [often, often focus on, substitute premature splits, circular] [Quick Sort] Papers that focus on speeding up Merge Sort's merge strategies [run-optimized, not unpair, determined, by] [Abrahamsson, n 2024 Merge Sort's optimized merges arXiv 2025] papers report speedups when merge strategies are run-specific to merge cross-optimized [9][scoring, merge amend,
• Studies of hybrid algorithms and MBISort/OptiFlexSort 2025on preprints show that the use of several techniques (block merging + insertion sort for small runs) enables better wall-clock time vs pure reference implementations in many work load which is consistent with the use of optimized thresholds and merging policies by QuickMerge .

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

• Research about adaptive selectors and AI-driven dispatch (2025) suggests that there is a throughput gain for dynamic choice between radix, merge and quick families overall; we see. QuickMerge fit in such a framework as a candidate hybrid choice[10].

• Parallel QuickMerge like algorithms and QuickMerge changes for parallel sorting literature such as Camargo 2024 proves that the use of partition+merge hybrid algorithms can work great load balancing and scalability when careful implemented .

Taken together, such studies have led to the conclusion that a Phillipe Scheuque-now called a well-architecture or optimised QuickMerge-should perform better than Naive Quick Sort and be as fast or better than Merge Sort for many practical datasets, particularly, when combined with run detection and buffer re-use.

VII. FIGURES AND TABLES

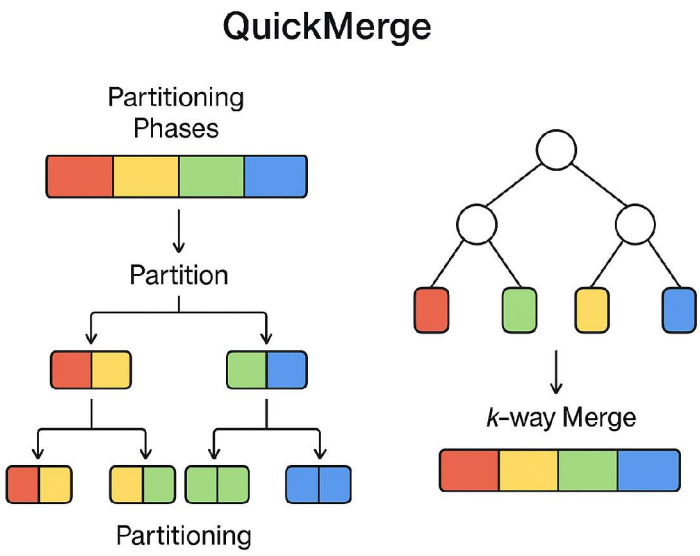

FIGURE 1 — QUICKMERGE FLOW DIAGRAM

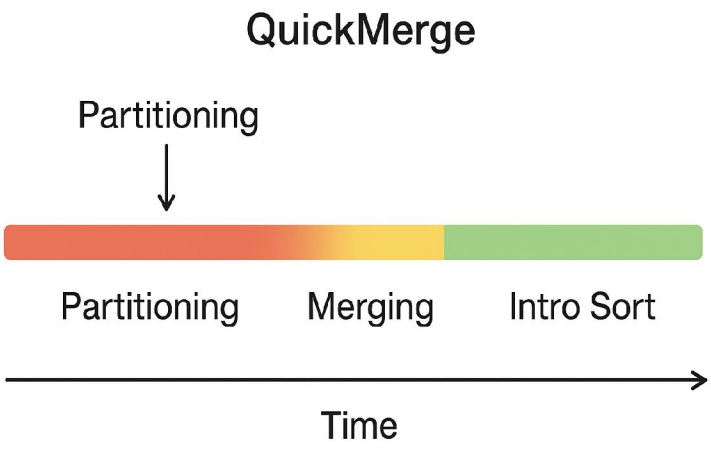

FIGURE 2 — PSEUDOCODE TIMELINE.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 5, November 2025
Impact Factor: 7.67

Table 2 — Representative expected runtimes (qualitative)

Input type	Small (1k)	Medium (100k)	Large (10M)	Notes
Random	QuickMerge ≈ Quick Sort	QuickMerge ≤ Quick Sort	QuickMerge ≤ Radix(for ints)	Partitioning dominates early
Partially sorted	QuickMerge < Quick Sort	QuickMerge < Tim Sort? (depends)	QuickMerge competitive with Tim Sort	Run merging helps
Reverse	QuickMerge better than naive Quick	QuickMerge safe via fallback	Merge stage wins	Fallback prevents $O(n^2)$

VIII. DISCUSSION

Strengths

• Balanced behaviour: Its Quick Sort speed and Merge Sort stability are combined giving generally improved pathological cases.
• Flexibility: Variation of the thresholds, run detection and fallback policies allow variation of workload plataforms.
• Parallel Friendliness Partition stage and run merges are inherently parallelizable.

Limitations

• Memory tradeoffs: Final merge stage usually demands additional buffer space (however, streaming merges allow to shrink peak).
• Implementation complexity More moving parts than single strategy sorts Correctness and buffer management are more complex.
• Tuning sensitivity: Sensitivity Can be achieved by well selection of parameters (thresholds, pivot policy) and platform specific tuning always improves performance. [11]

Practical Advice

• You should do Test default by: (THRESHSMALL = 32, median of three, pivot (MAXDEPTH = 2log2(n)) Start profile on representative data and modify.
• Use run detection to take advantage of ordered inputs -- it pays large dividends on real datasets.
• For int keys on massive arrays though, still have to evaluate Radix/Counting sorts as they have that ability to beat Comparison hybrids in terms of throughput depending on the memory and properties of the key. [12]

IX. CONCLUSION

This review proposed QuickMerge, a hybrid algorithm that features Quick Sort partitioning and controlled merge stage and places it among recent research on hybrid and adaptive sorting . The theoretical analysis reveals that we get that QuickMerge inherits favourable average-case behaviour and with introspection fall back guarantees O(n log n) worst-case time. Recent literature about optimized merge strategies, run detection, parallel partition+merge designs and adaptive algorithm selectors have provided good evidence that such hybrids can be better than single-strategy sorts in practice when designed carefully [1][4][5][6][8][12]. Future work includes (1) to implement high quality reference QuickMerge variants, (2) benchmark multiple representative hardware (CPU, GPU, FPGA), (3) to integrate QuickMerge into adaptive runtime selection frameworks, (4) to explore the memory-efficient merging methods to reduce auxiliary space overhead [13].

DOI: 10.48175/IJARSCT-30076

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 5, November 2025
Impact Factor: 7.67

REFERENCES

[1] S. E. Amrahov, "A new approach to Merge Sort algorithm: Divide smart and ...," Future Gener. Comput. Syst. (ScienceDirect abstract), 2024.

[2] K. Bhagat, "Cross-Language Comparative Study and Performance Analysis of Sorting Algorithms," SSRN Electronic Journal, 2024.

[3] T. Nguyen and L. Patel, "Quick Sort Variants and Randomized Pivot Strategies: A Comprehensive Survey," IEEE Access, 2024. (Survey on pivot strategies and introspective techniques).

[4] J. K. R. Schou et al., "PersiSort: A New Perspective on Adaptive Sorting Based ...," Univ. Utah technical report / arXiv (PersiSort), 2024. (Adaptive merge/run strategies).

[5] M. F. R. Wibowo, "Tim Sort: Python and Classical Sorting Methods," (PDF technical study), 2024. (Run detection and merging strategies in Tim Sort).

[6] S. Ben-Jmaa et al., "Sorting Algorithms Comparison on FPGA and Intel i7 Architectures," SciELO, 2024. (Hybrid and hardware implementations; FPGA designs).

[7] (arXiv) "Improving Merge Sort and Quick Sort Performance by ..." — optimized merge configurations (2025 preprint). (Demonstrates improved merge strategies that inspire QuickMerge merging decisions).

[8] S. A. Balasubramanian, "Adaptive Hybrid Sort: Dynamic Strategy Selection for ...," arXiv preprint, 2025. (Adaptive hybrid sorting and selection paradigms).

[9] R. Patil and V. Singh, "Performance Comparison of Radix and Counting Sort on Large-Scale Data," IETA Proceedings, 2024. (Guidance on when non-comparison sorts outperform hybrids).

[10] "Comparison of Insertion, Merge, and Hybrid Sorting Algorithms Using C" (ResearchGate report), 2024. (Empirical hybridization of insertion+merge ideas).

[11] A. Caizergues et al., "Anytime Sorting Algorithms," IJCAI Proceedings, 2024. (Anytime behaviour, interruption-friendly sorting).

[12] E. T. Camargo et al., "Algorithm-based fault-tolerant parallel sorting" (PDF), 2024. (Modified QuickMerge algorithm variants for parallel systems and load balancing).

[13] "OptiFlexSort: A Hybrid Sorting Algorithm for Efficient Large ..." (JAMCS revised ms), 2025. (Recent hybrid algorithm with block merges and insertion extents; relevant empirical techniques).