Program Structures and Algorithms
Spring 2024

NAME: Ting Guo
NUID: 002834835
GITHUB LINK:https://github.com/Nangongnuanshan/INFO6205

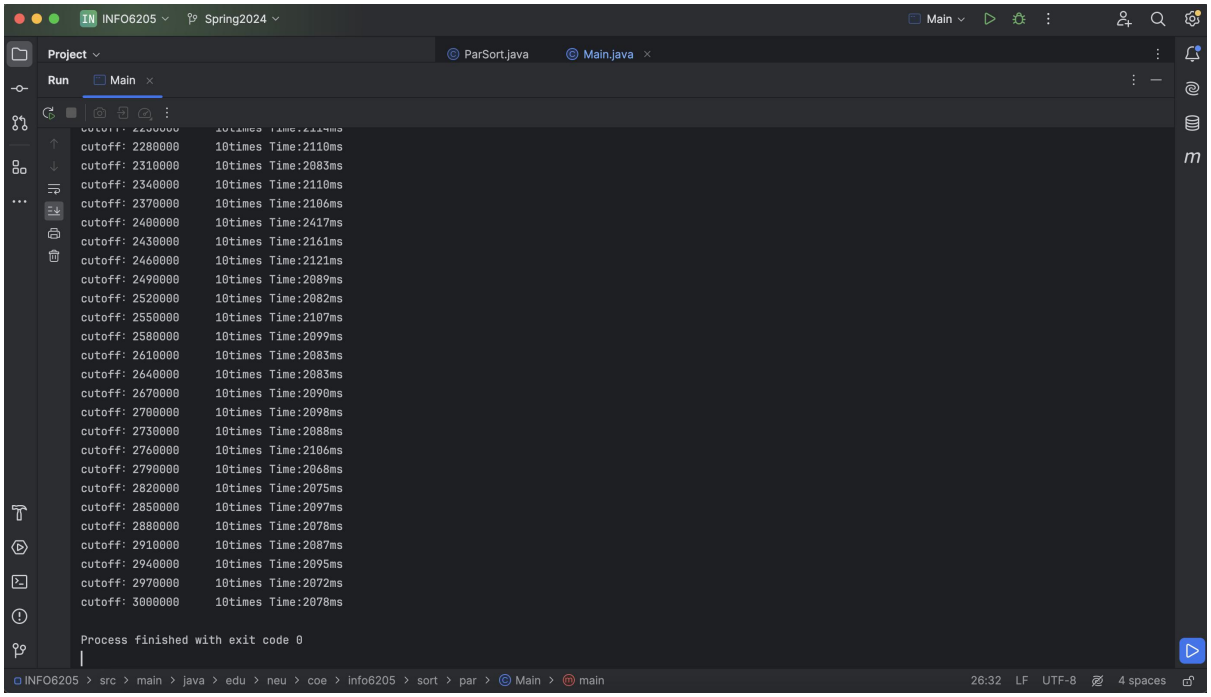
Task:Assignment 5

Conclusion:

According to the code, when the length of array is less than cutoff, the code would use Array.sort() to sort the array. If the length of array is more than cutoff, the code would use our own sort(a kind of merge sort). Now, Array.sort() is based on Dual-Pivot Quicksort .
For small to medium-sized datasets, or on single-core processors, Dual-Pivot Quicksort might be faster than parallel merge sort due to its lower overhead and efficient data partitioning strategy. For large datasets, especially on multi-core or multiprocessor systems, parallel merge sort may offer better performance because it can fully utilize parallel processing capabilities to speed up the sorting process.
According to the result, 2000000(the length of array) is not small or medium. So, for large datasets, we tend to use our own sort. When cutoff < length, we use our own sort and it does run quicker than using Array.sort()(cutoff > length).

```
public static void sort(int[] array, int from, int to) {
    if (to - from < cutoff) Arrays.sort(array, from, to);
    else {
        // FIXME next few lines should be removed from public repo.
        CompletableFuture<int[]> parsort1 = parsort(array, from, to: from + (to - from) / 2);
        CompletableFuture<int[]> parsort2 = parsort(array, from: from + (to - from) / 2, to);
        CompletableFuture<int[]> parsort = parsort1.thenCombine(parsort2, (xs1, xs2) -> {
            int[] result = new int[xs1.length + xs2.length];
            // TO IMPLEMENT
            int i = 0;
            int j = 0;
            for (int k = 0; k < result.length; k++) {
                if (i >= xs1.length) {
                    result[k] = xs2[j++];
                } else if (j >= xs2.length) {
                    result[k] = xs1[i++];
                } else if (xs2[j] < xs1[i]) {
                    result[k] = xs2[j++];
                } else {
                    result[k] = xs1[i++];
                }
            }
            return result;
        });
```

Result:


INFO6205
Spring2024
Main
Project
ParSort.java
Main.java
Run
Main
cutoff: 2280000 10times Time:2110ms
cutoff: 2310000 10times Time:2083ms
cutoff: 2340000 10times Time:2110ms
cutoff: 2370000 10times Time:2106ms
cutoff: 2400000 10times Time:2417ms
cutoff: 2430000 10times Time:2161ms
cutoff: 2460000 10times Time:2121ms
cutoff: 2490000 10times Time:2089ms
cutoff: 2520000 10times Time:2082ms
cutoff: 2550000 10times Time:2107ms
cutoff: 2580000 10times Time:2099ms
cutoff: 2610000 10times Time:2083ms
cutoff: 2640000 10times Time:2083ms
cutoff: 2670000 10times Time:2090ms
cutoff: 2700000 10times Time:2098ms
cutoff: 2730000 10times Time:2088ms
cutoff: 2760000 10times Time:2106ms
cutoff: 2790000 10times Time:2068ms
cutoff: 2820000 10times Time:2075ms
cutoff: 2850000 10times Time:2097ms
cutoff: 2880000 10times Time:2078ms
cutoff: 2910000 10times Time:2087ms
cutoff: 2940000 10times Time:2095ms
cutoff: 2970000 10times Time:2072ms
cutoff: 3000000 10times Time:2078ms
Process finished with exit code 0
INFO6205 › src › main › java › edu › neu › coe › info6205 › sort › par › Main › main
26:32 LF UTF-8 4 spaces