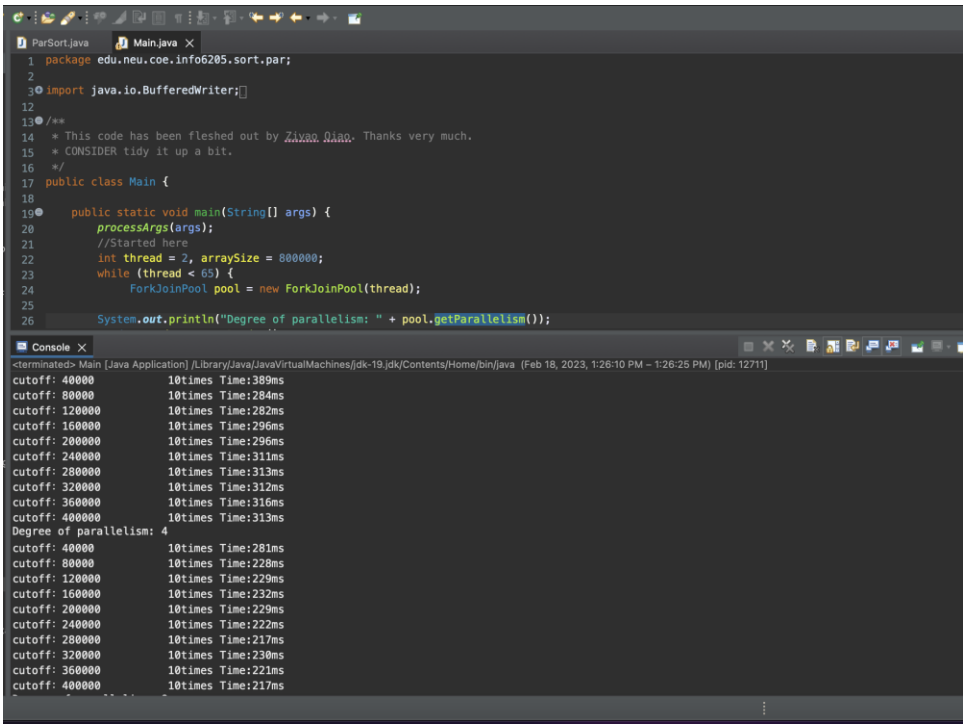
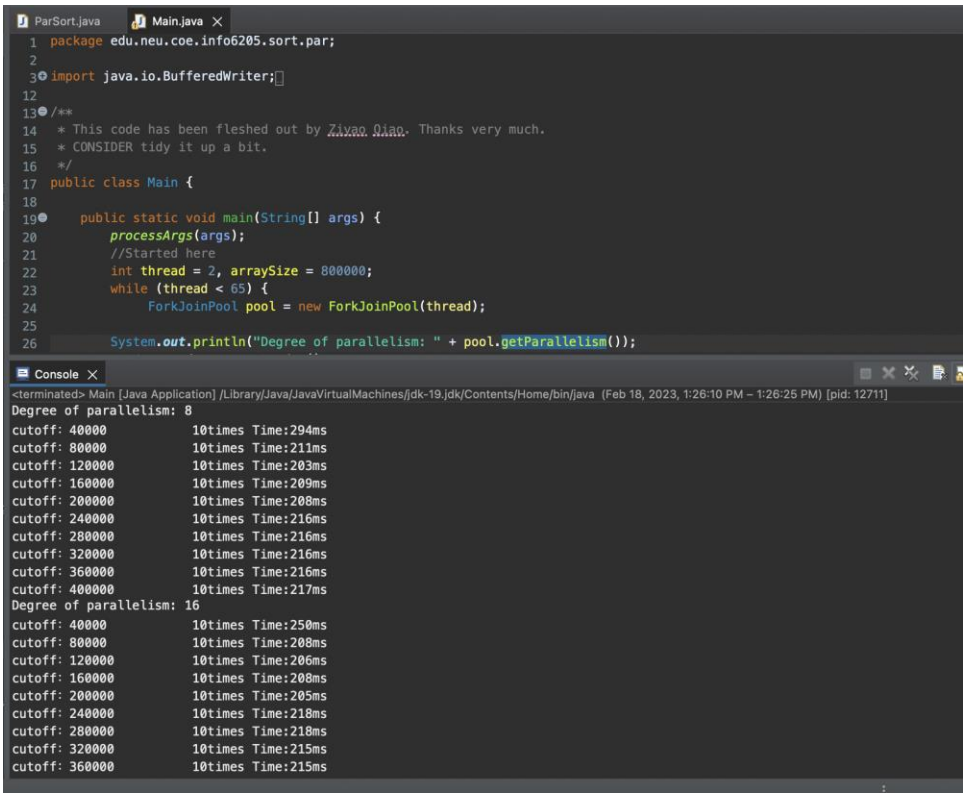
Assignment 5 (Parallel Sorting)

Screenshots of output:

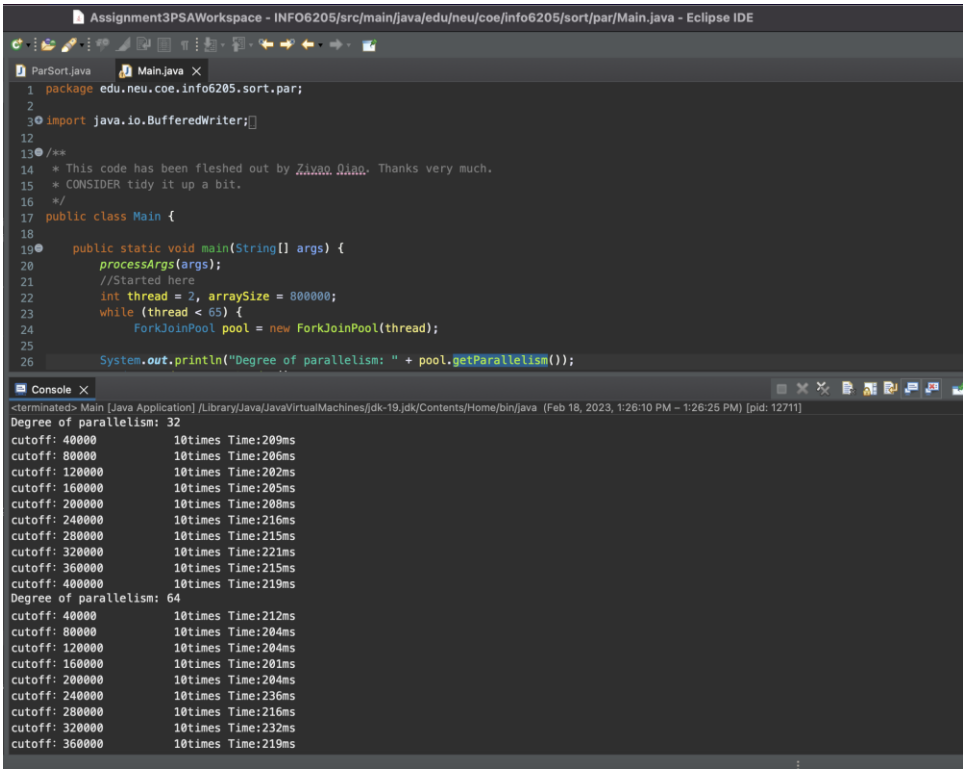
The screenshot of the terminal output with increasing Degree of parallelism and respective cutoff values:





```
ParSort.java   Main.java
 1 package edu.neu.coe.info6205.sort.par;
 2
 3 import java.io.BufferedWriter;
12
13 /**
14  * This code has been fleshed out by Zivao Qiao. Thanks very much.
15  * CONSIDER tidy it up a bit.
16  */
17 public class Main {
18
19     public static void main(String[] args) {
20         processArgs(args);
21         //Started here
22         int thread = 2, arraySize = 800000;
23         while (thread < 65) {
24             ForkJoinPool pool = new ForkJoinPool(thread);
25
26         System.out.println("Degree of parallelism: " + pool.getParallelism());

Console
<terminated> Main [Java Application] /Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java  (Feb 18, 2023, 1:26:10 PM – 1:26:25 PM) [pid: 12711]
Degree of parallelism: 8
cutoff: 40000           10times Time:294ms
cutoff: 80000           10times Time:211ms
cutoff: 120000          10times Time:203ms
cutoff: 160000          10times Time:209ms
cutoff: 200000          10times Time:208ms
cutoff: 240000          10times Time:216ms
cutoff: 280000          10times Time:216ms
cutoff: 320000          10times Time:216ms
cutoff: 360000          10times Time:216ms
cutoff: 400000          10times Time:217ms
Degree of parallelism: 16
cutoff: 40000           10times Time:250ms
cutoff: 80000           10times Time:208ms
cutoff: 120000          10times Time:206ms
cutoff: 160000          10times Time:208ms
cutoff: 200000          10times Time:205ms
cutoff: 240000          10times Time:218ms
cutoff: 280000          10times Time:218ms
cutoff: 320000          10times Time:215ms
cutoff: 360000          10times Time:215ms
```



```
Assignment3PSAWorkspace - INFO6205/src/main/java/edu/neu/coe/info6205/sort/par/Main.java - Eclipse IDE
ParSort.java   Main.java
 1 package edu.neu.coe.info6205.sort.par;
 2
 3 import java.io.BufferedWriter;
12
13 /**
14  * This code has been fleshed out by Zivao Qiao. Thanks very much.
15  * CONSIDER tidy it up a bit.
16  */
17 public class Main {
18
19     public static void main(String[] args) {
20         processArgs(args);
21         //Started here
22         int thread = 2, arraySize = 800000;
23         while (thread < 65) {
24             ForkJoinPool pool = new ForkJoinPool(thread);
25
26         System.out.println("Degree of parallelism: " + pool.getParallelism());

Console
<terminated> Main [Java Application] /Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java  (Feb 18, 2023, 1:26:10 PM – 1:26:25 PM) [pid: 12711]
Degree of parallelism: 32
cutoff: 40000           10times Time:209ms
cutoff: 80000           10times Time:206ms
cutoff: 120000          10times Time:202ms
cutoff: 160000          10times Time:205ms
cutoff: 200000          10times Time:208ms
cutoff: 240000          10times Time:216ms
cutoff: 280000          10times Time:215ms
cutoff: 320000          10times Time:221ms
cutoff: 360000          10times Time:215ms
cutoff: 400000          10times Time:219ms
Degree of parallelism: 64
cutoff: 40000           10times Time:212ms
cutoff: 80000           10times Time:204ms
cutoff: 120000          10times Time:204ms
cutoff: 160000          10times Time:201ms
cutoff: 200000          10times Time:204ms
cutoff: 240000          10times Time:236ms
cutoff: 280000          10times Time:216ms
cutoff: 320000          10times Time:232ms
cutoff: 360000          10times Time:219ms
```

INFO – 6205 PROGRAM STRUCTURES AND ALGORITHMS

When array size and cutoff values are different :

Array size: 500000 and CutOff: 4000

Cutoff	**4000**	**8000**	**12000**	**16000**	**20000**	**24000**	**28000**	**32000**	**36000**	**40000**
2	27	21.9	17.2	16.6	15.9	15.9	15.9	16.5	16.6	16.6
4	16.7	13.5	13.6	13.4	13.2	13.4	13.5	13.6	14	13.7
8	15.3	12.7	13.2	12.3	12.5	12.8	12.4	12.3	12.5	12.6
16	14.7	12.5	13	12.2	19.1	13	12.5	12.9	12.6	12.8
32	15.3	15.4	13.2	12.8	12.6	12.6	12.2	12.6	12.2	12.6
64	14.6	12.5	12.5	12.5	12.6	12.4	12.5	12.3	12.5	12.3

Array size: 800000 and CutOff: 10000

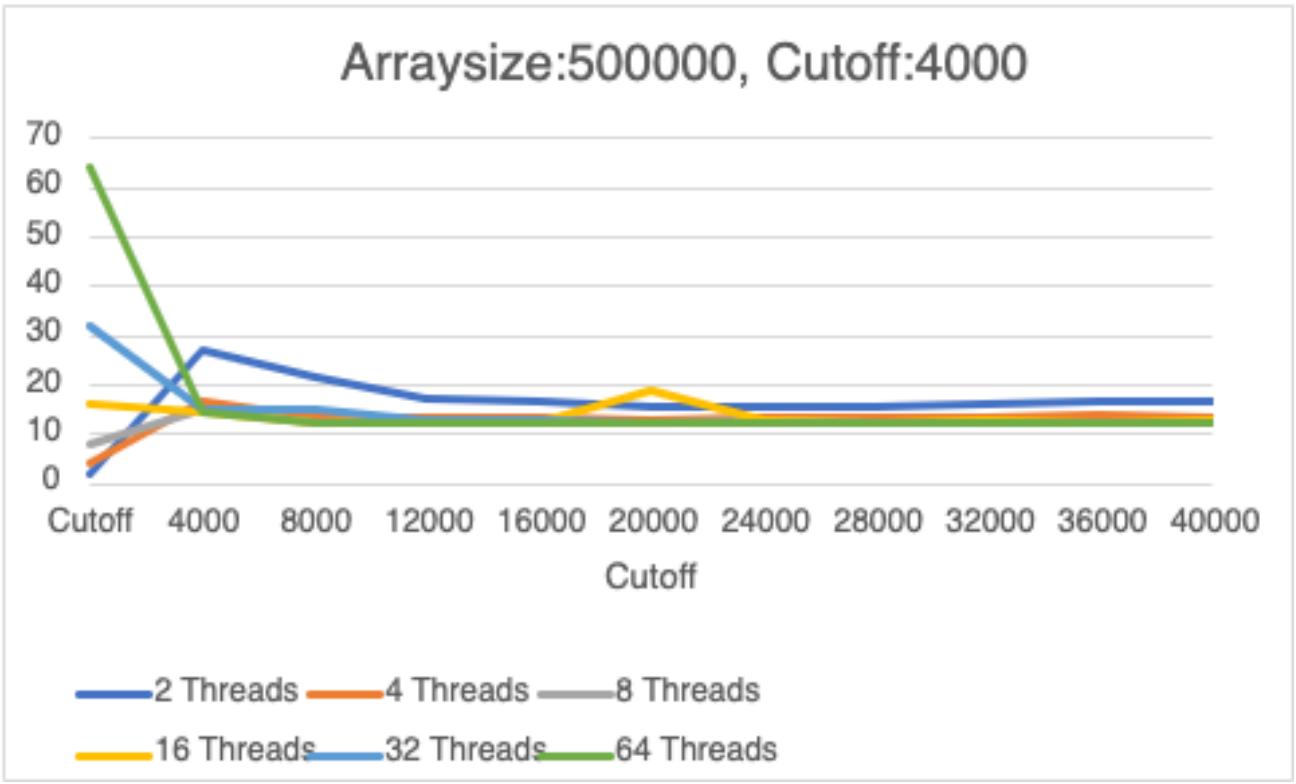
Cutoff	**10000**	**20000**	**30000**	**40000**	**50000**	**60000**	**70000**	**80000**	**90000**	**100000**
2	47.8	26.7	28	26.2	27.4	27.4	27.5	27.9	27	26.3
4	28.1	23.1	24.7	25.8	23.3	23.3	22.4	22.4	22.2	26.5
8	28.5	26.5	22.2	21.3	20.2	21.4	20.2	20.5	20.1	20.8
16	26.3	24	21.5	20.8	20.2	30.7	23.2	23.4	26.3	22.5
32	32	27.2	21.8	21.9	20.5	20.5	21	23.5	25.7	21.9
64	27.7	23.1	23.4	23.8	22.8	21.9	24.7	20.7	20.5	20.7

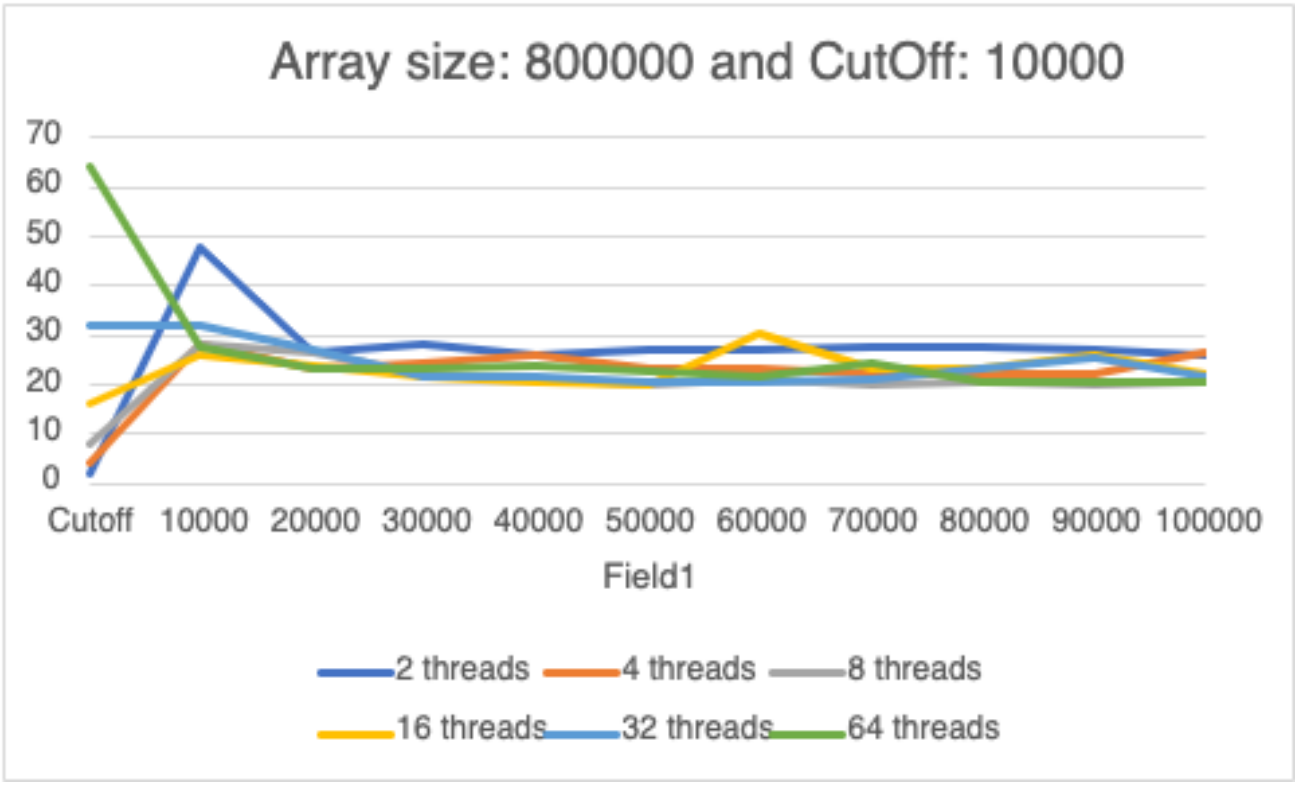
Array size: 110000 and CutOff: 20000

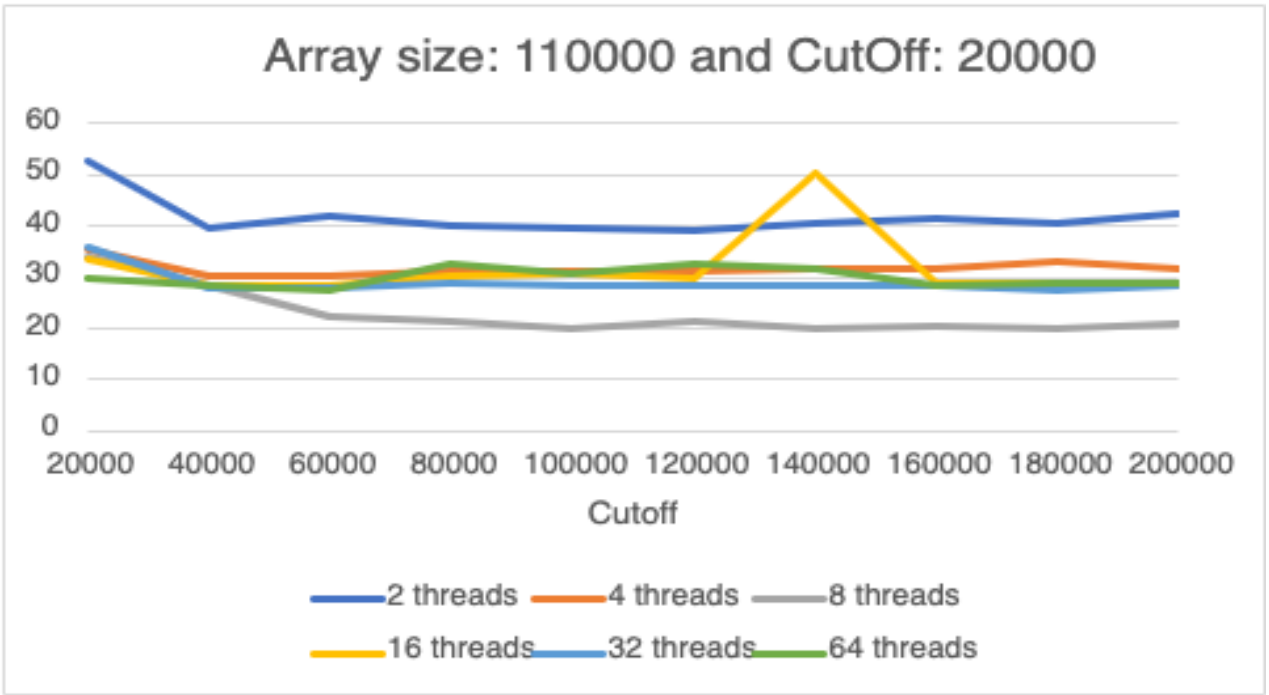
Cutoff	**20000**	**40000**	**60000**	**80000**	**100000**	**120000**	**140000**	**160000**	**180000**	**200000**
2	52.7	39.7	41.9	40.1	39.6	39	40.4	41.7	40.5	42.4
4	35.5	30.5	30.2	31.2	31.2	31	31.7	31.7	33.2	31.7
8	34.1	28.6	28.2	28.8	28.7	28.7	28.2	28.7	28.9	28.3
16	33.5	28.5	28.6	30.4	30.8	29.8	50.5	28.9	28.9	28.3
32	36.1	28.1	27.9	28.9	28.4	28.6	28.3	28.4	27.6	28.2
64	29.8	28.2	27.7	32.6	30.9	32.7	31.5	28.2	29	28.8

In the above table representations, the cut off values are represented in the left column, the cutoff values are represented in top rows and the table contains the time with respect to varied array size.

Graph representation:




Array size: 800000 and CutOff: 10000
70
60
50
40
30
20
10
0
Cutoff 10000 20000 30000 40000 50000 60000 70000 80000 90000 100000
Field1
2 threads
4 threads
8 threads
16 threads
32 threads
64 threads


Array size: 110000 and CutOff: 20000
60
50
40
30
20
10
0
20000 40000 60000 80000 100000 120000 140000 160000 180000 200000
Cutoff
2 threads
4 threads
8 threads
16 threads
32 threads
64 threads

Observations:

After analyzing the graphs and data presented in the tables, we can draw the following conclusions:

Adjusting the cutoff values and number of threads for different array sizes, we observed that increasing the number of threads beyond 8 does not enhance the algorithm's performance. Therefore, utilizing 8 threads is the optimal choice.

Based on the above graphs, we can infer that the algorithm's peak performance occurs at approximately 25% of the array size. As a result, we can deduce that this leads to the minimum amount of time required to execute the algorithm.