Program Structures and Algorithms
Spring 2022
Assignment 4 (Parallel sort)

Name: Vraj Himanshu Reshamdalal
(**NUID**): 002927484

Task:
Please see the presentation on *Assignment on Parallel Sorting* under the *Exams. etc.* module. Your 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.

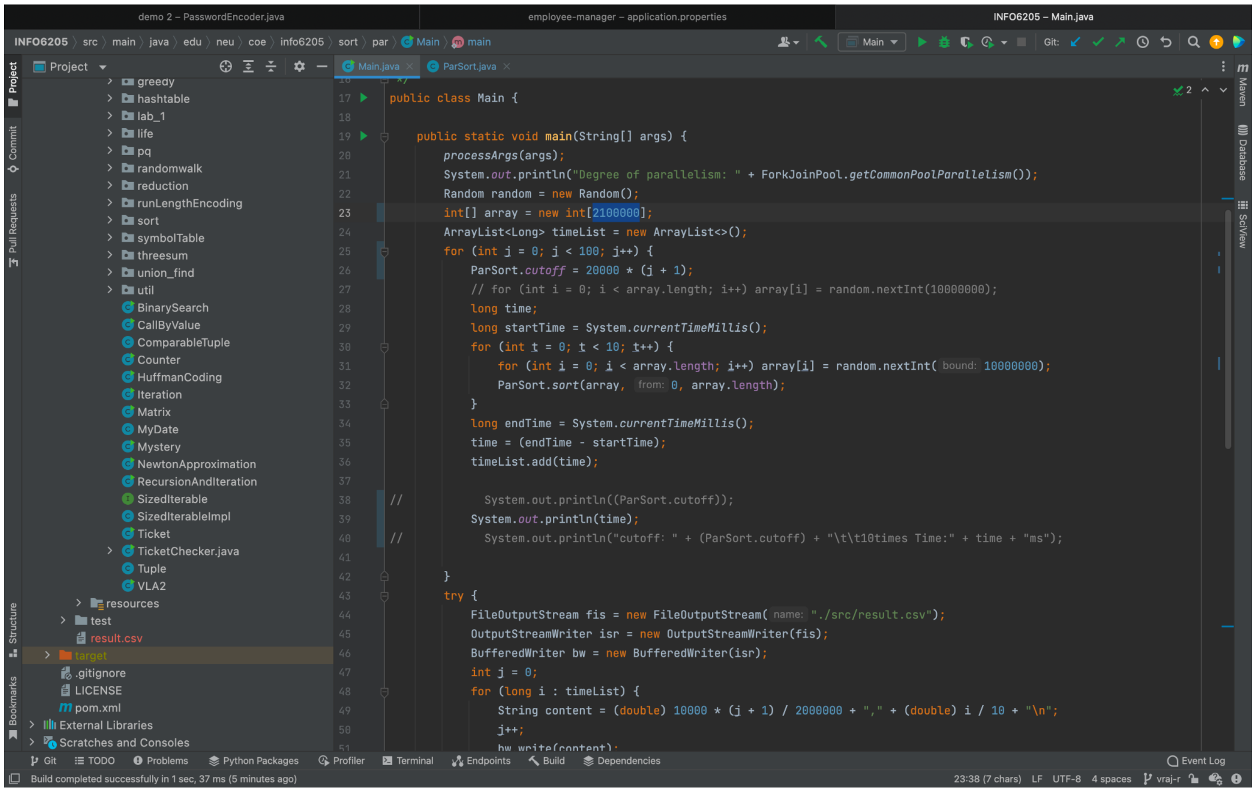
There is a *Main* class and the *ParSort* class in the *sort.par* package of the INFO6205 repository. The *Main* class can be used as is but the *ParSort* class needs to be implemented where you see "TODO..." [it turns out that these TODOs are already implemented].
Unless you have a good reason not to, you should just go along with the Java8-style future implementations provided for you in the class repository.
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.
For varying the number of threads available, you might want to consult the following resources:

- https://www.callicoder.com/java-8-completablefuture-tutorial/#a-note-about-executor-and-thread-pool (Links to an external site.)
- https://stackoverflow.com/questions/36569775/how-to-set-forkjoinpool-with-the-desired-number-of-worker-threads-in-completable

---

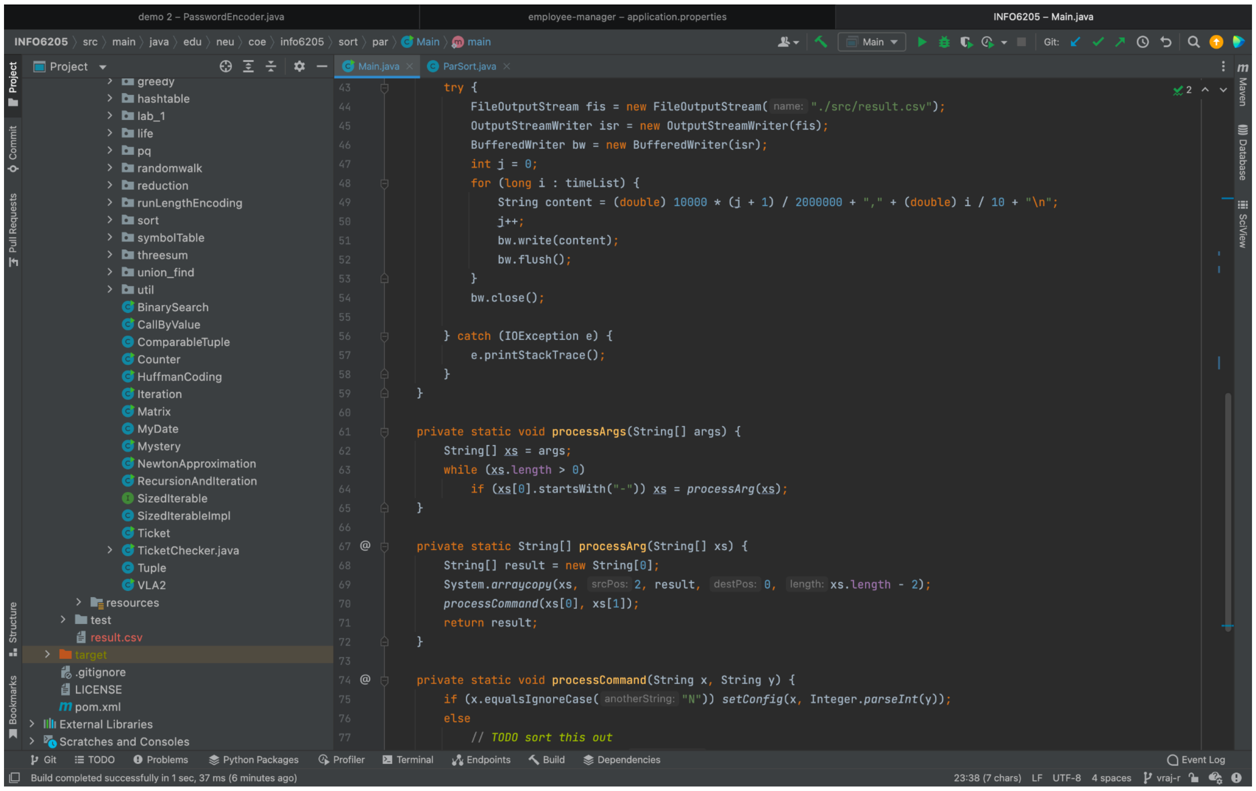
CODE

File name: Main.java



```java
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[2100000];
        ArrayList<Long> timeList = new ArrayList<>();
        for (int j = 0; j < 100; j++) {
            ParSort.cutoff = 20000 * (j + 1);
            // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
            long time;
            long startTime = System.currentTimeMillis();
            for (int t = 0; t < 10; t++) {
                for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
                ParSort.sort(array, 0, array.length);
            }
            long endTime = System.currentTimeMillis();
            time = (endTime - startTime);
            timeList.add(time);

//            System.out.println((ParSort.cutoff));
            System.out.println(time);
//            System.out.println("cutoff: " + (ParSort.cutoff) + "\t\t10times Time:" + time + "ms");

        }
        try {
            FileOutputStream fis = new FileOutputStream("./src/result.csv");
            OutputStreamWriter isr = new OutputStreamWriter(fis);
            BufferedWriter bw = new BufferedWriter(isr);
            int j = 0;
            for (long i : timeList) {
                String content = (double) 10000 * (j + 1) / 2000000 + "," + (double) i / 10 + "\n";
                j++;
                bw.write(content);
```

Build completed successfully in 1 sec, 37 ms (5 minutes ago)


demo 2 – PasswordEncoder.java | employee-manager – application.properties | INFO6205 – Main.java

INFO6205 〉 src 〉 main 〉 java 〉 edu 〉 neu 〉 coe 〉 info6205 〉 sort 〉 par 〉 Main 〉 main

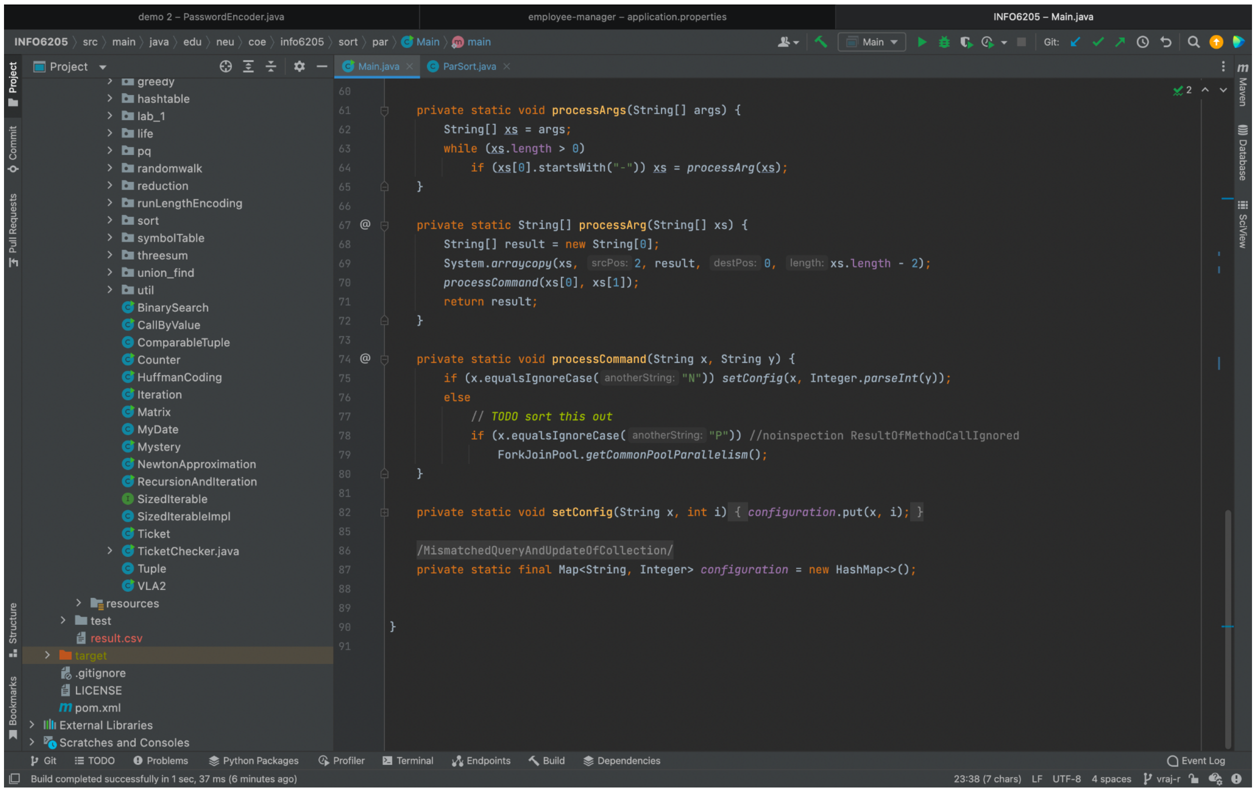
Main.java | ParSort.java

```
greedy
hashtable
lab_1
life
pq
randomwalk
reduction
runLengthEncoding
sort
symbolTable
threesum
union_find
util
BinarySearch
CallByValue
ComparableTuple
Counter
HuffmanCoding
Iteration
Matrix
MyDate
Mystery
NewtonApproximation
RecursionAndIteration
SizedIterable
SizedIterableImpl
Ticket
TicketChecker.java
Tuple
VLA2
resources
test
result.csv
target
.gitignore
LICENSE
pom.xml
External Libraries
Scratches and Consoles
```

```java
        try {
            FileOutputStream fis = new FileOutputStream( name: "./src/result.csv");
            OutputStreamWriter isr = new OutputStreamWriter(fis);
            BufferedWriter bw = new BufferedWriter(isr);
            int j = 0;
            for (long i : timeList) {
                String content = (double) 10000 * (j + 1) / 2000000 + "," + (double) i / 10 + "\n";
                j++;
                bw.write(content);
                bw.flush();
            }
            bw.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void processArgs(String[] args) {
        String[] xs = args;
        while (xs.length > 0)
            if (xs[0].startsWith("-")) xs = processArg(xs);
    }

    private static String[] processArg(String[] xs) {
        String[] result = new String[0];
        System.arraycopy(xs,  srcPos: 2, result,  destPos: 0,  length: xs.length - 2);
        processCommand(xs[0], xs[1]);
        return result;
    }

    private static void processCommand(String x, String y) {
        if (x.equalsIgnoreCase( anotherString: "N")) setConfig(x, Integer.parseInt(y));
        else
            // TODO sort this out
```

Git | TODO | Problems | Python Packages | Profiler | Terminal | Endpoints | Build | Dependencies | Event Log

Build completed successfully in 1 sec, 37 ms (6 minutes ago) | 23:38 (7 chars) LF UTF-8 4 spaces | vraj-r


demo 2 – PasswordEncoder.java | employee-manager – application.properties | INFO6205 – Main.java

INFO6205 〉 src 〉 main 〉 java 〉 edu 〉 neu 〉 coe 〉 info6205 〉 sort 〉 par 〉 Main 〉 main

Main.java | ParSort.java

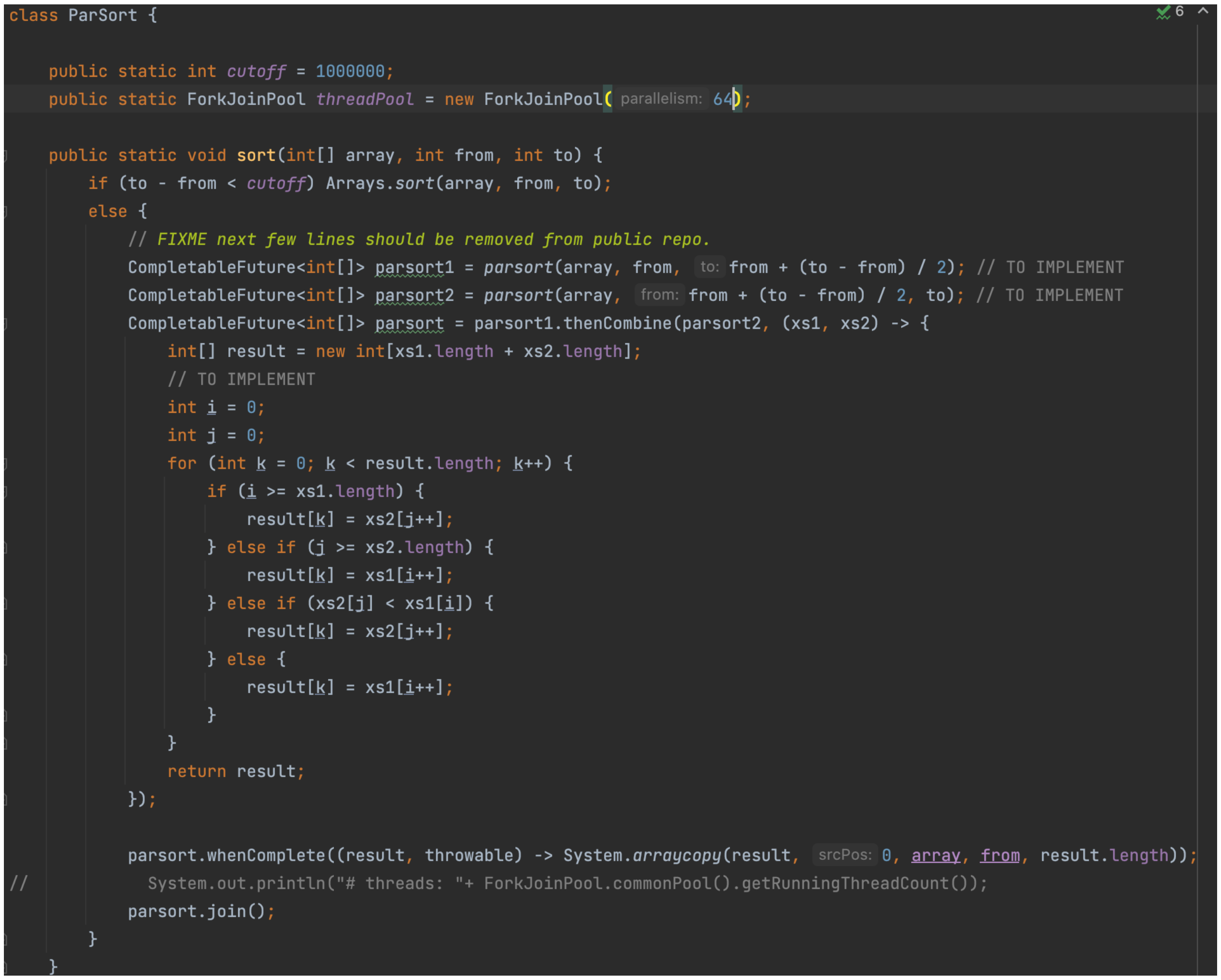
```
greedy
hashtable
lab_1
life
pq
randomwalk
reduction
runLengthEncoding
sort
symbolTable
threesum
union_find
util
BinarySearch
CallByValue
ComparableTuple
Counter
HuffmanCoding
Iteration
Matrix
MyDate
Mystery
NewtonApproximation
RecursionAndIteration
SizedIterable
SizedIterableImpl
Ticket
TicketChecker.java
Tuple
VLA2
resources
test
result.csv
target
.gitignore
LICENSE
pom.xml
External Libraries
Scratches and Consoles
```

```java

    private static void processArgs(String[] args) {
        String[] xs = args;
        while (xs.length > 0)
            if (xs[0].startsWith("-")) xs = processArg(xs);
    }

    private static String[] processArg(String[] xs) {
        String[] result = new String[0];
        System.arraycopy(xs,  srcPos: 2, result,  destPos: 0,  length: xs.length - 2);
        processCommand(xs[0], xs[1]);
        return result;
    }

    private static void processCommand(String x, String y) {
        if (x.equalsIgnoreCase( anotherString: "N")) setConfig(x, Integer.parseInt(y));
        else
            // TODO sort this out
            if (x.equalsIgnoreCase( anotherString: "P")) //noinspection ResultOfMethodCallIgnored
                ForkJoinPool.getCommonPoolParallelism();
    }

    private static void setConfig(String x, int i) { configuration.put(x, i); }

    /MismatchedQueryAndUpdateOfCollection/
    private static final Map<String, Integer> configuration = new HashMap<>();


}
```

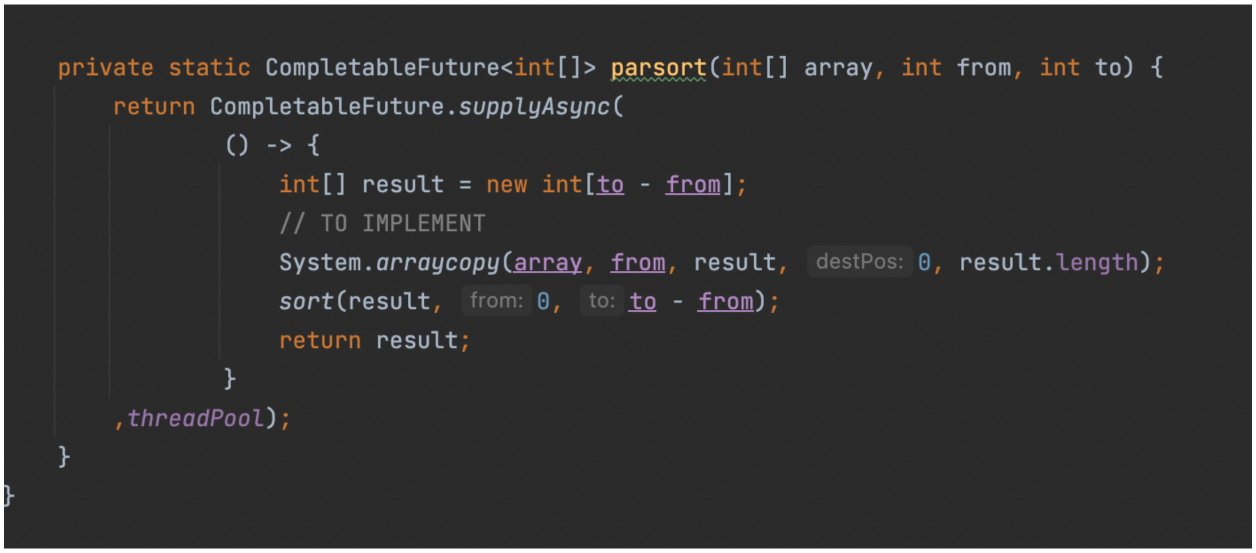
Git | TODO | Problems | Python Packages | Profiler | Terminal | Endpoints | Build | Dependencies | Event Log

Build completed successfully in 1 sec, 37 ms (6 minutes ago) | 23:38 (7 chars) LF UTF-8 4 spaces | vraj-r

File name: Parsort.java

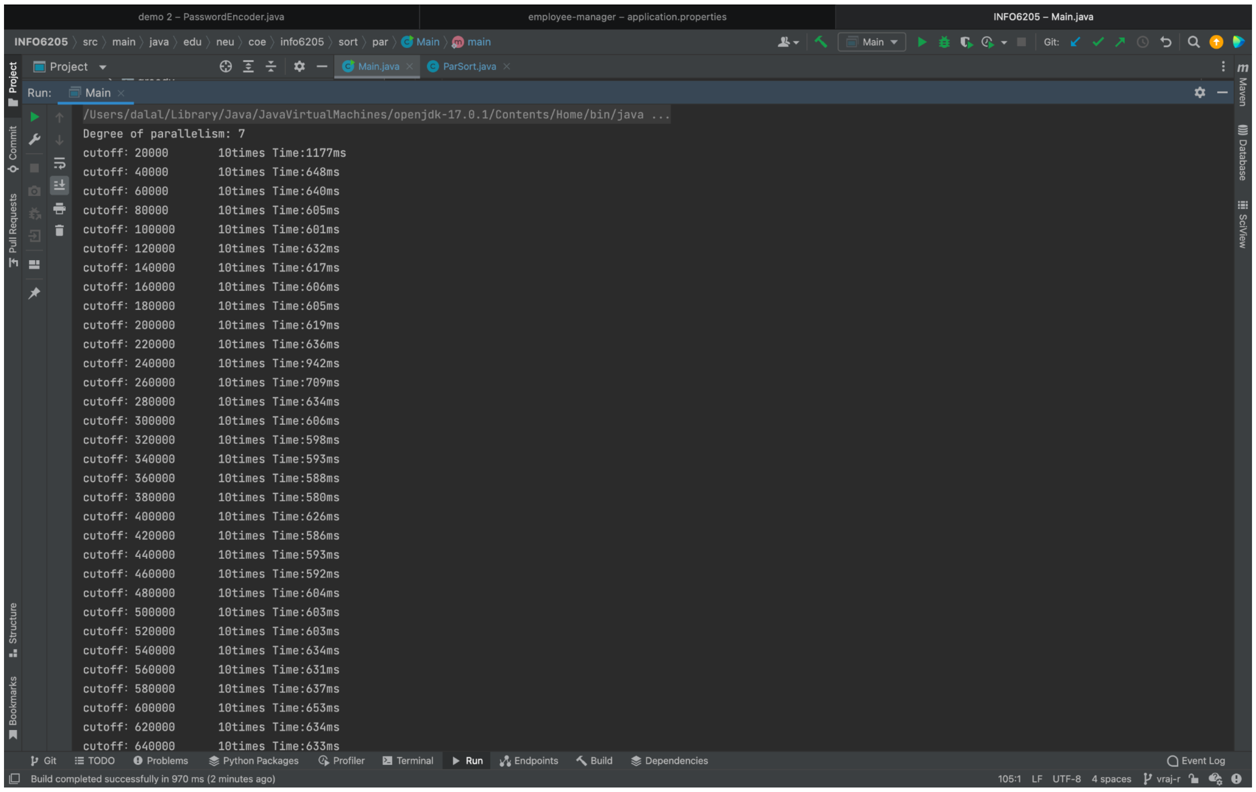


```
class ParSort {

    public static int cutoff = 1000000;
    public static ForkJoinPool threadPool = new ForkJoinPool( parallelism: 64);

    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); // TO IMPLEMENT
            CompletableFuture<int[]> parsort2 = parsort(array,  from: from + (to - from) / 2, to); // TO IMPLEMENT
            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;
            });

            parsort.whenComplete((result, throwable) -> System.arraycopy(result,  srcPos: 0, array, from, result.length));
//              System.out.println("# threads: "+ ForkJoinPool.commonPool().getRunningThreadCount());
            parsort.join();
        }
    }
```



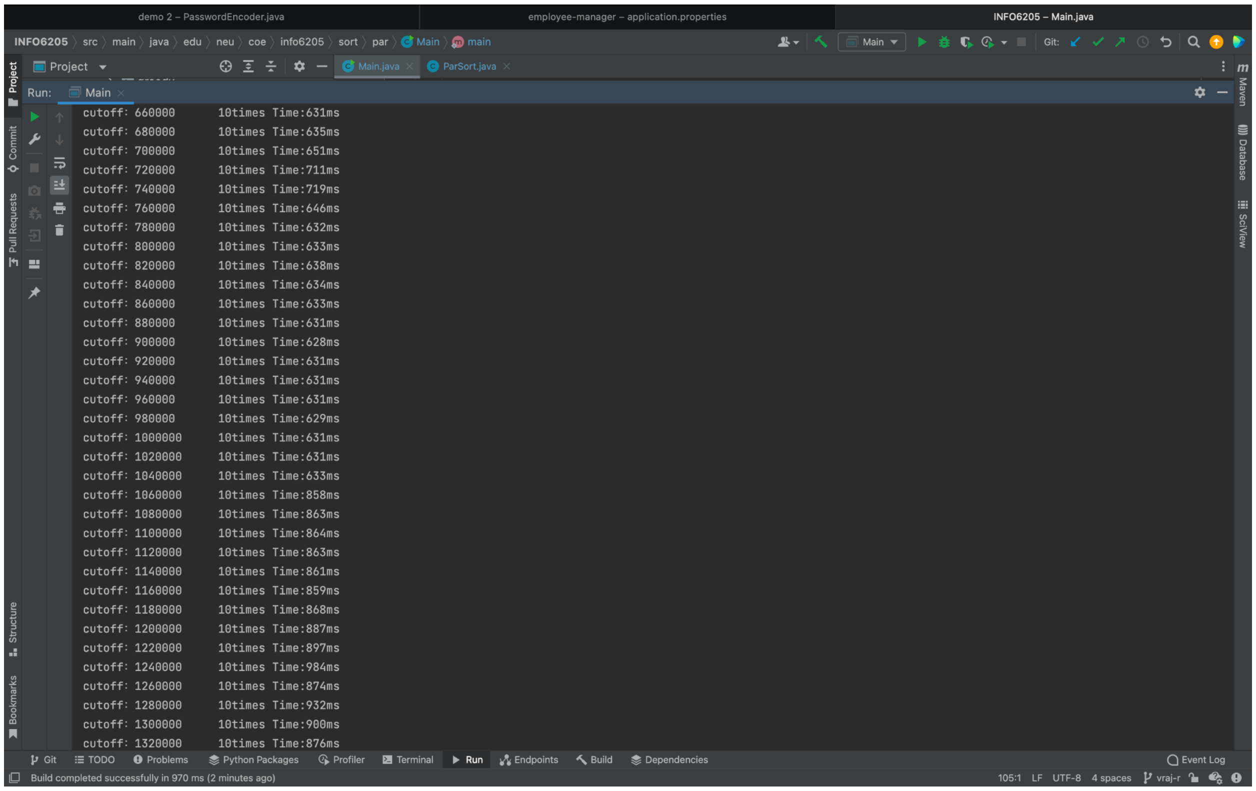
```java
    private static CompletableFuture<int[]> parsort(int[] array, int from, int to) {
        return CompletableFuture.supplyAsync(
                () -> {
                    int[] result = new int[to - from];
                    // TO IMPLEMENT
                    System.arraycopy(array, from, result, destPos: 0, result.length);
                    sort(result, from: 0, to: to - from);
                    return result;
                }
        ,threadPool);
    }
}
```

Output



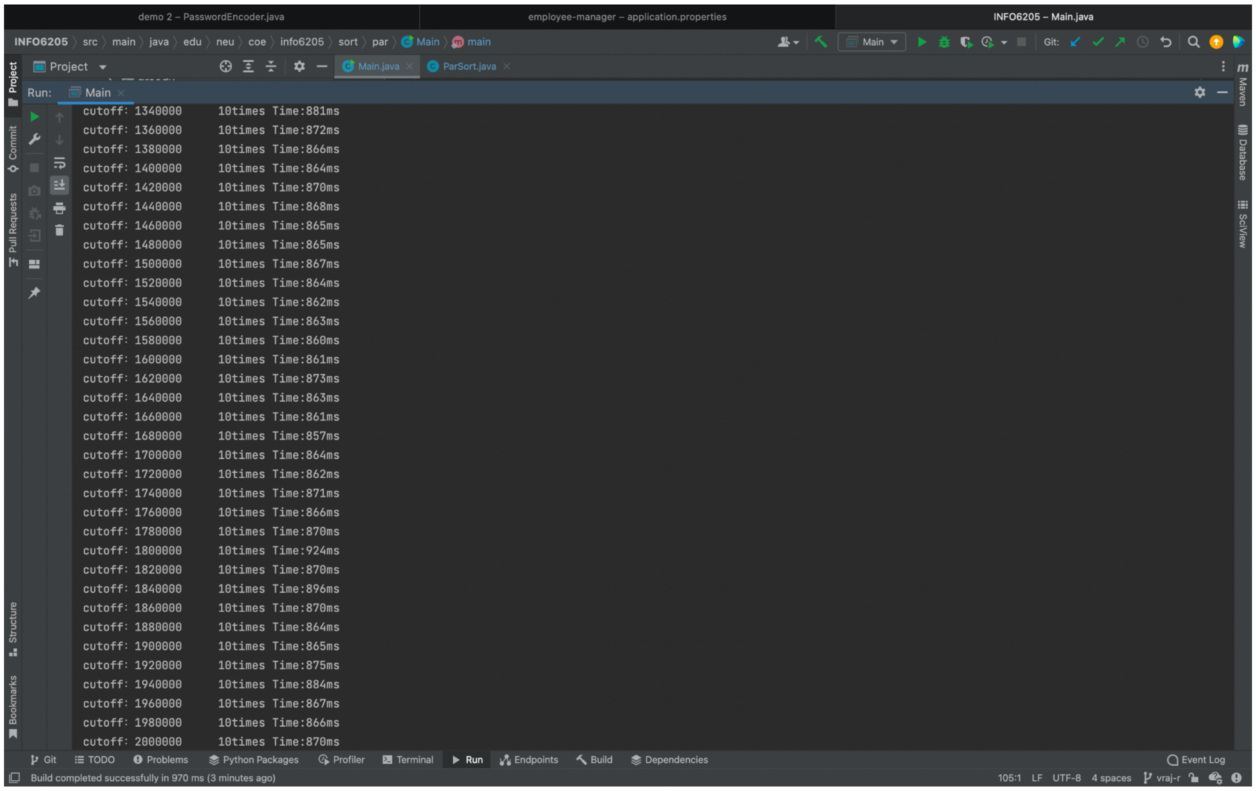
```
/Users/dalal/Library/Java/JavaVirtualMachines/openjdk-17.0.1/Contents/Home/bin/java ...
Degree of parallelism: 7
cutoff: 20000       10times Time:1177ms
cutoff: 40000       10times Time:648ms
cutoff: 60000       10times Time:640ms
cutoff: 80000       10times Time:605ms
cutoff: 100000      10times Time:601ms
cutoff: 120000      10times Time:632ms
cutoff: 140000      10times Time:617ms
cutoff: 160000      10times Time:606ms
cutoff: 180000      10times Time:605ms
cutoff: 200000      10times Time:619ms
cutoff: 220000      10times Time:636ms
cutoff: 240000      10times Time:942ms
cutoff: 260000      10times Time:709ms
cutoff: 280000      10times Time:634ms
cutoff: 300000      10times Time:606ms
cutoff: 320000      10times Time:598ms
cutoff: 340000      10times Time:593ms
cutoff: 360000      10times Time:588ms
cutoff: 380000      10times Time:580ms
cutoff: 400000      10times Time:626ms
cutoff: 420000      10times Time:586ms
cutoff: 440000      10times Time:593ms
cutoff: 460000      10times Time:592ms
cutoff: 480000      10times Time:604ms
cutoff: 500000      10times Time:603ms
cutoff: 520000      10times Time:603ms
cutoff: 540000      10times Time:634ms
cutoff: 560000      10times Time:631ms
cutoff: 580000      10times Time:637ms
cutoff: 600000      10times Time:653ms
cutoff: 620000      10times Time:634ms
cutoff: 640000      10times Time:633ms
```

Build completed successfully in 970 ms (2 minutes ago)



```
cutoff: 660000      10times Time:631ms
cutoff: 680000      10times Time:635ms
cutoff: 700000      10times Time:651ms
cutoff: 720000      10times Time:711ms
cutoff: 740000      10times Time:719ms
cutoff: 760000      10times Time:646ms
cutoff: 780000      10times Time:632ms
cutoff: 800000      10times Time:633ms
cutoff: 820000      10times Time:638ms
cutoff: 840000      10times Time:634ms
cutoff: 860000      10times Time:633ms
cutoff: 880000      10times Time:631ms
cutoff: 900000      10times Time:628ms
cutoff: 920000      10times Time:631ms
cutoff: 940000      10times Time:631ms
cutoff: 960000      10times Time:631ms
cutoff: 980000      10times Time:629ms
cutoff: 1000000     10times Time:631ms
cutoff: 1020000     10times Time:631ms
cutoff: 1040000     10times Time:633ms
cutoff: 1060000     10times Time:858ms
cutoff: 1080000     10times Time:863ms
cutoff: 1100000     10times Time:864ms
cutoff: 1120000     10times Time:863ms
cutoff: 1140000     10times Time:861ms
cutoff: 1160000     10times Time:859ms
cutoff: 1180000     10times Time:868ms
cutoff: 1200000     10times Time:887ms
cutoff: 1220000     10times Time:897ms
cutoff: 1240000     10times Time:984ms
cutoff: 1260000     10times Time:874ms
cutoff: 1280000     10times Time:932ms
cutoff: 1300000     10times Time:900ms
cutoff: 1320000     10times Time:876ms
```

Build completed successfully in 970 ms (2 minutes ago)



```
cutoff: 1340000     10times Time:881ms
cutoff: 1360000     10times Time:872ms
cutoff: 1380000     10times Time:866ms
cutoff: 1400000     10times Time:864ms
cutoff: 1420000     10times Time:870ms
cutoff: 1440000     10times Time:868ms
cutoff: 1460000     10times Time:865ms
cutoff: 1480000     10times Time:865ms
cutoff: 1500000     10times Time:867ms
cutoff: 1520000     10times Time:864ms
cutoff: 1540000     10times Time:862ms
cutoff: 1560000     10times Time:863ms
cutoff: 1580000     10times Time:860ms
cutoff: 1600000     10times Time:861ms
cutoff: 1620000     10times Time:873ms
cutoff: 1640000     10times Time:863ms
cutoff: 1660000     10times Time:861ms
cutoff: 1680000     10times Time:857ms
cutoff: 1700000     10times Time:864ms
cutoff: 1720000     10times Time:862ms
cutoff: 1740000     10times Time:871ms
cutoff: 1760000     10times Time:866ms
cutoff: 1780000     10times Time:870ms
cutoff: 1800000     10times Time:924ms
cutoff: 1820000     10times Time:870ms
cutoff: 1840000     10times Time:896ms
cutoff: 1860000     10times Time:870ms
cutoff: 1880000     10times Time:864ms
cutoff: 1900000     10times Time:865ms
cutoff: 1920000     10times Time:875ms
cutoff: 1940000     10times Time:884ms
cutoff: 1960000     10times Time:867ms
cutoff: 1980000     10times Time:866ms
cutoff: 2000000     10times Time:870ms
```

Build completed successfully in 970 ms (3 minutes ago)

Evidence/graph:

1)For array size= 2000000

Table

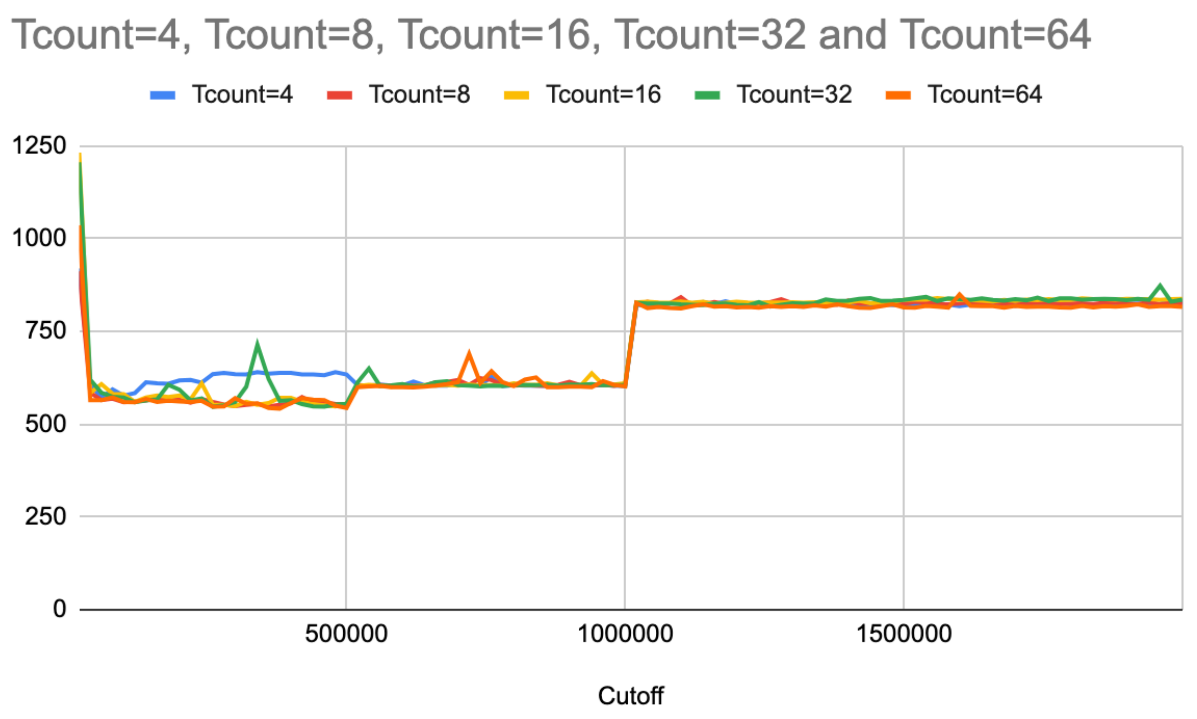
Cutoff	Tcount=4	Tcount=8	Tcount=16	Tcount=32	Tcount=64
20000	920	912	1232	1206	1036
40000	608	585	584	620	565
60000	571	565	608	585	565

80000	594	569	582	574	572
100000	577	559	581	572	560
120000	584	561	560	559	559
140000	613	564	572	564	568
160000	610	570	577	567	560
180000	609	564	573	606	563
200000	618	566	577	593	561
220000	619	558	567	564	560
240000	612	568	610	569	563
260000	635	560	552	547	547
280000	638	553	552	551	548
300000	635	549	549	560	570
320000	634	552	560	601	555
340000	640	555	552	714	556
360000	636	548	557	622	544
380000	638	552	571	563	542
400000	638	556	571	565	557
420000	634	573	561	554	570
440000	634	558	554	548	566
460000	632	560	551	547	565
480000	640	554	548	553	550
500000	634	554	556	555	544
520000	605	606	603	613	600
540000	604	603	606	650	602
560000	607	605	604	603	603
580000	604	604	601	605	600
600000	604	606	606	608	600
620000	615	604	602	602	599
640000	605	604	605	604	601
660000	604	613	604	613	604
680000	604	614	606	616	608
700000	610	619	604	605	613
720000	606	606	607	604	689
740000	605	624	605	602	611
760000	628	619	605	604	643

780000	603	605	604	603	613
800000	605	604	610	605	603
820000	604	605	606	605	620
840000	604	604	606	605	626
860000	603	603	610	607	600
880000	605	606	604	604	600
900000	606	614	605	607	601
920000	603	605	604	606	601
940000	604	604	637	608	600
960000	605	607	607	605	616
980000	604	603	607	605	606
1000000	603	608	610	605	602
1020000	821	823	828	827	827
1040000	822	830	830	825	813
1060000	822	823	828	825	816
1080000	825	825	827	825	813
1100000	828	842	831	823	812
1120000	825	820	827	822	818
1140000	829	823	830	821	823
1160000	823	829	824	825	817
1180000	831	824	827	825	818
1200000	824	827	830	822	815
1220000	824	824	828	820	816
1240000	827	825	827	829	814
1260000	821	829	828	817	818
1280000	827	837	828	822	816
1300000	825	826	829	826	818
1320000	821	826	827	825	816
1340000	828	824	829	826	820
1360000	824	824	829	836	817
1380000	829	822	828	832	823
1400000	819	833	828	833	818
1420000	823	825	833	838	814
1440000	820	825	826	840	813
1460000	824	825	828	832	818

1480000	821	825	830	833	823
1500000	822	824	835	835	815
1520000	822	830	836	839	814
1540000	825	828	836	843	819
1560000	826	824	840	832	817
1580000	822	823	837	839	814
1600000	818	825	835	837	850
1620000	822	829	834	835	819
1640000	825	824	833	839	818
1660000	822	830	831	835	818
1680000	827	824	835	833	814
1700000	819	830	832	837	819
1720000	824	824	836	834	816
1740000	821	828	834	841	817
1760000	823	826	837	832	817
1780000	827	823	834	839	815
1800000	823	825	835	839	814
1820000	821	826	839	836	819
1840000	819	824	837	837	815
1860000	824	827	837	838	818
1880000	825	829	836	837	817
1900000	824	825	838	834	819
1920000	825	823	837	838	823
1940000	824	829	837	835	816
1960000	824	823	835	873	818
1980000	836	828	837	830	818
2000000	826	825	838	836	816

Graph



2)For array size = 2050000

Table

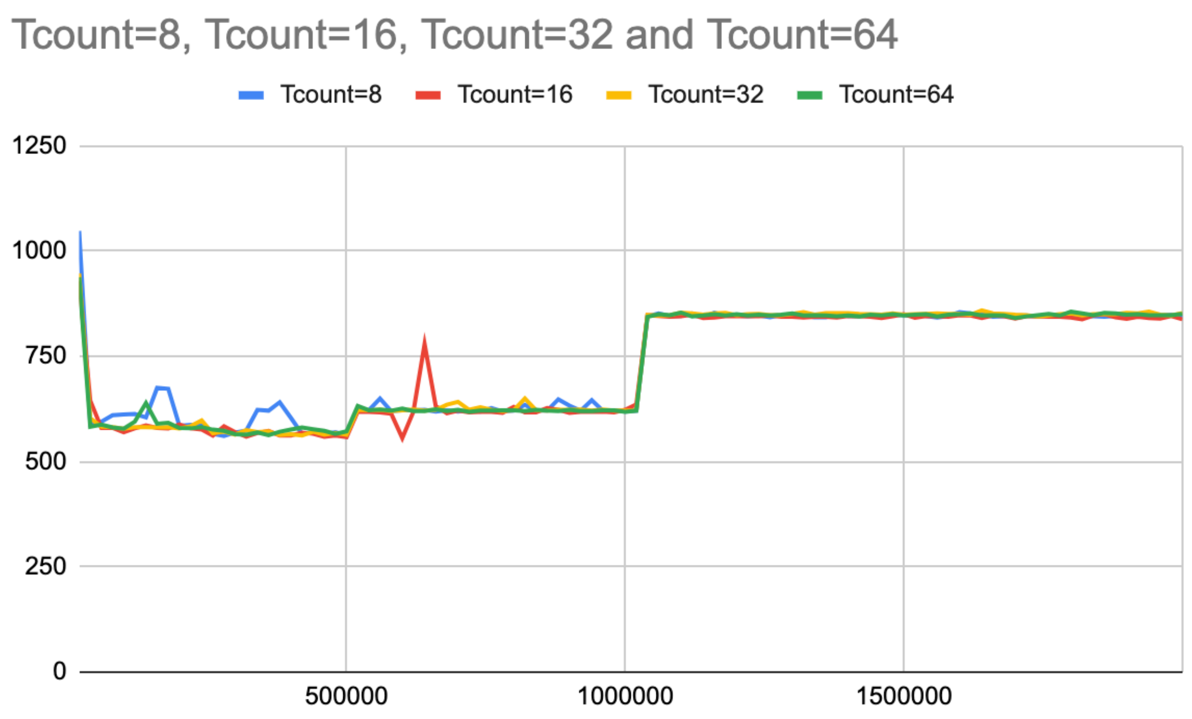
Cutoff	Tcount=4	Tcount=8	Tcount=16	Tcount=32	Tcount=64
20000	1045	1048	929	947	938
40000	607	588	645	603	583
60000	609	595	580	583	588
80000	590	610	580	582	581
100000	595	612	570	578	578
120000	589	613	579	582	595
140000	619	605	586	582	639
160000	627	675	580	581	590
180000	623	673	579	582	592
200000	647	585	588	579	580
220000	621	587	579	584	579

240000	625	586	576	598	582
260000	646	566	562	567	576
280000	645	561	584	571	573
300000	640	569	570	564	565
320000	656	574	559	575	564
340000	654	623	568	570	569
360000	646	621	573	572	563
380000	646	641	562	563	571
400000	642	604	562	565	576
420000	650	566	569	562	581
440000	645	570	566	571	577
460000	646	565	559	564	573
480000	643	570	562	568	566
500000	644	562	558	564	572
520000	617	622	619	623	632
540000	618	622	618	622	622
560000	616	650	617	623	624
580000	617	620	614	620	621
600000	618	622	556	622	626
620000	616	622	619	624	620
640000	615	623	779	621	620
660000	625	619	632	623	624
680000	618	622	615	635	620
700000	617	619	621	642	623
720000	615	622	617	623	618
740000	616	619	618	629	621
760000	618	627	618	623	621
780000	616	619	616	620	622
800000	615	621	630	622	622
820000	617	635	617	650	620
840000	618	621	617	622	623
860000	617	621	627	621	621
880000	617	648	623	623	620
900000	618	633	616	623	623
920000	617	621	618	624	620

940000	617	646	618	621	620
960000	617	621	619	624	622
980000	625	622	617	621	622
1000000	641	619	623	621	618
1020000	620	621	636	622	621
1040000	839	845	848	848	844
1060000	841	852	846	847	850
1080000	846	846	844	848	848
1100000	842	846	845	854	854
1120000	842	852	849	852	845
1140000	846	844	841	849	847
1160000	845	854	842	852	850
1180000	841	847	845	853	847
1200000	845	847	845	848	850
1220000	843	845	845	850	847
1240000	842	847	845	851	849
1260000	845	843	847	848	847
1280000	845	848	844	848	849
1300000	840	848	844	851	852
1320000	841	847	842	855	847
1340000	843	843	844	849	848
1360000	842	843	844	853	847
1380000	841	850	842	853	845
1400000	835	846	846	853	847
1420000	842	846	845	851	845
1440000	841	848	844	850	849
1460000	841	845	841	849	847
1480000	838	849	845	852	850
1500000	841	848	850	849	847
1520000	842	847	842	850	849
1540000	844	846	846	851	850
1560000	867	842	845	852	845
1580000	840	846	844	851	849
1600000	842	855	848	851	850
1620000	844	852	847	849	852

1640000	842	850	841	859	847
1660000	838	844	847	852	848
1680000	843	845	845	851	847
1700000	840	847	840	849	841
1720000	840	847	845	848	845
1740000	842	844	845	845	848
1760000	845	847	844	847	851
1780000	840	848	844	850	847
1800000	845	848	842	852	856
1820000	840	845	838	847	852
1840000	841	845	847	850	848
1860000	842	844	850	850	853
1880000	843	846	843	850	852
1900000	839	849	839	853	849
1920000	852	847	844	852	850
1940000	844	845	841	856	848
1960000	856	847	840	849	848
1980000	838	849	846	847	848
2000000	847	845	838	853	851

Graph



3) For array size = 2100000

Table

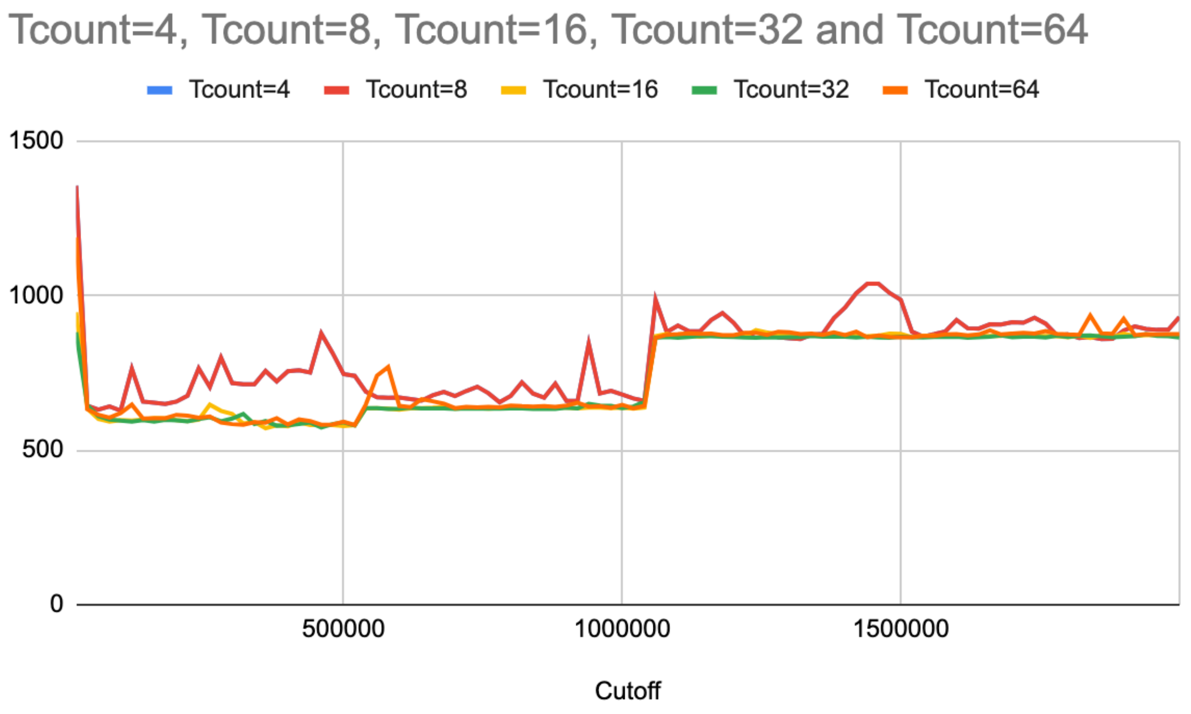
Cutoff	Tcount=4	Tcount=8	Tcount=16	Tcount=32	Tcount=64
20000	1355	1355	946	881	1188
40000	644	644	634	645	632
60000	631	631	601	610	614
80000	641	641	592	599	605
100000	628	628	597	595	620
120000	763	763	595	592	647
140000	657	657	599	597	602
160000	653	653	604	592	603
180000	649	649	603	598	603
200000	657	657	597	596	614
220000	676	676	594	593	611
240000	764	764	598	600	605

260000	704	704	647	606	609
280000	799	799	627	593	589
300000	717	717	617	602	584
320000	713	713	586	617	582
340000	713	713	591	584	590
360000	756	756	571	594	589
380000	722	722	580	579	603
400000	755	755	577	580	583
420000	758	758	590	584	599
440000	751	751	581	589	594
460000	877	877	581	573	582
480000	815	815	581	584	583
500000	746	746	578	590	592
520000	740	740	581	580	581
540000	689	689	635	636	648
560000	671	671	636	636	741
580000	669	669	634	633	769
600000	670	670	631	634	643
620000	665	665	635	637	639
640000	660	660	636	635	665
660000	677	677	637	635	658
680000	688	688	639	635	650
700000	675	675	634	633	636
720000	691	691	636	635	640
740000	705	705	636	634	638
760000	684	684	634	634	640
780000	655	655	636	634	638
800000	675	675	637	635	644
820000	719	719	636	635	642
840000	683	683	637	633	641
860000	670	670	638	634	642
880000	715	715	637	633	640
900000	659	659	637	637	644
920000	660	660	635	635	653
940000	844	844	637	650	641

960000	683	683	637	644	642
980000	692	692	636	643	637
1000000	680	680	637	636	646
1020000	667	667	634	641	635
1040000	659	659	638	656	643
1060000	986	986	869	863	863
1080000	882	882	876	866	873
1100000	903	903	869	863	875
1120000	884	884	869	866	878
1140000	885	885	867	869	876
1160000	921	921	871	868	877
1180000	943	943	870	867	872
1200000	912	912	870	866	872
1220000	868	868	868	864	879
1240000	881	881	888	863	879
1260000	879	879	880	865	873
1280000	865	865	876	864	883
1300000	862	862	869	865	881
1320000	859	859	872	866	875
1340000	873	873	874	869	877
1360000	876	876	869	867	873
1380000	927	927	871	867	881
1400000	961	961	869	867	872
1420000	1007	1007	871	864	883
1440000	1038	1038	868	867	865
1460000	1038	1038	870	864	871
1480000	1008	1008	877	863	866
1500000	986	986	876	867	865
1520000	883	883	865	865	865
1540000	867	867	869	864	869
1560000	875	875	871	866	869
1580000	884	884	868	866	874
1600000	921	921	870	867	875
1620000	894	894	870	863	871
1640000	893	893	868	865	874

1660000	907	907	869	867	887
1680000	906	906	869	871	874
1700000	914	914	877	865	877
1720000	912	912	871	867	879
1740000	928	928	869	867	877
1760000	909	909	869	864	885
1780000	872	872	868	870	876
1800000	874	874	865	866	874
1820000	863	863	870	870	873
1840000	866	866	863	871	935
1860000	859	859	872	868	878
1880000	860	860	866	865	876
1900000	887	887	877	867	925
1920000	900	900	869	868	872
1940000	892	892	872	875	875
1960000	889	889	873	869	874
1980000	889	889	872	868	875
2000000	930	930	871	864	875

Graph



Conclusion:

1) Parallel sort performance does not vary much if thread count ***>= 4*** (greater than or equal to 4).
2) The performance is best when Cutoffs is below ***50%*** of the array size. For Cutoffs greater than or equal to ***50%*** of the array size the performance degrades and remains constant. The performance time degrades by approximately ***~ 220ms.***