### **1. O(n log n) Sorting: Merge Sort**

a. **Chosen Sort**: Merge Sort was chosen due to its efficient performance and its guaranteed O(n log n) time complexity.

b. **Code Source**: The implementation of Merge Sort was adapted from online resources: https://www.programiz.com/java-programming/examples/merge-sort

c. **Consistency of Timings**: The timings for Merge Sort were consistent across multiple runs, with minor variations

d. **Growth Rate Comparison**: The observed growth rate of Merge Sort matched the theoretical growth rate of O(n log n) closely, especially for larger array sizes.

**2. O(n²) Sorting: Bubble Sort**

a. **Chosen Sort**: Bubble Sort was selected as a simple and well-known O(n²) sorting algorithm, suitable for educational purposes.

b. **Code Source**: The Bubble Sort implementation was adapted from online resources: https://www.geeksforgeeks.org/java-program-for-bubble-sort/

c. **Consistency of Timings**: The timings for Bubble Sort were less consistent, with variations observed in different runs.

d. **Growth Rate Comparison**: The growth rate observed for Bubble Sort did not match the theoretical O(n²) growth perfectly.

**3. Arrays.sort()**

a. **Theoretical Growth Rate**: The theoretical growth rate for Arrays.sort() is O(n log n) for average and worst-case scenarios, as it uses a dual-pivot quicksort algorithm.

b. **Consistency of Timings**: The timings for Arrays.sort() were consistent across multiple runs, demonstrating its efficiency and reliability.

c. **Growth Rate Comparison**: The growth rate observed for Arrays.sort() closely matched the theoretical O(n log n) growth rate.

|  |  |  |  |
| --- | --- | --- | --- |
| **Array Size** | **Bubble Sort (O(n²))** | **Merge Sort (O(n log n))** | **Arrays.sort()** |
| 20 | 0 ms | 0 ms | 1 ms |
| 100 | 0 ms | 0 ms | 1 ms |
| 1000 | 4 ms | 0 ms | 0 ms |
| 5000 | 27 ms | 1 ms | 1 ms |

MergeSort - Test array before sorting:

-1082849689

-647041088

275571094

-1499259514

-1682361150

-1003159844

207422276

-2035338368

-581413879

425267603

307815779

490615325

698761915

-423623883

1293634705

-1134781999

754666154

371757901

-350742165

-1044908262

MergeSort - Test array after sorting:

-2035338368

-1682361150

-1499259514

-1134781999

-1082849689

-1044908262

-1003159844

-647041088

-581413879

-423623883

-350742165

207422276

275571094

307815779

371757901

425267603

490615325

698761915

754666154

1293634705

BubbleSort - Test array before sorting:

1370634912

825703473

1783334634

-1023657686

285871303

1867289764

-283765058

1463871678

-300750626

-511945396

-686231847

-1522312764

-1201945595

1273666049

1714381472

476115726

-50509304

686701210

866867149

1914051275

BubbleSort - Test array after sorting:

-1522312764

-1201945595

-1023657686

-686231847

-511945396

-300750626

-283765058

-50509304

285871303

476115726

686701210

825703473

866867149

1273666049

1370634912

1463871678

1714381472

1783334634

1867289764

1914051275

ArraysSort - Test array before sorting:

-178420303

-1507181313

-1836136864

-1029372724

1181161298

2002461308

-617805998

940305189

1726744997

1167615231

143622489

-692982048

696751620

888405876

-651182454

1891086257

-2077908250

1139540814

1979278350

858542919

ArraysSort - Test array after sorting:

-2077908250

-1836136864

-1507181313

-1029372724

-692982048

-651182454

-617805998

-178420303

143622489

696751620

858542919

888405876

940305189

1139540814

1167615231

1181161298

1726744997

1891086257

1979278350

2002461308