# Table A

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Settings | Bubble sort | Enhanced Bubble sort | Selection sort | Merge sort | Quick sort |
| size: 50 Highest number: 500 Number of arrays: 1000 | Time = 46,352,700  nanoseconds  Swaps = 613837 | Time = 29,037,000  nanoseconds  Swaps = 616 | Time = 26,059,800  nanoseconds  Swaps = 45 | Time = 135,484,700  nanoseconds  Swaps = 115,608 | Time = 15,975,200  nanoseconds  Swaps = 134,117 |
| size: 1000 Highest number: 500 Number of arrays: 1000 | Time = 1,168,139,200  nanoseconds  Swaps = 249,509,446 | Time = 7,291,583,000  nanoseconds  Swaps = 246,502 | Time = 568,753,800  nanoseconds  Swaps = 994 | Time = 724,685,300  nanoseconds  Swaps = 4,387,381 | Time = 104,996,200  nanoseconds  Swaps = 5,375,343 |
| size: 50 Highest number: 500 Number of arrays: 1,000,000 | Time = 7,880,928,700  nanoseconds  Swaps = 611,260,305 | Time = 7,977,521,000  nanoseconds  Swaps = 676 | Time = 4,767,911,500  nanoseconds  Swaps = 46 | Time = 1,0271,165,100  nanoseconds  Swaps = 115,329,932 | Time = 2,963,325,200  nanoseconds  Swaps = 134,623,682 |
| size: 100,000 Highest number: 500 Number of arrays: 1 | Time = 32,885,697,600  nanoseconds  Swaps = 2,499,038,887 | Time = 26,926,697,800  nanoseconds  Swaps = 2,490,051,579 | Time = 8,602,416,700  nanoseconds  Swaps = 99,781 | Time = 248,305,400  nanoseconds  Swaps = 775,339 | Time = 136,755,400  nanoseconds  Swaps = 709,809 |
| size: 1000 Highest number: 500 Number of arrays: 1000 | Time = 1,157,891,500  nanoseconds  Swaps = 249,483,512 | Time = 4,930,265,200  nanoseconds  Swaps = 248,604 | Time = 495,732,300  nanoseconds  Swaps = 990 | Time = 511,738,800  nanoseconds  Swaps = 4,388,771 | Time = 94,869,000  nanoseconds  Swaps = 5,305,094 |
| size: 10000 Highest number: 500 Number of arrays: 100 | Time = 15,997,948,100  nanoseconds  Swaps = 2,496,114,646 | Time = 30,020,358,600  nanoseconds  Swaps = 24,687,085 | Time = 7,030,777,300  nanoseconds  Swaps = 9,979 | Time = 890,856,700  nanoseconds  Swaps = 6,120,921 | Time = 181,409,700  nanoseconds  Swaps = 6,547,000 |
|  |  |  |  |  |  |

# Table B

|  |  |  |
| --- | --- | --- |
| Settings | Quick sort | Merge sort |
| size: 1000000 Highest number: 500 Number of arrays: 1 | Time = 798,965,700  nanoseconds  Swaps = 6,370,406 | Time = 786,928,900  nanoseconds  Swaps = 9,381,039 |
| size: 1000000 Highest number: 50000 Number of arrays: 1 | Time = 177,160,700  nanoseconds  Swaps = 11,307,960 | Time = 808,955,600  nanoseconds  Swaps = 9,395,495 |

# Questions

1.Does the growth differences in runtimes match what has been discussed in class?

**Yes. The growth difference in runtime is exactly same what has been discussed during class. As bubble sort being the slowest and quick sort is faster and so on.**

2. Does the difference in time between the various sorting algorithms match what we would expect?

**Yes. I expected bubble sort algorithms to perform as well, or even better than merge and quick sort for smaller size input arrays. merge and quick sort algorithms perform at almost constant time despite increasing sizes of inputs, making them the fastest for large array sizes.**

3. Does the number of swaps between Bubble sort, Enhanced Bubble sort, and Selection sort match what we would expect?

**Yes. Bubble, and enhanced bubble sort algorithms perform a lot of swaps since they make swaps for each adjacent elements in the array for each pass, requiring passes equal or close to the size of the array (worst case = n^2). However, the number of swaps in selection sort remain relatively low despite increasing array sizes since it performs swaps corresponding to the minimum value in the array per pass.**

4. Observe the results from Table B. For the first test did Merge sort perform better or worse than Quick sort? Why? Which performed better during the second test? Why?

**The first test in table B showed that merge and quick sort had relatively equal performance, with quick sort performing slightly better than the merge sort with regards to time taken. However, in the second test, the quick sort algorithm significantly performed better then merge sort algorithm with regards to time taken to sort the array as the value complexity in the arrays increased. This is because the quick sort algorithm uses a binary tree approach which takes constant time to sort values of any size.**

5. The Arrays class in java uses Quick sort for sorting arrays of primitive data types. Looking at the data gathered, does it make sense to use this algorithm, or would you suggest a different algorithm?

**No. Given the consistent performance displayed by quick sort despite the increases in array sizes and value ranges, I would suggest sticking with quick sort for sorting arrays.**