Comparative Report: Selection Sort vs Binary Insertion Sort

1. Algorithm Overview

Selection Sort:

Repeatedly finds the minimum element from the unsorted portion and moves it to the start.

Divides array into sorted and unsorted segments.

Simple, in-place, predictable, but inefficient for large arrays.

Time complexity: $O(n^2)$ comparisons and swaps.

Binary Insertion Sort:

Uses binary search to find the correct insertion position in the sorted portion.

Reduces comparisons to $O(n \log n)$ but shifting elements remains $O(n^2)$.

Efficient for small or nearly sorted arrays.

2. Complexity Summary

Algorithm	Comparisons	Shifts	Runtime	Space
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Binary Insertion Sort	$O(n \log n)$	$O(n^2)$	$O(n^2)$	$O(1)$

Key point: Binary Insertion Sort reduces comparisons but shifts dominate runtime for large arrays.

3. Code Optimizations

Selection Sort:

Add flag to break early if array is already sorted.

Use local variables to reduce memory access.

Binary Insertion Sort:

Use block shifting (e.g., System.arraycopy) instead of element-by-element shifts.

Skip insertion when elements are already in order.

Separate binarySearch() and insert() functions for clarity.

4. Empirical Observations

Selection Sort: Consistent $O(n^2)$ execution time, predictable performance.

Binary Insertion Sort:

Comparisons grow roughly as $O(n \log n)$.

Shifts grow quadratically, dominating runtime for large n.

Faster than Selection Sort on small or partially sorted datasets.

Example: For n = 10,000, Binary Insertion Sort completes faster than Selection Sort due to fewer comparisons, but for n = 100,000, shifts make it slow.

5. Conclusion

Selection Sort: Best for small datasets and educational purposes; simple and memory-efficient.

Binary Insertion Sort: Reduces comparisons, better for nearly sorted arrays, but shifting remains the bottleneck.

Recommendation: For large datasets (≥10,000 elements), use more advanced algorithms like MergeSort or HeapSort.

Summary: Binary Insertion Sort improves comparison efficiency, while Selection Sort provides predictable, straightforward behavior. For practical applications, Binary Insertion Sort is preferable for partially sorted arrays, but neither is suitable for very large datasets due to $O(n^2)$ time complexity.