

Program Structures & Algorithms

Fall 2021

Assignment No. 5

- ◎ Task (List down the tasks performed in the Assignment)
 1. Experiment and come up with a good value for cutoff.
 2. Decide on an ideal number of separate threads and arrange for that number of partitions to be parallelized.
 3. Prepare a report that shows the results of experiments and draw a conclusion (or more) about the efficacy of this method of parallelizing sort.
 4. Analyze and summarize the experimental data, and get the relationship between cutoff numbers, array's length and the number of threads.
- ◎ Relationship Conclusion: (For ex : $z = a * b$)
 - I. The relationship between the best cutoff number, the array's length and the number of threads is:
$$\text{the best cutoff number} = \frac{\text{arrays.length}}{\text{threadNumber}}$$
 - II. The ParSort always does best (compared by the times the best cutoff numbers cost) when there are 4 threads.
 - III. It is not that the more threads, the faster the ParSort speed.

Degree of parallelism: 2

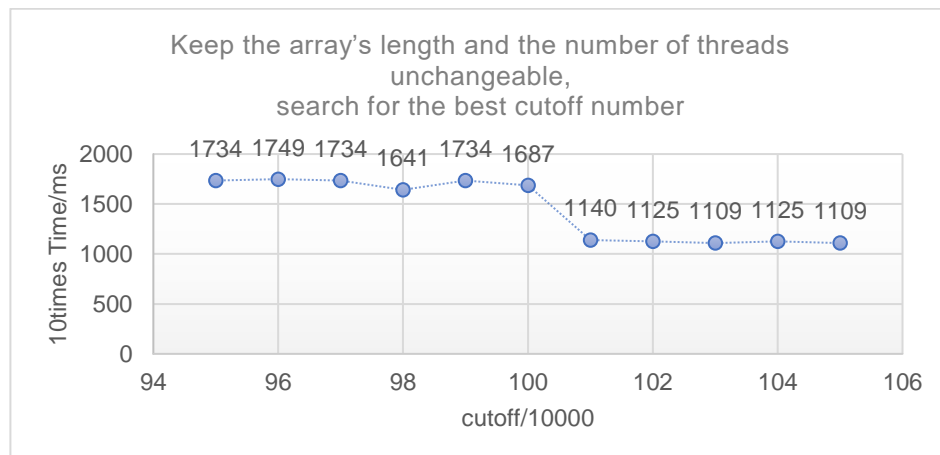
cutoff: 950000	10times Time:1734ms
cutoff: 960000	10times Time:1749ms
cutoff: 970000	10times Time:1734ms
cutoff: 980000	10times Time:1641ms
cutoff: 990000	10times Time:1734ms
cutoff: 1000000	10times Time:1687ms
cutoff: 1010000	10times Time:1140ms
cutoff: 1020000	10times Time:1125ms
cutoff: 1030000	10times Time:1109ms
cutoff: 1040000	10times Time:1125ms
cutoff: 1050000	10times Time:1109ms

It seems that when the cutoff number is 1,010,000, the time significantly shortened, compared to the cutoff number is 1,000,000.

At which time, $\frac{arrays.length}{threadNumber} = \frac{2000000}{2} = 1,000,000$.

Simulation:

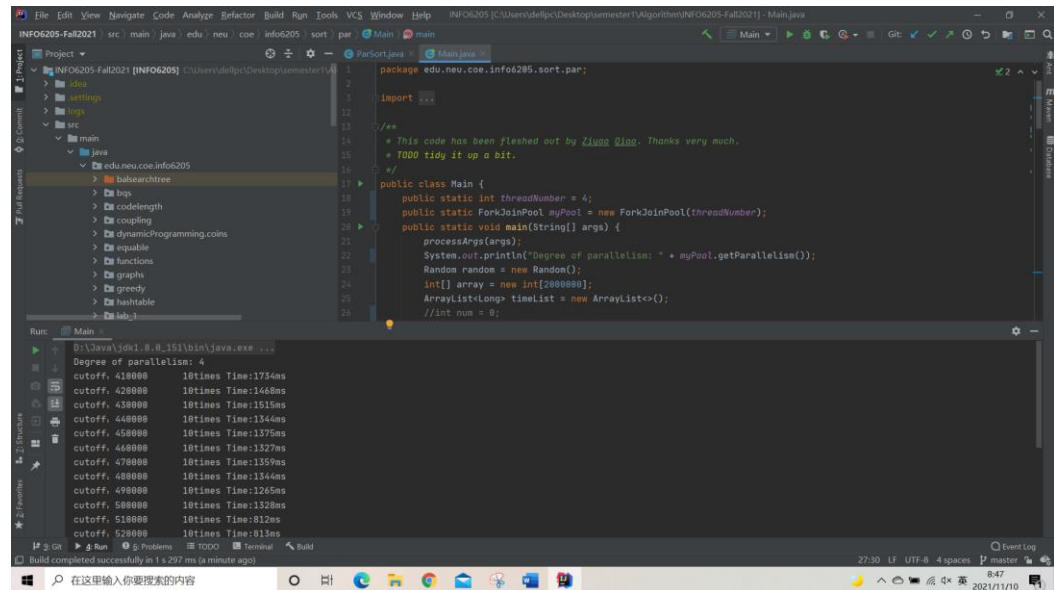
cutoff/10000	10times Time/ms
95	1734
96	1749
97	1734
98	1641
99	1734
100	1687
101	1140
102	1125
103	1109
104	1125
105	1109



It seems that if the cutoff number is bigger than 1,000,000, which is equals to $\frac{arrays.length}{threadNumber}$, the 10times Time drops a lot.

- ② Keep the array's length unchangeable, change the number of threads, search for the best cutoff number.

(1)Output when threadNumber = 4 and array's length = 2,000,000



```
package edu.neu.coe.info6205.sort.par;

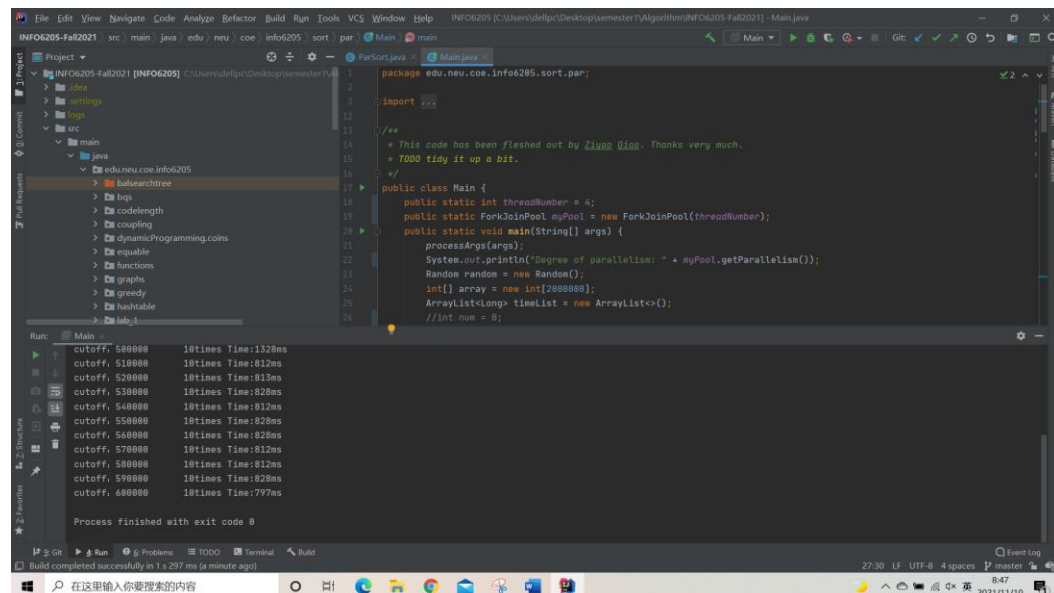
import ...

/**
 * This code has been fleshed out by Ziyao Qiao. Thanks very much.
 * TODO tidy it up a bit.
 */

public class Main {
    public static int threadNumber = 4;
    public static ForkJoinPool myPool = new ForkJoinPool(threadNumber);
    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " + myPool.getParallelism());
        Random random = new Random();
        int[] array = new int[2000000];
        ArrayList<Long> timeList = new ArrayList<>();
        //int num = 0;
    }
}
```

Run: Main

cutoff	10times Time
410000	1734ms
420000	1468ms
430000	1515ms
440000	1344ms
450000	1375ms
460000	1327ms
470000	1359ms
480000	1344ms
490000	1265ms
500000	1128ms
510000	812ms
520000	813ms



```
package edu.neu.coe.info6205.sort.par;

import ...

/**
 * This code has been fleshed out by Ziyao Qiao. Thanks very much.
 * TODO tidy it up a bit.
 */

public class Main {
    public static int threadNumber = 4;
    public static ForkJoinPool myPool = new ForkJoinPool(threadNumber);
    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " + myPool.getParallelism());
        Random random = new Random();
        int[] array = new int[2000000];
        ArrayList<Long> timeList = new ArrayList<>();
        //int num = 0;
    }
}
```

Run: Main

cutoff	10times Time
500000	1328ms
510000	812ms
520000	813ms
530000	828ms
540000	812ms
550000	828ms
560000	828ms
570000	812ms
580000	812ms
590000	828ms
600000	797ms

Process finished with exit code 0

Output text:

Degree of parallelism: 4

cutoff: 450000	10times Time:1375ms
cutoff: 460000	10times Time:1327ms
cutoff: 470000	10times Time:1359ms
cutoff: 480000	10times Time:1344ms
cutoff: 490000	10times Time:1265ms
cutoff: 500000	10times Time:1328ms
cutoff: 510000	10times Time:812ms
cutoff: 520000	10times Time:813ms
cutoff: 530000	10times Time:828ms
cutoff: 540000	10times Time:812ms
cutoff: 550000	10times Time:828ms

We could find that in this case, time drops a lot after the cutoff is more than 500,000.

$$\text{At which time, } \frac{\text{arrays.length}}{\text{threadNumber}} = \frac{2000000}{4} = 500,000 .$$

(2)Output when threadNumber = 8 and array's length = 2,000,000

```

public static int threadNumber = 8;
public static ForkJoinPool myPool = new ForkJoinPool(threadNumber);
public static void main(String[] args) {
    processArgs(args);
    System.out.println("Degree of parallelism: " + myPool.getParallelism());
    Random random = new Random();
    int[] array = new int[2000000];
    ArrayList<Long> timeList = new ArrayList<>();
    //int num = 8;
    for (int i = 20; i <= 30; i++) {
        ParSort.cutoff = 10000 * (i + 1);
        //num++;
        // 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(bound: 200000000);
            ParSort.sort(array, from: 0, array.length);
        }
    }
}

```

```

Run: Main
Degree of parallelism: 8
cutoff, 210000    10times Time:1734ms
cutoff, 220000    10times Time:1344ms
cutoff, 230000    10times Time:1296ms
cutoff, 240000    10times Time:1359ms
cutoff, 250000    10times Time:1328ms
cutoff, 260000    10times Time:891ms
cutoff, 270000    10times Time:874ms
cutoff, 280000    10times Time:891ms
cutoff, 290000    10times Time:898ms
cutoff, 300000    10times Time:844ms
Process finished with exit code 0

```

Output text:

Degree of parallelism: 8	
cutoff: 210000	10times Time:1734ms

cutoff: 220000	10times Time:1344ms
cutoff: 230000	10times Time:1296ms
cutoff: 240000	10times Time:1359ms
cutoff: 250000	10times Time:1328ms
cutoff: 260000	10times Time:891ms
cutoff: 270000	10times Time:874ms
cutoff: 280000	10times Time:891ms
cutoff: 290000	10times Time:890ms
cutoff: 300000	10times Time:844ms

We could find that in this case, time drops a lot after the cutoff is more than 250,000.

At which time, $\frac{arrays.length}{threadNumber} = \frac{2000000}{8} = 250,000$.

(3)Output when threadNumber = 16 and array's length = 2,000,000

```

public class Main {
    public static int threadNumber = 16;
    public static ForkJoinPool myPool = new ForkJoinPool(threadNumber);
    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " + myPool.getParallelism());
        Random random = new Random();
        int[] array = new int[2000000];
        ArrayList<Long> timeList = new ArrayList<>();
        //int num = 0;
        for (int j = 0; j < 10; j++) {
            ParSort.cutoff = 120000 + 1000 * (j + 1);
            //num++;
            // 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( bound: 200000000);
            }
            time = System.currentTimeMillis() - startTime;
            timeList.add(time);
        }
        for (Long time : timeList) {
            System.out.println(time);
        }
    }
}

```

```

Run: Main
D:\Java\jdk1.8.0_151\bin\java.exe ...
Degree of parallelism: 16
cutoff: 121000      10times Time:2578ms
cutoff: 122000      10times Time:1593ms
cutoff: 123000      10times Time:1500ms
cutoff: 124000      10times Time:1499ms
cutoff: 125000      10times Time:1570ms
cutoff: 126000      10times Time:1894ms
cutoff: 127000      10times Time:1921ms
cutoff: 128000      10times Time:1677ms
cutoff: 129000      10times Time:1380ms
cutoff: 130000      10times Time:1171ms
Process finished with exit code 0
Build completed successfully in 1 s 313 ms (a minute ago)

```

Output text:

Degree of parallelism: 16

cutoff: 121000	10times Time:2578ms
cutoff: 122000	10times Time:1593ms
cutoff: 123000	10times Time:1500ms

cutoff: 124000	10times Time:1499ms
cutoff: 125000	10times Time:1578ms
cutoff: 126000	10times Time:1094ms
cutoff: 127000	10times Time:1281ms
cutoff: 128000	10times Time:1077ms
cutoff: 129000	10times Time:1188ms
cutoff: 130000	10times Time:1171ms

We could find that in this case, time drops a lot after the cutoff is more than 125,000.

At which time, $\frac{arrays.length}{threadNumber} = \frac{2000000}{16} = 125,000$.

(4)Output when threadNumber = 32 and array's length = 2,000,000

```

public class Main {
    public static int threadNumber = 32;
    public static ForkJoinPool myPool = new ForkJoinPool(threadNumber);
    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " + myPool.getParallelism());
        Random random = new Random();
        int[] array = new int[2000000];
        ArrayList<Long> timeList = new ArrayList<>();
        //int num = 0;
        for (int i = 0; i < 10; i++) {
            ParSort.cutoff = 42000 + 100 * (i + 1);
            //num++;
            // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
            long time;
            long startTime = System.currentTimeMillis();
            for (int i = 0; i < 10; i++) {
                for (int i = 0; i < array.length; i++) array[i] = random.nextInt( bound: 20000000);
            }
            timeList.add(new Long(System.currentTimeMillis() - startTime));
        }
        System.out.println("Time list: " + timeList);
    }
}

```

Run: Main

```

Degree of parallelism: 32
cutoff: 62100 10times Time:3905ms
cutoff: 62200 10times Time:2390ms
cutoff: 62300 10times Time:2109ms
cutoff: 62400 10times Time:2218ms
cutoff: 62500 10times Time:2340ms
cutoff: 62600 10times Time:1516ms
cutoff: 62700 10times Time:1577ms
cutoff: 62800 10times Time:1594ms
cutoff: 62900 10times Time:1531ms
cutoff: 63000 10times Time:1562ms
Process finished with exit code 0

```

Output text:

Degree of parallelism: 32	
cutoff: 62100	10times Time:3905ms
cutoff: 62200	10times Time:2390ms
cutoff: 62300	10times Time:2109ms
cutoff: 62400	10times Time:2218ms
cutoff: 62500	10times Time:2340ms

cutoff: 62600	10times Time:1516ms
cutoff: 62700	10times Time:1577ms
cutoff: 62800	10times Time:1594ms
cutoff: 62900	10times Time:1531ms
cutoff: 63000	10times Time:1562ms

We could find that in this case, time drops a lot after the cutoff is more than 62,500.

$$\text{At which time, } \frac{\text{arrays.length}}{\text{threadNumber}} = \frac{2000000}{32} = 62,500.$$

(5)Output when threadNumber = 64 and array's length = 2,000,000

```

public class Main {
    public static int threadNumber = 64;
    public static ForkJoinPool myPool = new ForkJoinPool(threadNumber);
    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " + myPool.getParallelism());
        Random random = new Random();
        int[] array = new int[2000000];
        ArrayList<Long> timeList = new ArrayList<>();
        //int num = 0;
        for (int j = 0; j < 10; j++) {
            ParSort.cutoff = 31200 + 10 * (j + 1);
            //num++;
            // 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);
            }
            time = System.currentTimeMillis() - startTime;
            timeList.add(Long.valueOf(time));
        }
        System.out.println("10times Time: " + timeList.get(0));
    }
}

```

```

D:\Java\jdk1.8.0_151\bin>java.exe ...
Degree of parallelism: 64
cutoff: 31210 10times Time:6155ms
cutoff: 31220 10times Time:5390ms
cutoff: 31230 10times Time:3437ms
cutoff: 31240 10times Time:3171ms
cutoff: 31250 10times Time:3405ms
cutoff: 31260 10times Time:2218ms
cutoff: 31270 10times Time:1969ms
cutoff: 31280 10times Time:2187ms
cutoff: 31290 10times Time:2140ms
cutoff: 31300 10times Time:1969ms

```

Output text:

Degree of parallelism: 64	
cutoff: 31210	10times Time:6155ms
cutoff: 31220	10times Time:5390ms
cutoff: 31230	10times Time:3437ms
cutoff: 31240	10times Time:3171ms
cutoff: 31250	10times Time:3405ms
cutoff: 31260	10times Time:2218ms
cutoff: 31270	10times Time:1969ms

cutoff: 31280	10times Time:2187ms
cutoff: 31290	10times Time:2140ms
cutoff: 31300	10times Time:1969ms

We could find that in this case, time drops a lot after the cutoff is more than 31,250.

At which time, $\frac{arrays.length}{threadNumber} = \frac{2000000}{64} = 31,250$.

(6)Output when threadNumber = 128 and array's length = 2,000,000

```

public class Main {
    public static int threadNumber = 128;
    public static ForkJoinPool myPool = new ForkJoinPool(threadNumber);
    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " + myPool.getParallelism());
        Random random = new Random();
        int[] array = new int[2000000];
        ArrayList<Long> timeList = new ArrayList<>();
        //int num = 0;
        for (int j = 0; j < 10; j++) {
            ParSort.cutoff = 15620 + (j + 1);
            //num++;
            // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
            long startTime = System.currentTimeMillis();
            for (int i = 0; i < 10; i++) {
                for (int j = 0; j < array.length; j++) array[j] = random.nextInt(bound: 20000000);
            }
        }
    }
}

```

```

Run: Main
Degree of parallelism: 128
cutoff: 15621      10times Time:9264ms
cutoff: 15622      10times Time:6498ms
cutoff: 15623      10times Time:6452ms
cutoff: 15624      10times Time:5686ms
cutoff: 15625      10times Time:5702ms
cutoff: 15626      10times Time:3015ms
cutoff: 15627      10times Time:2671ms
cutoff: 15628      10times Time:2624ms
Process finished with exit code 0

```

Output text:

Degree of parallelism: 128

cutoff: 15621	10times Time:9264ms
cutoff: 15622	10times Time:6498ms
cutoff: 15623	10times Time:6452ms
cutoff: 15624	10times Time:5686ms
cutoff: 15625	10times Time:5702ms
cutoff: 15626	10times Time:3015ms
cutoff: 15627	10times Time:2671ms
cutoff: 15628	10times Time:2624ms

cutoff: 15629	10times Time:2749ms
cutoff: 15630	10times Time:3156ms

We could find that in this case, time drops a lot after the cutoff is more than 15,625.

At which time, $\frac{arrays.length}{threadNumber} = \frac{2000000}{128} = 15,625$.

(7)Output when threadNumber = 256 and array's length = 2,000,000

```

public class Main {
    public static int threadNumber = 256;
    public static ForkJoinPool myPool = new ForkJoinPool(threadNumber);
    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " + myPool.getParallelism());
        Random random = new Random();
        int[] array = new int[2000000];
        ArrayList<Long> timeList = new ArrayList<>();
        //int num = 0;
        for (int j = 0; j < 48; j++) {
            ParSort.cutoff = 7806 + (j + 1);
            //num++;
            // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(100000000);
            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(100000000);
            }
            time = System.currentTimeMillis() - startTime;
            timeList.add(Long.valueOf(time));
        }
    }
}

```

```

Run: Main
Degree of parallelism: 256
cutoff: 7806      10times Time:22932ms
cutoff: 7807      10times Time:10404ms
cutoff: 7808      10times Time:11497ms
cutoff: 7809      10times Time:11872ms
cutoff: 7810      10times Time:10435ms
cutoff: 7811      10times Time:10482ms
cutoff: 7812      10times Time:10701ms
cutoff: 7813      10times Time:9388ms
cutoff: 7814      10times Time:5077ms
cutoff: 7815      10times Time:5893ms
Process finished with exit code 0

```

Output text:

Degree of parallelism: 256	
cutoff: 7806	10times Time:22932ms
cutoff: 7807	10times Time:10404ms
cutoff: 7808	10times Time:11497ms
cutoff: 7809	10times Time:11872ms
cutoff: 7810	10times Time:10435ms
cutoff: 7811	10times Time:10482ms
cutoff: 7812	10times Time:10701ms
cutoff: 7813	10times Time:9388ms
cutoff: 7814	10times Time:5077ms

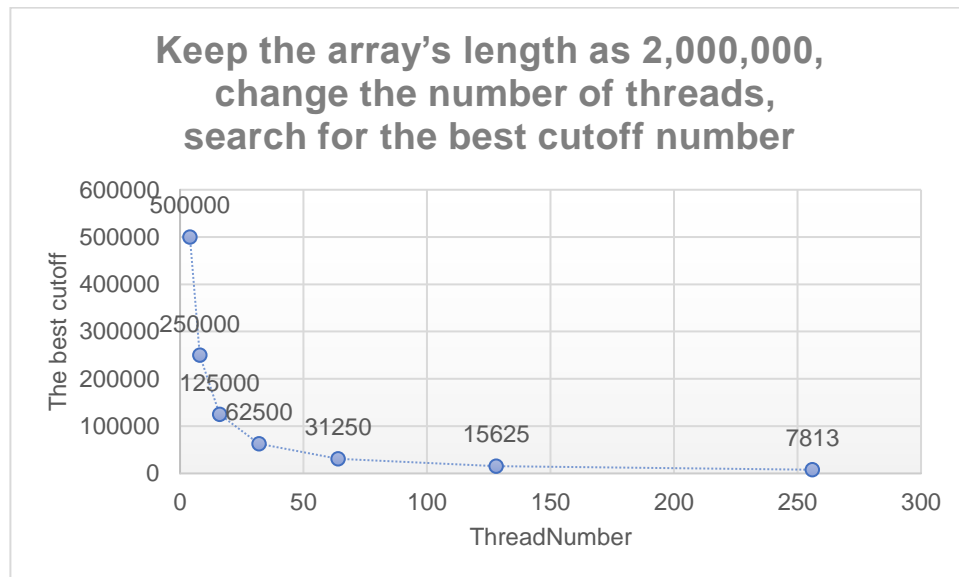
cutoff: 7815	10times Time:5093ms
---------------------	----------------------------

We could find that in this case, time drops a lot after the cutoff is more than 7,813.

At which time, $\frac{arrays.length}{threadNumber} = \frac{2000000}{256} = 7,812.5$.

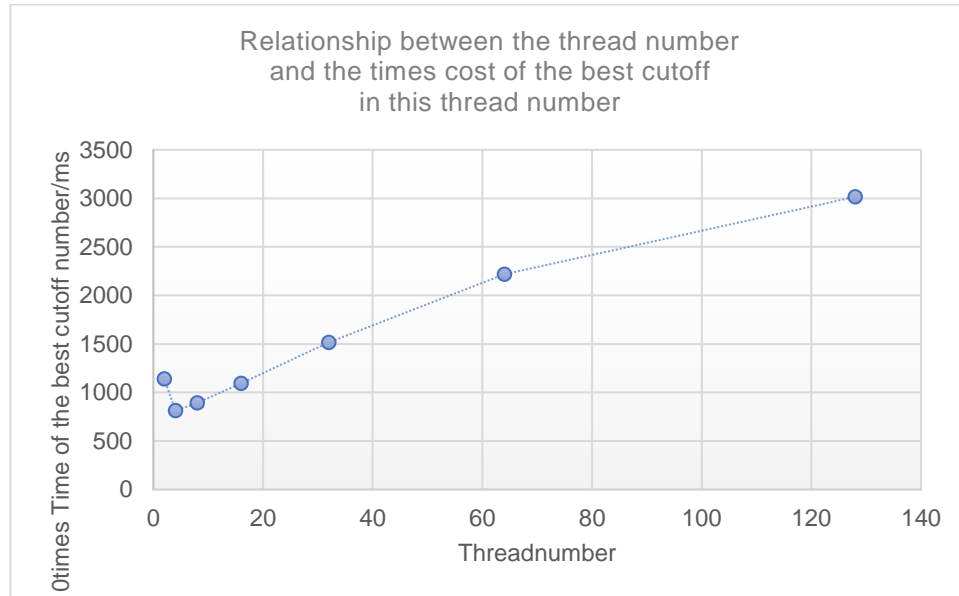
The simulation between the thread number and the best cutoff number when array's length is unchangeable is showed below.

Threadnumber	The best cutoff
4	500000
8	250000
16	125000
32	62500
64	31250
128	15625
256	7813



It shows that y-axis is inversely proportional to x, which means with the growth of the thread numbers, the best cutoffs decrease. So in the function to get the best cutoff number, the thread number should be in the position of the denominator.

Threadnumber	10times Time of the best cutoff number/ms
2	1140
4	812
8	891
16	1094
32	1516
64	2218
128	3015
256	5077

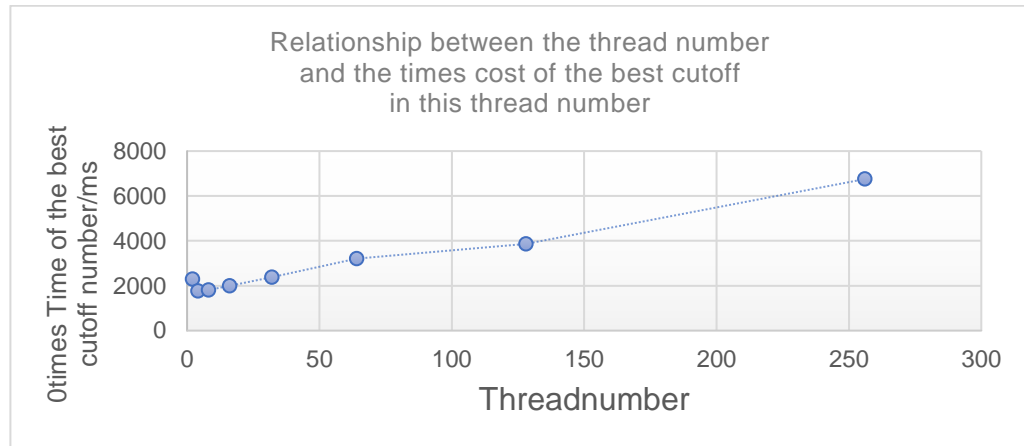


We can also make a conclusion that, though each thread number has its' own best cutoff number, the best cutoffs' time costs are different. In this simulation, we could find that when the thread number is 4, and the array's length is 2,000,000, it takes least time in the ParSort. In other words, the ParSort does best when there are 4 threads when the array's length is unchangeable.

To test if the thread number is 4 is best fits the ParSort, we could also make tests to test it. Because the steps are same as the above, I just give out the data and simulation directly.

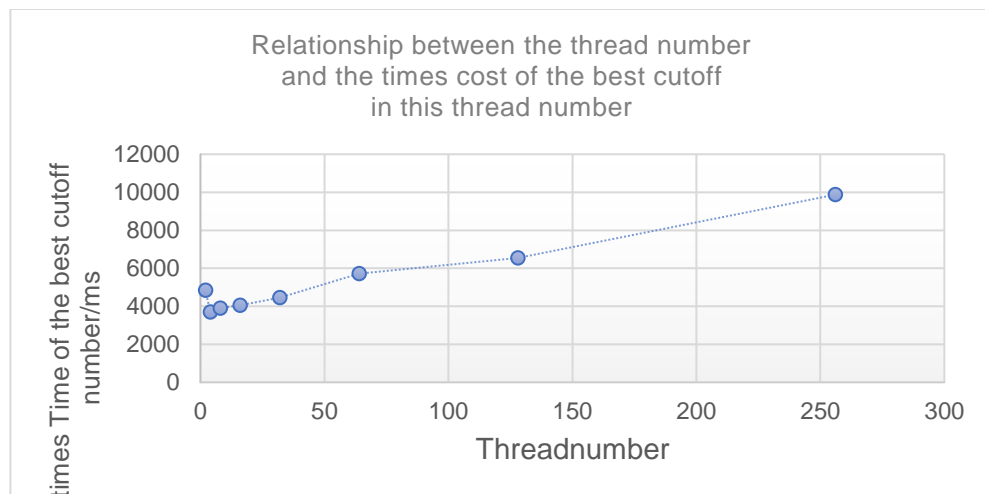
When the array's length = 4,000,000

Threadnumber	10times Time of the best cutoff number/ms
2	2296
4	1765
8	1812
16	1999
32	2374
64	3202
128	3859
256	6753



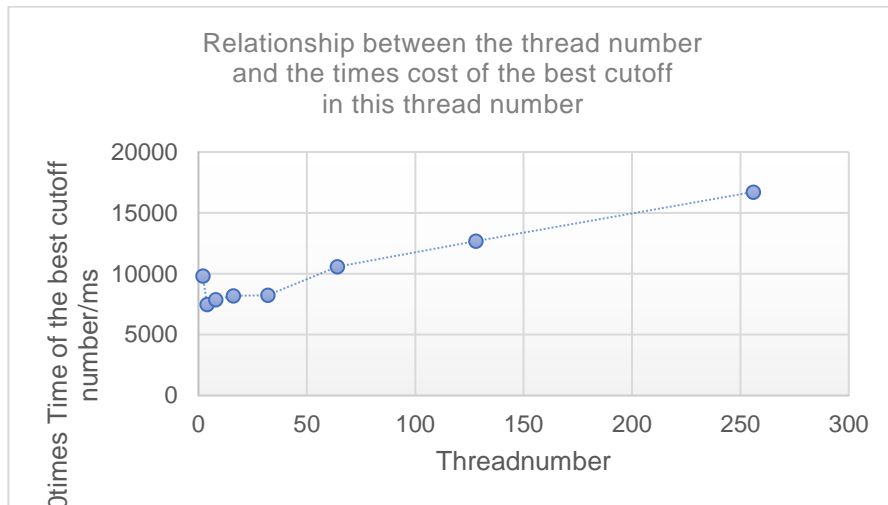
When the array's length = 8,000,000

Threadnumber	10times Time of the best cutoff number/ms
2	4843
4	3702
8	3906
16	4046
32	4452
64	5717
128	6546
256	9873

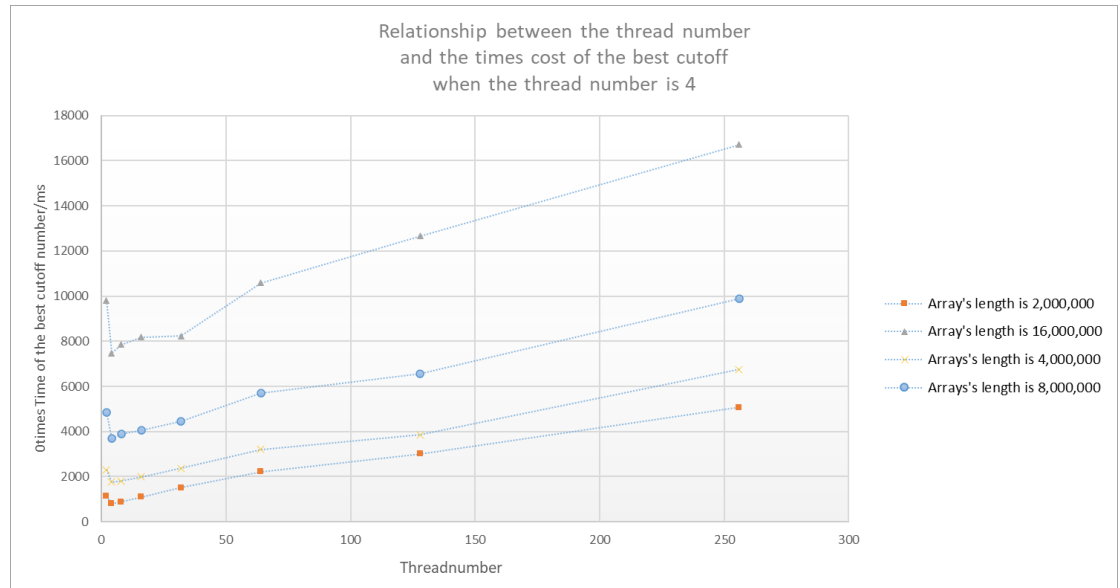


When the array's length = 16,000,000

Threadnumber	10times Time of the best cutoff number/ms
2	9810
4	7482
8	7858
16	8186
32	8233
64	10591
128	12669
256	16714



We could see that even the length of the array changes, the best cutoff numbers always cost the least time when the thread number is 4, which confirms the previous conclusion, that the ParSort does best when there are 4 threads when the array's length is unchangeable.



We could also find in the above diagram that the time costs of the best cutoff numbers increase with the number of threads when the number of threads is not 4, which means the increase of thread numbers doesn't improve the ParSort's performance. Why it happens? Personally, I think the main reason is, if multiple threads operate on the same atomic data synchronously, the resource may also be occupied easily.

③ Keep the number of threads unchangeable, change the array's length, search for the best cutoff number

(1) Output when threadNumber = 4 and array's length = 1,000,000

The screenshot shows an IDE with a Java project named 'INFO6205-Fall2021'. The code in 'Main.java' defines a 'Main' class with a 'main' method that uses a 'ForkJoinPool' with 4 threads to sort an array of 1,000,000 integers. The output window shows the following results:

```

Degree of parallelism: 4
cutoff: 210000    10times Time:953ms
cutoff: 220000    10times Time:703ms
cutoff: 230000    10times Time:719ms
cutoff: 240000    10times Time:687ms
cutoff: 250000    10times Time:703ms
cutoff: 260000    10times Time:438ms
cutoff: 270000    10times Time:437ms
cutoff: 280000    10times Time:422ms
cutoff: 290000    10times Time:437ms
cutoff: 300000    10times Time:406ms

```

Output text:

Degree of parallelism: 4

cutoff: 210000	10times Time:953ms
cutoff: 220000	10times Time:703ms
cutoff: 230000	10times Time:719ms
cutoff: 240000	10times Time:687ms
cutoff: 250000	10times Time:703ms
cutoff: 260000	10times Time:438ms
cutoff: 270000	10times Time:437ms
cutoff: 280000	10times Time:422ms
cutoff: 290000	10times Time:437ms
cutoff: 300000	10times Time:406ms

We could find that in this case, time drops a lot after the cutoff is more than 250,000.

At which time, $\frac{arrays.length}{threadNumber} = \frac{1000000}{4} = 250,000$.

(2)Output when threadNumber = 4 and array's length = 2,000,000


```

package edu.neu.coe.info6205.sort.par;

import java.util.*;

/**
 * This code has been fleshed out by Ziyao Qiao. Thanks very much.
 * 7000 tidy it up a bit.
 */
public class Main {
    public static int threadNumber = 4;
    public static ForkJoinPool myPool = new ForkJoinPool(threadNumber);
    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " + myPool.getParallelism());
        Random random = new Random();
        int[] array = new int[2000000];
        ArrayList<Long> timeList = new ArrayList<>();
        //int num = 0;
    }
}

```

Run: Main

```

Degree of parallelism: 4
cutoff: 430000 10times Time:1734ms
cutoff: 420000 10times Time:1448ms
cutoff: 430000 10times Time:1515ms
cutoff: 440000 10times Time:1344ms
cutoff: 450000 10times Time:1375ms
cutoff: 460000 10times Time:1327ms
cutoff: 470000 10times Time:1359ms
cutoff: 480000 10times Time:1344ms
cutoff: 490000 10times Time:1265ms
cutoff: 500000 10times Time:1328ms
cutoff: 510000 10times Time:812ms
cutoff: 520000 10times Time:813ms

```

```

package edu.neu.coe.info6205.sort.par;

import java.util.*;

/**
 * This code has been fleshed out by Ziyao Qiao. Thanks very much.
 * 7000 tidy it up a bit.
 */
public class Main {
    public static int threadNumber = 4;
    public static ForkJoinPool myPool = new ForkJoinPool(threadNumber);
    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " + myPool.getParallelism());
        Random random = new Random();
        int[] array = new int[2000000];
        ArrayList<Long> timeList = new ArrayList<>();
        //int num = 0;
    }
}

```

Run: Main

```

cutoff: 500000 10times Time:1328ms
cutoff: 510000 10times Time:812ms
cutoff: 520000 10times Time:813ms
cutoff: 530000 10times Time:828ms
cutoff: 540000 10times Time:812ms
cutoff: 550000 10times Time:828ms
cutoff: 560000 10times Time:828ms
cutoff: 570000 10times Time:812ms
cutoff: 580000 10times Time:812ms
cutoff: 590000 10times Time:828ms
cutoff: 600000 10times Time:797ms

```

Process finished with exit code 0

Output text:

Degree of parallelism: 4	
cutoff: 450000	10times Time:1375ms
cutoff: 460000	10times Time:1327ms
cutoff: 470000	10times Time:1359ms
cutoff: 480000	10times Time:1344ms
cutoff: 490000	10times Time:1265ms
cutoff: 500000	10times Time:1328ms
cutoff: 510000	10times Time:812ms
cutoff: 520000	10times Time:813ms

cutoff: 530000	10times Time:828ms
cutoff: 540000	10times Time:812ms
cutoff: 550000	10times Time:828ms

We could find that in this case, time drops a lot after the cutoff is more than 500,000.

At which time, $\frac{\text{arrays.length}}{\text{threadNumber}} = \frac{2000000}{4} = 500,000$.

(3)Output when threadNumber = 4 and array's length = 4,000,000

```

INFO6205-Fall2021 [INFO6205] C:\Users\delip\Desktop\semester1\Algorithms\INFO6205-Fall2021\Main.java
15 // 7000 tidy it up a bit.
16
17 public class Main {
18     public static int threadNumber = 4;
19     public static ForkJoinPool myPool = new ForkJoinPool(threadNumber);
20     public static void main(String[] args) {
21         processArgs(args);
22         System.out.println("Degree of parallelism: " + myPool.getParallelism());
23         Random random = new Random();
24         int[] array = new int[4000000];
25         ArrayList<Long> timeList = new ArrayList<>();
26         //int num = 0;
27         for (int j = 99; j < 110; j++) {
28             ParSort.cutoff = 10000 * (j + 1);
29             //num++;
30             // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(100000000);
31             long time;
32             long startTime = System.currentTimeMillis();

```

Run: Main

cutoff	10times Time
910000	10times Time:1358ms
920000	10times Time:2948ms
930000	10times Time:2890ms
940000	10times Time:3046ms
950000	10times Time:3093ms
960000	10times Time:2937ms
970000	10times Time:3080ms
980000	10times Time:2658ms
990000	10times Time:2921ms
1000000	10times Time:3042ms
1010000	10times Time:1734ms
1020000	10times Time:1797ms

Run: Main

cutoff	10times Time
1030000	10times Time:1749ms
1040000	10times Time:1744ms
1050000	10times Time:1750ms
1060000	10times Time:1794ms
1070000	10times Time:1781ms
1080000	10times Time:1765ms
1090000	10times Time:1765ms
1100000	10times Time:1797ms

Process finished with exit code 0

Output text:

Degree of parallelism: 4	
cutoff: 910000	10times Time:3358ms
cutoff: 920000	10times Time:2968ms
cutoff: 930000	10times Time:2890ms
cutoff: 940000	10times Time:3046ms
cutoff: 950000	10times Time:3093ms
cutoff: 960000	10times Time:2937ms
cutoff: 970000	10times Time:3000ms
cutoff: 980000	10times Time:2858ms
cutoff: 990000	10times Time:2921ms
cutoff: 1000000	10times Time:3062ms
cutoff: 1010000	10times Time:1734ms
cutoff: 1020000	10times Time:1797ms
cutoff: 1030000	10times Time:1749ms
cutoff: 1040000	10times Time:1734ms
cutoff: 1050000	10times Time:1750ms
cutoff: 1060000	10times Time:1796ms
cutoff: 1070000	10times Time:1781ms
cutoff: 1080000	10times Time:1765ms
cutoff: 1090000	10times Time:1765ms
cutoff: 1100000	10times Time:1797ms

We could find that in this case, time drops a lot after the cutoff is more than 1,000,000.

At which time, $\frac{arrays.length}{threadNumber} = \frac{4000000}{4} = 1,000,000$.

(4)Output when threadNumber = 4 and array's length = 8,000,000

```

15 // + 1000 tidy it up a bit.
16 //
17 public class Main {
18     public static int threadNumber = 4;
19     public static ForkJoinPool myPool = new ForkJoinPool(threadNumber);
20     public static void main(String[] args) {
21         processArgs(args);
22         System.out.println("Degree of parallelism: " + myPool.getParallelism());
23         Random random = new Random();
24         int[] array = new int[1000000];
25         ArrayList<Long> timeList = new ArrayList<>();
26         //int num = 0;
27         for (int i = 190; i < 210; i++) {
28             ParSort.cutoff = 10000 * (i + 1);
29             //num++;
30             // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(100000000);
31             long time;
32             long startTime = System.currentTimeMillis();

```

Run: Main

```

B:\Java\jdk1.8.0_151\bin\java.exe ...
Degree of parallelism: 4
cutoff: 1910000 10times Time:6826ms
cutoff: 1920000 10times Time:6327ms
cutoff: 1930000 10times Time:6139ms
cutoff: 1940000 10times Time:6218ms
cutoff: 1950000 10times Time:6389ms
cutoff: 1960000 10times Time:6357ms
cutoff: 1970000 10times Time:6296ms
cutoff: 1980000 10times Time:6248ms
cutoff: 1990000 10times Time:5937ms
cutoff: 2000000 10times Time:6311ms
cutoff: 2010000 10times Time:3624ms
cutoff: 2020000 10times Time:3639ms

```

```

cutoff: 2000000 10times Time:6311ms
cutoff: 2010000 10times Time:3624ms
cutoff: 2020000 10times Time:3639ms
cutoff: 2030000 10times Time:3671ms
cutoff: 2040000 10times Time:3656ms
cutoff: 2050000 10times Time:3671ms
cutoff: 2060000 10times Time:3624ms
cutoff: 2070000 10times Time:3655ms
cutoff: 2080000 10times Time:3671ms
cutoff: 2090000 10times Time:3641ms
cutoff: 2100000 10times Time:3783ms
Process finished with exit code 0

```

Output text:

Degree of parallelism: 4

cutoff: 1950000	10times Time:6389ms
cutoff: 1960000	10times Time:6357ms
cutoff: 1970000	10times Time:6296ms
cutoff: 1980000	10times Time:6248ms
cutoff: 1990000	10times Time:5937ms
cutoff: 2000000	10times Time:6311ms
cutoff: 2010000	10times Time:3624ms

cutoff: 2020000	10times Time:3639ms
cutoff: 2030000	10times Time:3671ms
cutoff: 2040000	10times Time:3656ms
cutoff: 2050000	10times Time:3671ms

We could find that in this case, time drops a lot after the cutoff is more than 2,000,000.

At which time, $\frac{arrays.length}{threadNumber} = \frac{8000000}{4} = 2,000,000$.

(5)Output when threadNumber = 4 and array's length = 16,000,000

```

15 // TODO: Find it up a bit.
16
17 public class Main {
18     public static int threadNumber = 4;
19     public static ForkJoinPool myPool = new ForkJoinPool(threadNumber);
20     public static void main(String[] args) {
21         processArgs(args);
22         System.out.println("Degree of parallelism: " + myPool.getParallelism());
23         Random random = new Random();
24         int[] array = new int[16000000];
25         ArrayList<Long> timeList = new ArrayList<>();
26         //int num = 8;
27         for (int j = 395; j < 410; j++) {
28             ParSort.cutoff = 4000000 * (j + 1);
29             //num++;
30             // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
31             long time;
32             long startTime = System.currentTimeMillis();

```

```

Run: Main
81\Java\jdk1.8.0_151\bin\java.exe ...
Degree of parallelism: 4
cutoff: 3960000 10times Time:13301ms
cutoff: 3970000 10times Time:13278ms
cutoff: 3980000 10times Time:12774ms
cutoff: 3990000 10times Time:13278ms
cutoff: 4000000 10times Time:13047ms
cutoff: 4010000 10times Time:7492ms
cutoff: 4020000 10times Time:7404ms
cutoff: 4030000 10times Time:7661ms

```

Output text:

Degree of parallelism: 4

cutoff: 3960000	10times Time:13301ms
cutoff: 3970000	10times Time:13278ms
cutoff: 3980000	10times Time:12774ms
cutoff: 3990000	10times Time:13278ms
cutoff: 4000000	10times Time:13047ms
cutoff: 4010000	10times Time:7492ms
cutoff: 4020000	10times Time:7404ms
cutoff: 4030000	10times Time:7661ms

cutoff: 4040000	10times Time:7704ms
cutoff: 4050000	10times Time:7480ms
cutoff: 4060000	10times Time:7405ms
cutoff: 4070000	10times Time:7592ms

We could find that in this case, time drops a lot after the cutoff is more than 4,000,000.

At which time, $\frac{arrays.length}{threadNumber} = \frac{16000000}{4} = 4,000,000$.

(6)Output when threadNumber = 4 and array's length = 32,000,000

```

package edu.neu.coe.info6205.sort.par;

import ...

/**
 * This code has been fleshed out by Zupao Qiao. Thanks very much.
 * TQOO tidy it up a bit.
 */

public class Main {

    public static int threadNumber = 4;
    public static ForkJoinPool myPool = new ForkJoinPool(threadNumber);

    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " + myPool.getParallelism());
        Random random = new Random();
        int[] array = new int[32000000];
        ArrayList<Long> timeList = new ArrayList<>();
        //int num = 0;
    }
}

```

```

D:\Java\jdk1.8.0_151\bin\java.exe ...
Degree of parallelism: 4
cutoff: 7960000 10times Time:28744ms
cutoff: 7970000 10times Time:26681ms
cutoff: 7980000 10times Time:27035ms
cutoff: 7990000 10times Time:26915ms
cutoff: 8000000 10times Time:26443ms
cutoff: 8010000 10times Time:16105ms
cutoff: 8020000 10times Time:16153ms
cutoff: 8030000 10times Time:16168ms

```

Output text:

Degree of parallelism: 4	
cutoff: 7960000	10times Time:28744ms
cutoff: 7970000	10times Time:26681ms
cutoff: 7980000	10times Time:27035ms
cutoff: 7990000	10times Time:26915ms
cutoff: 8000000	10times Time:26443ms
cutoff: 8010000	10times Time:16105ms
cutoff: 8020000	10times Time:16153ms
cutoff: 8030000	10times Time:16168ms

cutoff: 8040000	10times Time:16230ms
cutoff: 8050000	10times Time:16325ms

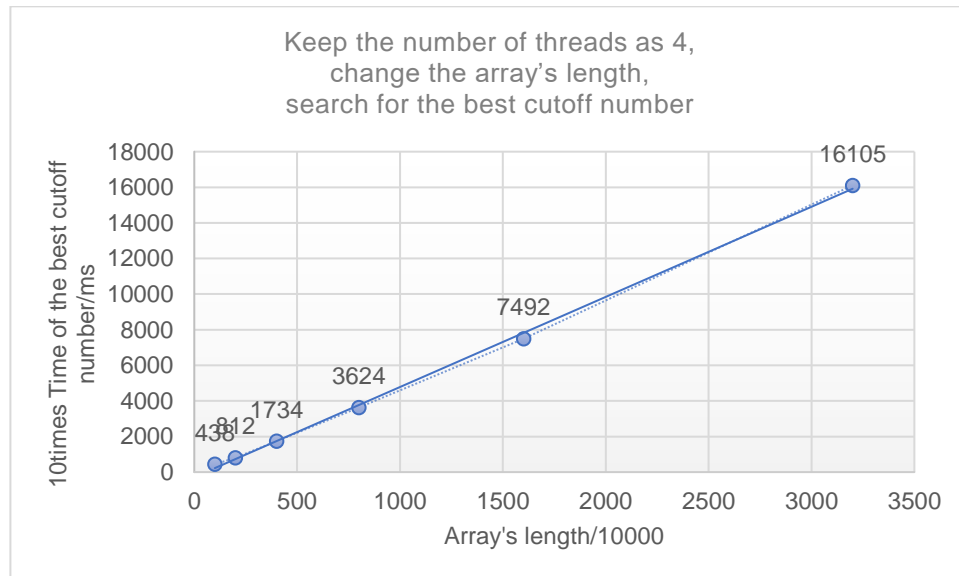
We could find that in this case, time drops a lot after the cutoff is more than 8,000,000.

At which time, $\frac{arrays.length}{threadNumber} = \frac{32000000}{4} = 8,000,000$.

It seems that when threadNumber = 4 and array's length = 64,000,000, my computer didn't work well so I stopped the simulation.

The simulation between the array's length and the best cutoff number when the number of threads is unchangeable is showed below.

Array's length/10000	10times Time of the best cutoff number/ms	x/y
100	438	4.38
200	812	4.06
400	1734	4.36
800	3624	4.53
1600	7492	4.68
3200	16105	5.03



Combined with the analysis of the previous point ②, we could make a conclusion that the function between the number of threads, the array's length and the best cutoff number is as the following:

We could make tests for our conclusion.

- (1) Output when threadNumber = 2 and array's length = 1,000,000**

The screenshot shows an IDE with a Java file named `Main.java`. The code is as follows:

```

15 // TODO tidy it up a bit.
16
17 public class Main {
18     public static int threadNumber = 2;
19     public static ForkJoinPool myPool = new ForkJoinPool(threadNumber);
20     public static void main(String[] args) {
21         processArgs(args);
22         System.out.println("Degree of parallelism: " + myPool.getParallelism());
23         Random rand = new Random();
24         int[] array = new int[1000000];
25         ArrayList<Long> timeList = new ArrayList<>();
26         //int num = 0;
27         for (int j = 0; j <= 10; j++) {
28             ParSort.cutoff = 10000 + (j + 1);
29             //num++;
30             // for (int i = 0; i < array.length; i++) array[i] = rand.nextInt(10000000);
31             long time;
32             long startTime = System.currentTimeMillis();

```

The Run output shows the following results:

```

Run: Main
Degree of parallelism: 2
cutoff: 40000 10times Time:1093ms
cutoff: 47000 10times Time:813ms
cutoff: 48000 10times Time:812ms
cutoff: 49000 10times Time:781ms
cutoff: 50000 10times Time:859ms
cutoff: 51000 10times Time:516ms
cutoff: 52000 10times Time:531ms
cutoff: 53000 10times Time:531ms
cutoff: 54000 10times Time:531ms
cutoff: 55000 10times Time:531ms
Process finished with exit code 0

```

Degree of parallelism: 2	
cutoff: 460000	10times Time:1093ms

cutoff: 470000	10times Time:813ms
cutoff: 480000	10times Time:812ms
cutoff: 490000	10times Time:781ms
cutoff: 500000	10times Time:859ms
cutoff: 510000	10times Time:516ms
cutoff: 520000	10times Time:531ms
cutoff: 530000	10times Time:531ms
cutoff: 540000	10times Time:531ms
cutoff: 550000	10times Time:531ms

(2)Output when threadNumber = 4 and array's length = 2,000,000

```

package edu.neu.coe.info6205.sort.par;

import ...

/**
 * This code has been fleshed out by Ziqiao Qiao. Thanks very much.
 * 7000 tidy it up a bit.
 */

public class Main {

    public static int threadNumber = 4;
    public static ForkJoinPool myPool = new ForkJoinPool(threadNumber);
    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " + myPool.getParallelism());
        Random random = new Random();
        int[] array = new int[2000000];
        ArrayList<Long> timeList = new ArrayList<>();
        //int num = 0;
    }
}

```

Run: Main

Cutoff	10times Time
470000	1734ms
420000	1348ms
430000	1515ms
440000	1344ms
450000	1375ms
460000	1327ms
470000	1359ms
480000	1344ms
490000	1265ms
500000	1328ms
510000	812ms
520000	813ms

```

package edu.neu.coe.info6205.sort.par;

import ...

/**
 * This code has been fleshed out by Ziqiao Qiao. Thanks very much.
 * 7000 tidy it up a bit.
 */

public class Main {

    public static int threadNumber = 4;
    public static ForkJoinPool myPool = new ForkJoinPool(threadNumber);
    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " + myPool.getParallelism());
        Random random = new Random();
        int[] array = new int[2000000];
        ArrayList<Long> timeList = new ArrayList<>();
        //int num = 0;
    }
}

```

Run: Main

Cutoff	10times Time
500000	1328ms
510000	812ms
520000	812ms
530000	828ms
540000	812ms
550000	828ms
560000	828ms
570000	812ms
580000	812ms
590000	828ms
600000	797ms

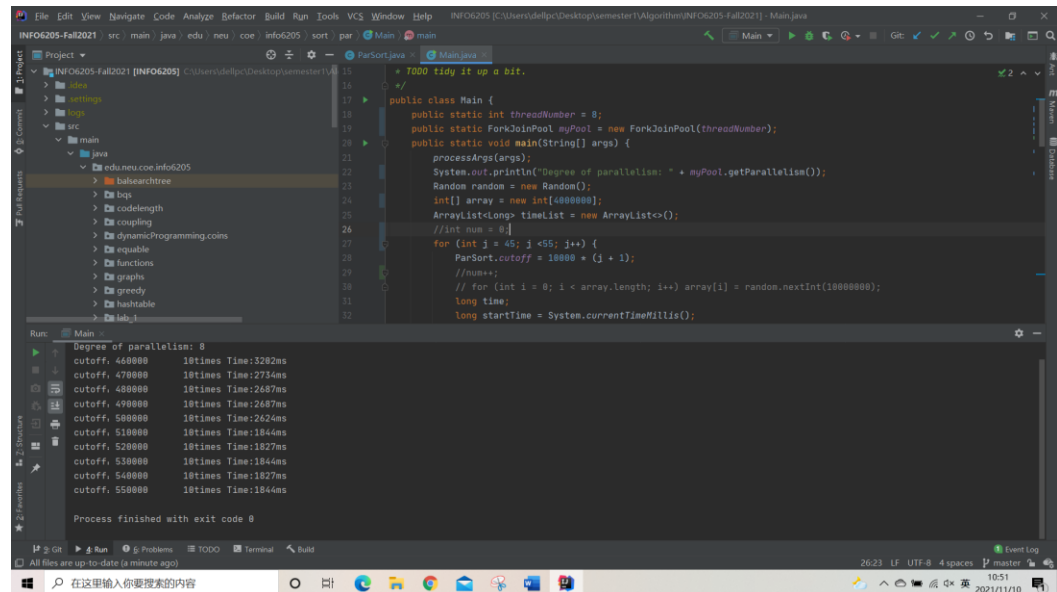
Process finished with exit code 0

Output text:

Degree of parallelism: 4

cutoff: 450000	10times Time:1375ms
cutoff: 460000	10times Time:1327ms
cutoff: 470000	10times Time:1359ms
cutoff: 480000	10times Time:1344ms
cutoff: 490000	10times Time:1265ms
cutoff: 500000	10times Time:1328ms
cutoff: 510000	10times Time:812ms
cutoff: 520000	10times Time:813ms
cutoff: 530000	10times Time:828ms
cutoff: 540000	10times Time:812ms
cutoff: 550000	10times Time:828ms

(3) Output when threadNumber = 8 and array's length = 4,000,000



```
INFO6205-Fall2021 src / main java edu neu coe info6205 sort par Main main
+ 7000 tidy it up a bit.
15
16
17 public class Main {
18     public static int threadNumber = 8;
19     public static ForkJoinPool myPool = new ForkJoinPool(threadNumber);
20     public static void main(String[] args) {
21         processArgs(args);
22         System.out.println("Degree of parallelism: " + myPool.getParallelism());
23         Random random = new Random();
24         int[] array = new int[4000000];
25         ArrayList<Long> timelist = new ArrayList<>();
26         //int num = 8;
27         for (int j = 45; j <= 55; j++) {
28             ParSort.cutoff = 10000 * (j + 1);
29             //num++;
30             // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
31             long time;
32             long startTime = System.currentTimeMillis();
33             myPool.invoke(new ParSort(array, 0, array.length - 1));
34             time = System.currentTimeMillis() - startTime;
35             timelist.add(Long.valueOf(time));
36         }
37         System.out.println("Average time: " + timelist.stream().mapToLong(Long::longValue).average().orElse(0L));
38     }
39 }
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
25
```

cutoff: 500000	10times Time:2624ms
cutoff: 510000	10times Time:1844ms
cutoff: 520000	10times Time:1827ms
cutoff: 530000	10times Time:1844ms
cutoff: 540000	10times Time:1827ms
cutoff: 550000	10times Time:1844ms

(4) Output when threadNumber = 16 and array's length = 8,000,000

The screenshot shows an IDE with a Java file named `Main.java`. The code defines a `Main` class with a `threadNumber` of 16 and a `ForkJoinPool` named `myPool`. The `main` method processes arguments, prints the degree of parallelism, initializes a random array of length 8,000,000, and runs a loop where the cutoff is increased by 10,000 for each iteration. The output window shows the results of these iterations.

```

+ TODO tidy it up a bit.
+//
public class Main {
    public static int threadNumber = 16;
    public static ForkJoinPool myPool = new ForkJoinPool(threadNumber);
    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " + myPool.getParallelism());
        Random random = new Random();
        int[] array = new int[8000000];
        ArrayList<Long> timelist = new ArrayList<>();
        //int num = 8;
        for (int j = 45; j < 55; j++) {
            ParSort.cutoff = 10000 * (j + 1);
            //num++;
            // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
            long time;
            long startTime = System.currentTimeMillis();

```

Run: Main

```

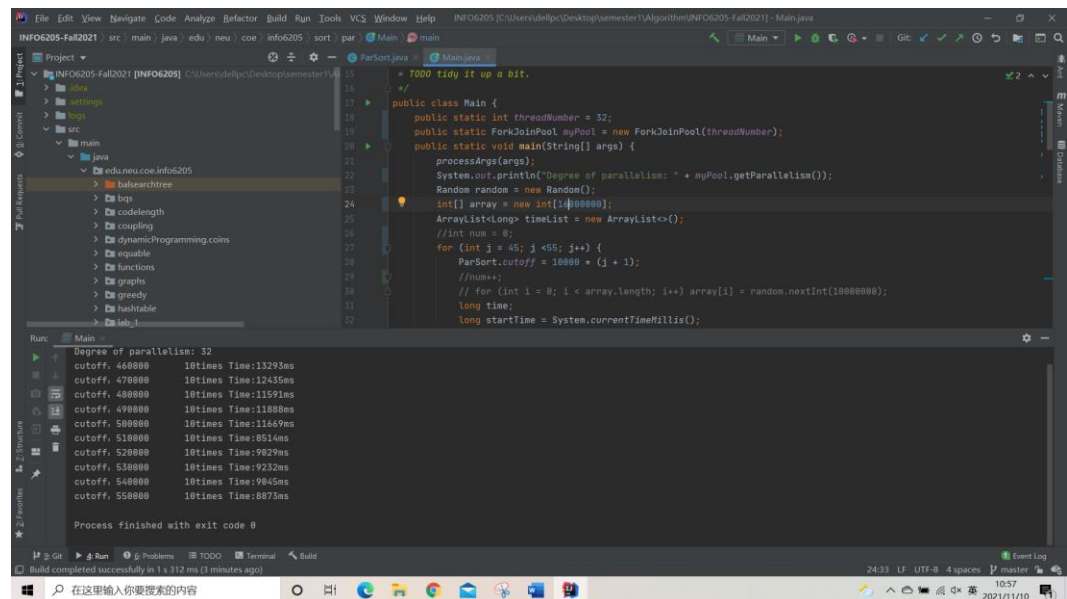
Degree of parallelism: 16
cutoff: 460000      10times Time:6624ms
cutoff: 470000      10times Time:5778ms
cutoff: 480000      10times Time:5514ms
cutoff: 490000      10times Time:5421ms
cutoff: 500000      10times Time:5514ms
cutoff: 510000      10times Time:3656ms
cutoff: 520000      10times Time:3702ms
cutoff: 530000      10times Time:3812ms
cutoff: 540000      10times Time:3780ms
cutoff: 550000      10times Time:3765ms
Process finished with exit code 0

```

Output text:

Degree of parallelism: 16	
cutoff: 460000	10times Time:6624ms
cutoff: 470000	10times Time:5778ms
cutoff: 480000	10times Time:5514ms
cutoff: 490000	10times Time:5421ms
cutoff: 500000	10times Time:5514ms
cutoff: 510000	10times Time:3656ms
cutoff: 520000	10times Time:3702ms
cutoff: 530000	10times Time:3812ms
cutoff: 540000	10times Time:3780ms
cutoff: 550000	10times Time:3765ms

(5) Output when threadNumber = 32 and array's length = 16,000,000



Output text:

Degree of parallelism: 32

cutoff: 460000	10times Time:13293ms
cutoff: 470000	10times Time:12435ms
cutoff: 480000	10times Time:11591ms
cutoff: 490000	10times Time:11888ms
cutoff: 500000	10times Time:11669ms
cutoff: 510000	10times Time:8514ms
cutoff: 520000	10times Time:9029ms
cutoff: 530000	10times Time:9232ms
cutoff: 540000	10times Time:9045ms
cutoff: 550000	10times Time:8873ms

We could see in the tests, if the $\frac{arrays.length}{threadNumber}$ is unchangeable, the best cutoff number always occurs at the number of 500,000, which is equals to the number of $\frac{arrays.length}{threadNumber}$. The function seems right.