

Diksha Godse – (002116411)

Program Structures & Algorithms

Spring 2022

Assignment No. 4

Task

Task is to implement a parallel sorting algorithm such that each partition of the array is sorted in parallel. You will consider two different schemes for deciding whether to sort in parallel.

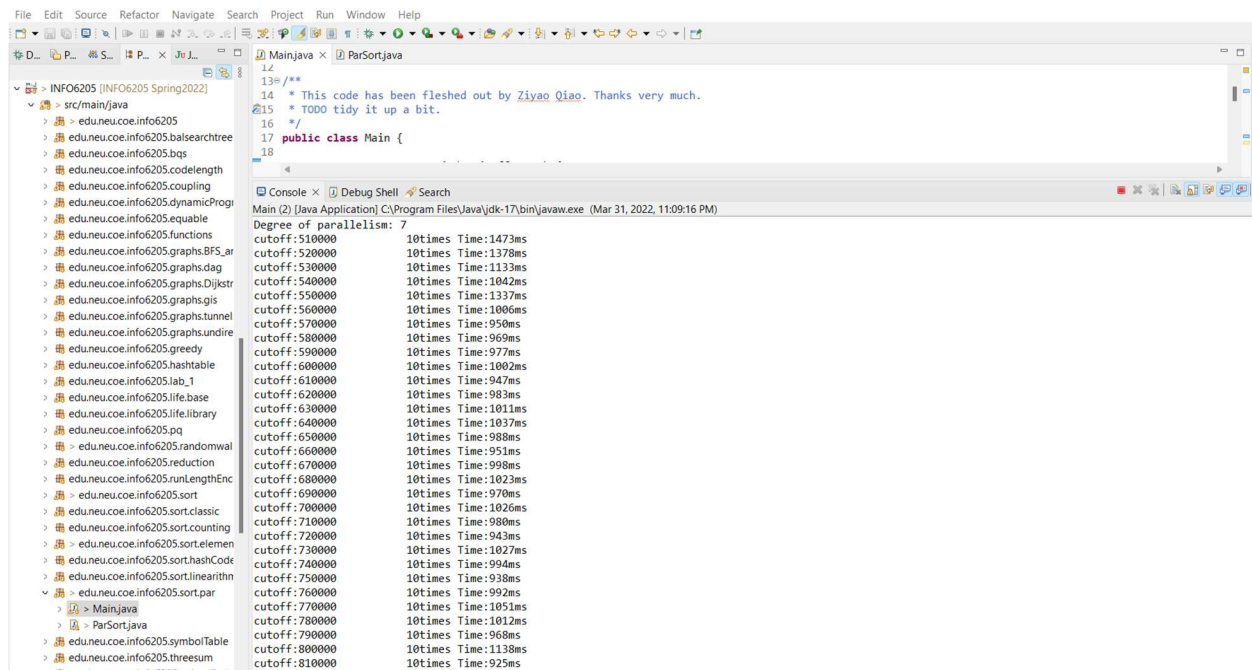
1. A cutoff (defaults to, say, 1000) which you will update according to the first argument in the command line when running. It's your job to experiment and come up with a good value for this cutoff. If there are fewer elements to sort than the cutoff, then you should use the system sort instead.
2. Recursion depth or the number of available threads. Using this determination, you might decide on an ideal number (t) of separate threads (stick to powers of 2) and arrange for that number of partitions to be parallelized (by preventing recursion after the depth of $\lg t$ is reached).
3. An appropriate combination of these.

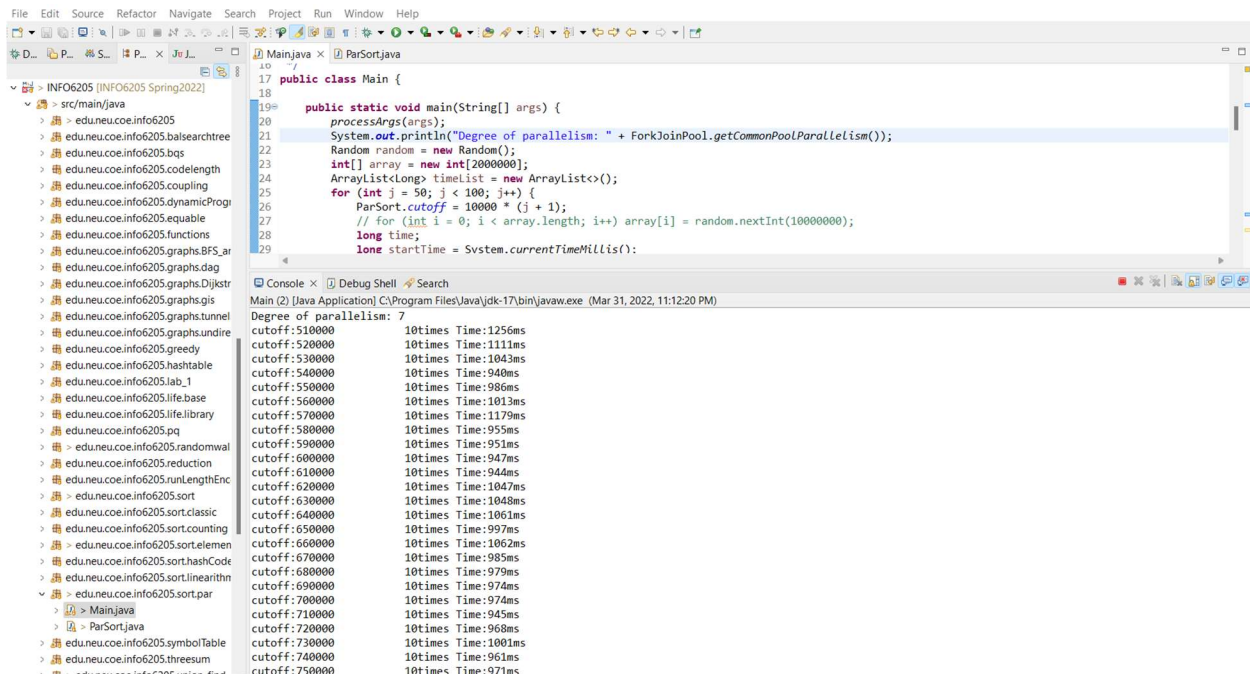
You must prepare a report that shows the results of your experiments and draws a conclusion (or more) about the efficacy of this method of parallelizing sort. Your experiments should involve sorting arrays of sufficient size for the parallel sort to make a difference. You should run with many different array sizes (they must be sufficiently large to make parallel sorting worthwhile, obviously) and different cutoff schemes.

Output

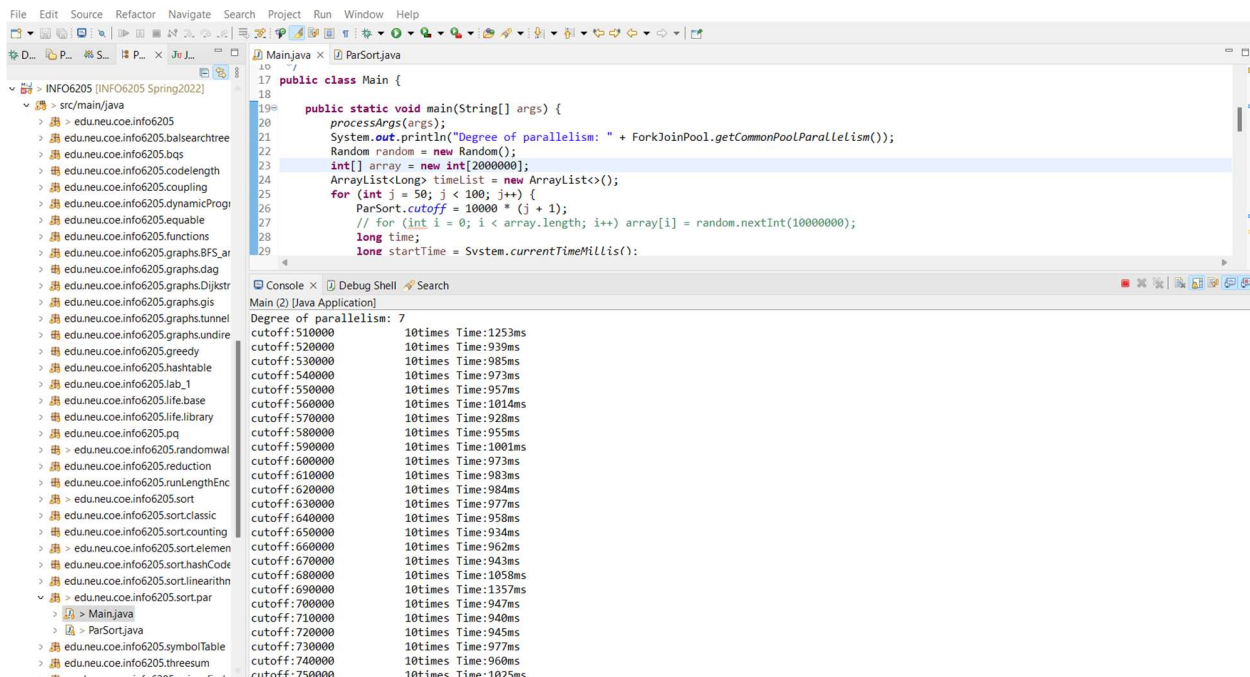
In main.java, I started by writing thread count (power of 2) and fixed array size.

1. Thread Count: 2
Array size: 2000000





4. Thread Count: 16 Array size: 2000000



5. Thread Count: 32 Array size: 2000000

```

10 class ParSort {
11
12     public static int cutoff = 1000;
13     public static int thread_count = 32;
14     public static ForkJoinPool thread_pool = new ForkJoinPool(thread_count);
15
16     public static void sort(int[] array, int from, int to) {
17         if (to - from < cutoff) Arrays.sort(array, from, to);
18         else {
19             // FIXME next few lines should be removed from public repo.
20             CompletableFuture<int[]> parsort1 = parsort(array, from, from + (to - from) / 2); // TO IMPLEMENT
21             CompletableFuture<int[]> parsort2 = parsort(array, from + (to - from) / 2, to); // TO IMPLEMENT
22             CompletableFuture<int[]> parsort = parsort1.thenCombine(parsort2, (xs1, xs2) -> {

```

Console Output:

```

Main (2) [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (Mar 31, 2022, 11:53:37 PM)
Degree of parallelism: 7
cutoff:510000      10times Time:1373ms
cutoff:520000      10times Time:1012ms
cutoff:530000      10times Time:1542ms
cutoff:540000      10times Time:1317ms
cutoff:550000      10times Time:1113ms
cutoff:560000      10times Time:1153ms
cutoff:570000      10times Time:1188ms
cutoff:580000      10times Time:1257ms
cutoff:590000      10times Time:2930ms
cutoff:600000      10times Time:2944ms
cutoff:610000      10times Time:2993ms
cutoff:620000      10times Time:2964ms
cutoff:630000      10times Time:2974ms
cutoff:640000      10times Time:2972ms
cutoff:650000      10times Time:2615ms
cutoff:660000      10times Time:982ms
cutoff:670000      10times Time:1007ms
cutoff:680000      10times Time:946ms
cutoff:690000      10times Time:981ms
cutoff:700000      10times Time:954ms
cutoff:710000      10times Time:999ms
cutoff:720000      10times Time:1007ms
cutoff:730000      10times Time:1029ms
cutoff:740000      10times Time:1000ms
cutoff:750000      10times Time:1073ms

```

6. Thread Count: 64 Array size: 2000000

```

10 class ParSort {
11
12     public static int cutoff = 1000;
13     public static int thread_count = 64;
14     public static ForkJoinPool thread_pool = new ForkJoinPool(thread_count);
15
16     public static void sort(int[] array, int from, int to) {
17         if (to - from < cutoff) Arrays.sort(array, from, to);
18         else {
19             // FIXME next few lines should be removed from public repo.
20             CompletableFuture<int[]> parsort1 = parsort(array, from, from + (to - from) / 2); // TO IMPLEMENT
21             CompletableFuture<int[]> parsort2 = parsort(array, from + (to - from) / 2, to); // TO IMPLEMENT
22             CompletableFuture<int[]> parsort = parsort1.thenCombine(parsort2, (xs1, xs2) -> {

```

Console Output:

```

Main (2) [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (Mar 31, 2022, 11:17:28 PM)
Degree of parallelism: 7
cutoff:510000      10times Time:1358ms
cutoff:520000      10times Time:1077ms
cutoff:530000      10times Time:1113ms
cutoff:540000      10times Time:960ms
cutoff:550000      10times Time:964ms
cutoff:560000      10times Time:1017ms
cutoff:570000      10times Time:1004ms
cutoff:580000      10times Time:932ms
cutoff:590000      10times Time:989ms
cutoff:600000      10times Time:977ms
cutoff:610000      10times Time:1009ms
cutoff:620000      10times Time:990ms
cutoff:630000      10times Time:949ms
cutoff:640000      10times Time:957ms
cutoff:650000      10times Time:973ms
cutoff:660000      10times Time:932ms
cutoff:670000      10times Time:978ms
cutoff:680000      10times Time:1008ms
cutoff:690000      10times Time:996ms
cutoff:700000      10times Time:1041ms
cutoff:710000      10times Time:986ms
cutoff:720000      10times Time:973ms
cutoff:730000      10times Time:959ms
cutoff:740000      10times Time:1020ms
cutoff:750000      10times Time:1070ms

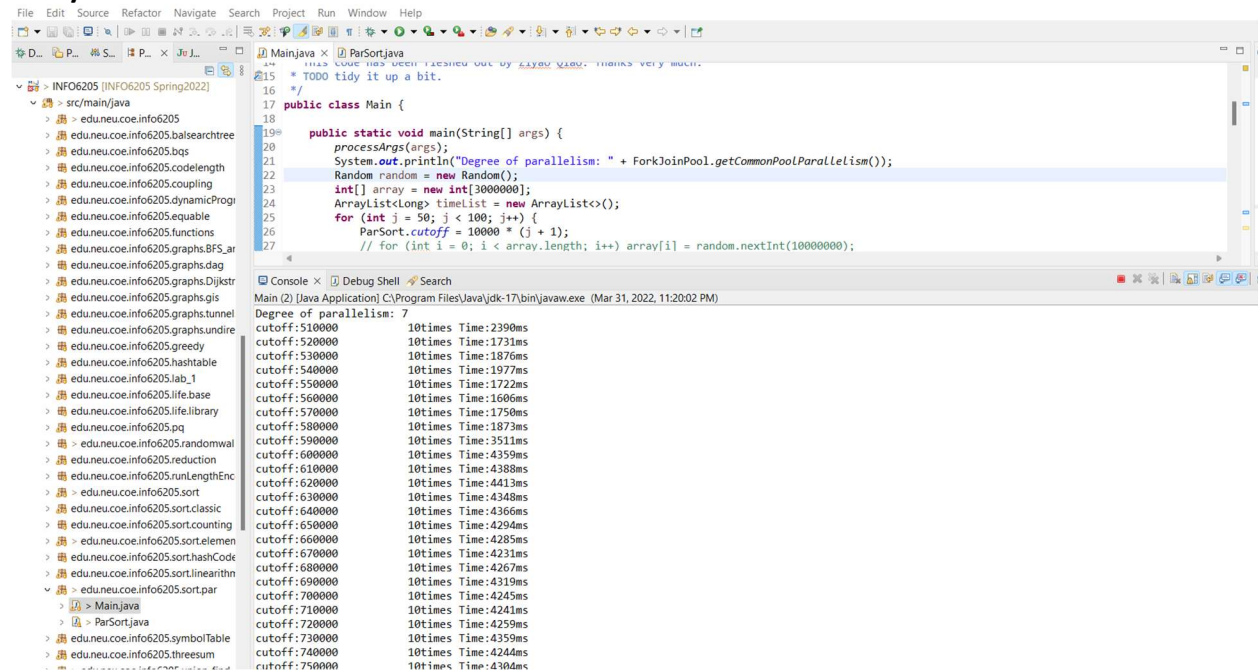
```

The above experiment was conducted using a cutoff range of 510000–990000, an array size of 2000000, and a thread count of 2-64. From the above tests, it is

noticed that the optimal cut-off value is between 580000 - 650000. As a result, the thread count is 620000, and it is most efficient when the thread count is 8. Now, let's validate this by modifying main.java with respect to array size and running an experiment with a thread count of 8 and a range of 510000-920000 for varied sizes of the array.

1. Thread Count: 8

Array size: 3000000



The screenshot shows an IDE with the file `ParSort.java` open. The code defines a `Main` class with a `main` method that uses `ForkJoinPool` for parallel sorting. The array size is set to 3,000,000. The console output shows the results of running the program with a degree of parallelism of 7. The output lists the cut-off value and execution time for various array sizes, ranging from 510,000 to 750,000. The times are generally around 10 times 170ms to 424ms.

```
public class Main {
    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " + ForkJoinPool.getCommonPoolParallelism());
        Random random = new Random();
        int[] array = new int[3000000];
        ArrayList<Long> timeList = new ArrayList<>();
        for (int j = 50; j < 100; j++) {
            ParSort.cutoff = 10000 * (j + 1);
            // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
        }
    }
}
```

Console Output:

```
Main (2) [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (Mar 31, 2022, 11:20:02 PM)
Degree of parallelism: 7
cutoff:510000      10times Time:2390ms
cutoff:520000      10times Time:1731ms
cutoff:530000      10times Time:1876ms
cutoff:540000      10times Time:1977ms
cutoff:550000      10times Time:1722ms
cutoff:560000      10times Time:1606ms
cutoff:570000      10times Time:1750ms
cutoff:580000      10times Time:1873ms
cutoff:590000      10times Time:3511ms
cutoff:600000      10times Time:4359ms
cutoff:610000      10times Time:4380ms
cutoff:620000      10times Time:4413ms
cutoff:630000      10times Time:4348ms
cutoff:640000      10times Time:4366ms
cutoff:650000      10times Time:4294ms
cutoff:660000      10times Time:4285ms
cutoff:670000      10times Time:4231ms
cutoff:680000      10times Time:4267ms
cutoff:690000      10times Time:4319ms
cutoff:700000      10times Time:4245ms
cutoff:710000      10times Time:4241ms
cutoff:720000      10times Time:4259ms
cutoff:730000      10times Time:4359ms
cutoff:740000      10times Time:4244ms
cutoff:750000      10times Time:4304ms
```

2. Thread Count: 8

Array size: 4000000

```
File Edit Source Refactor Navigate Search Project Run Window Help
Main.java x ParSort.java
INFO6205 (INFO6205 Spring2022)
src/main/java
edu.neu.coe.info6205
edu.neu.coe.info6205.balsearchtree
edu.neu.coe.info6205.bqs
edu.neu.coe.info6205.codelength
edu.neu.coe.info6205.coupling
edu.neu.coe.info6205.dynamicProgi
edu.neu.coe.info6205.equable
edu.neu.coe.info6205.functions
edu.neu.coe.info6205.graphs.BFS_ar
edu.neu.coe.info6205.graphs.dag
edu.neu.coe.info6205.graphs.Dijkstra
edu.neu.coe.info6205.graphs.gis
edu.neu.coe.info6205.graphs.gis
edu.neu.coe.info6205.graphs.tunnel
edu.neu.coe.info6205.graphs.undire
edu.neu.coe.info6205.greedy
edu.neu.coe.info6205.hashtable
edu.neu.coe.info6205.lab_1
edu.neu.coe.info6205.life.base
edu.neu.coe.info6205.life.library
edu.neu.coe.info6205.pq
edu.neu.coe.info6205.randomwal
edu.neu.coe.info6205.reduction
edu.neu.coe.info6205.runLengthEnc
edu.neu.coe.info6205.sort
edu.neu.coe.info6205.sort.classic
edu.neu.coe.info6205.sort.counting
edu.neu.coe.info6205.sort.elemen
edu.neu.coe.info6205.sort.hashCode
edu.neu.coe.info6205.sort.linearithn
edu.neu.coe.info6205.sort.par
Main.java
ParSort.java
edu.neu.coe.info6205.symbolTable
edu.neu.coe.info6205.threesum
edu.neu.coe.info6205.threesum

Main (2) [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (Mar 31, 2022, 11:22:24 PM)
Degree of parallelism: 7
cutoff:510000 10times Time:2728ms
cutoff:520000 10times Time:2300ms
cutoff:530000 10times Time:2336ms
cutoff:540000 10times Time:2281ms
cutoff:550000 10times Time:2256ms
cutoff:560000 10times Time:2344ms
cutoff:570000 10times Time:2326ms
cutoff:580000 10times Time:2440ms
cutoff:590000 10times Time:2247ms
cutoff:600000 10times Time:2817ms
cutoff:610000 10times Time:5308ms
cutoff:620000 10times Time:3283ms
cutoff:630000 10times Time:2525ms
cutoff:640000 10times Time:2212ms
cutoff:650000 10times Time:2162ms
cutoff:660000 10times Time:2014ms
cutoff:670000 10times Time:2042ms
cutoff:680000 10times Time:2045ms
cutoff:690000 10times Time:2343ms
cutoff:700000 10times Time:2242ms
cutoff:710000 10times Time:2273ms
cutoff:720000 10times Time:2610ms
cutoff:730000 10times Time:2296ms
```

2. Thread Count: 8 Array size: 500000

```
File Edit Source Refactor Navigate Search Project Run Window Help
Main.java x ParSort.java
INFO6205 (INFO6205 Spring2022)
src/main/java
edu.neu.coe.info6205
edu.neu.coe.info6205.balsearchtree
edu.neu.coe.info6205.bqs
edu.neu.coe.info6205.codelength
edu.neu.coe.info6205.coupling
edu.neu.coe.info6205.dynamicProgi
edu.neu.coe.info6205.equable
edu.neu.coe.info6205.functions
edu.neu.coe.info6205.graphs.BFS_ar
edu.neu.coe.info6205.graphs.dag
edu.neu.coe.info6205.graphs.Dijkstra
edu.neu.coe.info6205.graphs.gis
edu.neu.coe.info6205.graphs.gis
edu.neu.coe.info6205.graphs.tunnel
edu.neu.coe.info6205.graphs.undire
edu.neu.coe.info6205.greedy
edu.neu.coe.info6205.hashtable
edu.neu.coe.info6205.lab_1
edu.neu.coe.info6205.life.base
edu.neu.coe.info6205.life.library
edu.neu.coe.info6205.pq
edu.neu.coe.info6205.randomwal
edu.neu.coe.info6205.reduction
edu.neu.coe.info6205.runLengthEnc
edu.neu.coe.info6205.sort
edu.neu.coe.info6205.sort.classic
edu.neu.coe.info6205.sort.counting
edu.neu.coe.info6205.sort.elemen
edu.neu.coe.info6205.sort.hashCode
edu.neu.coe.info6205.sort.linearithn
edu.neu.coe.info6205.sort.par
Main.java
ParSort.java
edu.neu.coe.info6205.symbolTable
edu.neu.coe.info6205.threesum
edu.neu.coe.info6205.threesum

Main (2) [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (Mar 31, 2022, 11:24:32 PM)
Degree of parallelism: 7
cutoff:510000 10times Time:3177ms
cutoff:520000 10times Time:2591ms
cutoff:530000 10times Time:2675ms
cutoff:540000 10times Time:2696ms
cutoff:550000 10times Time:2696ms
cutoff:560000 10times Time:2663ms
cutoff:570000 10times Time:2566ms
cutoff:580000 10times Time:2602ms
cutoff:590000 10times Time:2669ms
cutoff:600000 10times Time:2852ms
cutoff:610000 10times Time:2523ms
cutoff:620000 10times Time:2573ms
cutoff:630000 10times Time:2539ms
cutoff:640000 10times Time:2595ms
cutoff:650000 10times Time:2563ms
cutoff:660000 10times Time:2527ms
cutoff:670000 10times Time:2692ms
cutoff:680000 10times Time:2691ms
cutoff:690000 10times Time:2820ms
cutoff:700000 10times Time:2789ms
cutoff:710000 10times Time:2918ms
```

2. Thread Count: 8 Array size: 600000

```

15 // * 1000 tidy it up a bit.
16 */
17 public class Main {
18
19     public static void main(String[] args) {
20         processArgs(args);
21         System.out.println("Degree of parallelism: " + ForkJoinPool.getCommonPoolParallelism());
22         Random random = new Random();
23         int[] array = new int[6000000];
24         ArrayList<Long> timeList = new ArrayList<>();
25         for (int j = 50; j < 100; j++) {
26             ParSort.cutoff = 10000 * (j + 1);
27             // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
28             long time;
29             long startTime = System.currentTimeMillis();
30             for (int t = 0; t < 10; t++) {
31                 for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);

```

Main (2) [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (Mar 31, 2022, 11:26:04 PM)		
Degree of parallelism: 7		
cutoff:510000	10times	Time:3617ms
cutoff:520000	10times	Time:3056ms
cutoff:530000	10times	Time:3070ms
cutoff:540000	10times	Time:2987ms
cutoff:550000	10times	Time:3102ms
cutoff:560000	10times	Time:3043ms
cutoff:570000	10times	Time:2999ms
cutoff:580000	10times	Time:3029ms
cutoff:590000	10times	Time:3634ms
cutoff:600000	10times	Time:2549ms
cutoff:610000	10times	Time:2232ms
cutoff:620000	10times	Time:2264ms
cutoff:630000	10times	Time:2713ms
cutoff:640000	10times	Time:2416ms
cutoff:650000	10times	Time:2251ms
cutoff:660000	10times	Time:2197ms
cutoff:670000	10times	Time:2444ms
cutoff:680000	10times	Time:2311ms
cutoff:690000	10times	Time:2222ms
cutoff:700000	10times	Time:2398ms
cutoff:710000	10times	Time:2779ms

Conclusion:

According to the results of the above studies, even though we multiplied the Array size from 2000000 to 6000000 by 1000000, it took longer, but the method works best when the cutoff value is 620000 and the thread count is 8.

Cut-off time: 620000

Thread count: 8