Assignment 1

Learning goals

• Implement classic divide-and-conquer algorithms with safe recursion patterns

• Analyze running-time recurrences using Master Theorem (3 cases) and Akra–Bazzi intuition; validate with measurements.

• Collect metrics (time, recursion depth, comparisons/allocations) and communicate results via a short report and clean Git history.

1) Algorithms (required)

1. MergeSort (D&C, Master Case 2)

Linear merge; reusable buffer; small-n cut-off (e.g., insertion sort).

2. QuickSort (robust)

Randomized pivot; recurse on the smaller partition, iterate over the larger one (bounded stack ≈ O(log n) typical).

3. Deterministic Select (Median-of-Medians, O(n))

Group by 5, median of medians as pivot, in-place partition; recurse only into the needed side (and prefer recursing into the smaller side).

4. Closest Pair of Points (2D, O(n log n))

Sort by x, recursive split, “strip” check by y order (classic 7–8 neighbor scan).

2) Report PDF

• Architecture notes (how depth/allocations are controlled).

• Recurrence analysis for each algorithm (2–6 sentences each: method used and Θ-result; Master or Akra–Bazzi intuition).

• Plots: time vs n; depth vs n; short discussion of constant-factor effects (cache, GC).

• Summary: alignment/mismatch between theory and measurements

GitHub workflow (issues, branches, commits)

Branches

• main — only working releases (tag v0.1, v1.0).

• Features: feature/mergesort, feature/quicksort, feature/select, feature/closest, feature/metrics.

Commit storyline

- init: maven, junit5, ci, readme
- feat(metrics): counters, depth tracker, CSV writer
- feat(mergesort): baseline + reuse buffer + cutoff + tests
- feat(quicksort): smaller-first recursion, randomized pivot + tests
- refactor(util): partition, swap, shuffle, guards
- feat(select): deterministic select (MoM5) + tests
- feat(closest): divide-and-conquer implementation + tests
- feat(cli): parse args, run algos, emit CSV
- bench(jmh): harness for select vs sort
- docs(report): master cases & AB intuition, initial plots
- fix: edge cases (duplicates, tiny arrays)
- release: v1.0

Testing

- Sorting: correctness on random and adversarial arrays; verify recursion depth is bounded (QS depth $\lesssim$ ~2*floor(log2 n) + O(1) under randomized pivot).
- Select: compare result with Arrays.sort(a)[k] across 100 random trials.
- Closest Pair: validate against O(n^2) on small n (≤ 2000); use only the fast version on large n.