Project4: Multithreaded Sorting Application & Fork-Join Sorting Application

518021911193 刘昊林

1 使用到的功能、函数和命令

1.1 采用 PthreadsAPI 的基本代码结构

```
int main()
{
        /* the thread identifier */
        pthread_t tid;
        /* set of attributes for the thread */
        pthread_attr_t attr;

        /* get the default attributes */
        pthread_attr_init(&attr);

        /* create the thread */
        pthread_create(&tid,&attr,runner,argv[1]);

        /* now wait for the thread to exit */
        pthread_join(tid,NULL);

        /* The thread will begin control in this function*/
        void *runner(void *param) {
                do something here.

                pthread_exit(0);
        }
}
```

1.2 采用 Java fork-join parallelism API 的基本代码结构

```
1         import java.util.concurrent.*;
2         public class SumTask extends RecursiveTask<Integer> {
3         static final int SIZE = 10000;
4         static final int THRESHOLD = 1000;
5         private int begin,end;
6         private int[] array;
7         public SumTask(int begin, int end, int[] array) {
8                 this.begin = begin;
9                 this.end = end;
10        this.array = array;
11        }
12        protected Integer compute() {
13                if (end - begin < THRESHOLD) {
14                int sum = 0;
15                for (int i = begin; i <= end; i++)
16                        sum += array[i];
17                return sum;
18                }
19        else {
20                int mid = begin + (end - begin) / 2;
21                SumTask leftTask = new SumTask(begin, mid, array);
22                SumTask rightTask = new SumTask(mid + 1, end, array);
23                leftTask.fork();
24                rightTask.fork();
25                return rightTask.join() + leftTask.join();
26                }}
27        public static void main(String[] args) {
28                ForkJoinPool pool = new ForkJoinPool();
29                int[] array = new int[SIZE];
30                java.util.Random rand = new java.util.Random();
31                for (int i = 0; i < SIZE; i++)
32                        array[i] = rand.nextInt(10);
33                SumTask task = new SumTask(0, SIZE-1, array);
34                int sum = pool.invoke(task);
35                System.out.println("The sum is " + sum);
36        }}
```

1.3 使用到的函数和类

1. pthread_attr_init(&attr) 函数，为线程设置参数。
2. pthread_create(&tid,&attr,runner,NULL) 函数，创建新的线程。
3. pthread_join(&tid,NULL) 函数，等待线程结束。
4. (java)RecursiveAction 子类，处理没有返回结果的任务的子类。
5. (java)System.out.println() 函数，控制输出的函数。
6. (java)task.fork() 函数，创建一个新线程。
7. (java)task.join() 函数，等待线程任务执行并返回。
8. (java)ForkJoinPool pool = new ForkJoinPool() 函数，创建一个新的 ForkJoinPool。
9. (java)scan_input.nextInt 函数，获取下一个为 int 型的数据输入。
10. (java)scan_input.next().charAt(0) 函数，获取下一个为 char 型的数据输入。
11. (java)java.util.Random() 函数，用于生成随机数。
12. (java)pool.invoke 函数，提交任务并且开始运行。

1.4 使用到的命令

1. make，编译 Multithread.c 文件。
2. ./Multithread，运行该文件。
3. java QuickSort.java，编译并且运行该文件。

2 实现思路

2.1 Part 1-Multithreaded Sorting Application

首先设置四个全局变量：原数组 *array、结果数组 *result、数组元素个数 num_of_array、线程计数器 thread_count。然后进行依次创建两个排序线程 sort_thread[0] 和 sort_thread[1]，在这两个线程中 sort_thread[0] 中的 thread_count 值为 0，sort_thread[1] 中的 thread_count 值为 1，根据 thread_count 的值来将原数组划分为大小相等两部分，并且分别利用冒泡排序进行排序。并且利用线程中的 pthread_exit(0) 函数和进程中的 pthread_join(sort_thread[i],NULL) 函数，依次等待线程的结束和返回。

数组的两个部分在两个线程中分别排好序之后，再创建一个归并线程 merge_thread，对原数组已经排好序的两部分进行简单的归并排序，并且把结果放入数组 result 中即可。然后利用线程中的 pthread_exit(0) 函数和进程中的 pthread_join(merge_thread,NULL) 函数，等待线程的结束和返回即可。

最终将排好序的数组输出。

2.2 Part 2-Fork-Join Sorting Application

首先引入一系列必要的头文件，例如 import java.util.concurrent.*、import java.util.Scanner、import java.util.Arrays。然后从 RecursiveAction 子类中创建一个与文件名对应的新类，MergeSort

中创建方式为 public class MergeSort extends RecursiveAction，QuickSort 中的创建方式为 public class QuickSort extends RecursiveAction。

然后实现这个新类的构造函数，由于类中的元素包括排序起点 begin(int)、排序终点 end(int)、排序数组 array(int array)，所以构造函数为 public QuickSort/MergeSort(int begin, int end, int[] array){this.begin = begin;this.end = end;this.array = array;}。

然后实现这个类的排序函数 compute()，当需要排序的范围 end-begin 小于 100 时，均采用插入排序。当排序范围大于 100 时，QuickSort 采用快速排序，首先把数组中元素排列为哨兵元素左侧的元素均小于哨兵元素，右侧均大于，然后利用 QuickSort lefttask = new QuickSort(begin, mid, array) 和 lefttask.fork()(右侧数组操作相同) 来创建两个新线程，分别用同样的方法对两个子序列进行排序，然后使用 lefttask.join() 等待线程运行结束并返回。MergeSort 采用归并排序，把数组层层二分，直到剩下两个元素是再比较大小并交换。在这个过程中，利用 MergeSort lefttask = new MergeSort(begin, mid, array) 和 lefttask.fork()(右侧数组操作相同) 创建两个新线程，分别用同样的方法对两个子序列进行排序，然后使用 lefttask.join() 等待线程运行结束并返回，返回后再对两个子序列进行遍历归并排序即可。

在排序的过程中，我还将每一段应用的排序方法进行输出，便于观察每个线程中排序区间的大小和具体排序方法。

最后是这个类的主函数 main(String [] args)，无论是 QuickSort 还是 MergeSort，均先由用户输入是用默认数组尺寸还是自己设置数组尺寸，然后获得数组的具体尺寸。然后产生同样大小的随机数数组，并且输出未排序的原始数组。然后依据参数创建一个新的排序类 task，并且使用 pool.invoke(task) 来运行。最终输出排好序的数组。

3 步骤截图

3.1 Part 1-Multithreaded Sorting Application

```
psyduckliu@ubuntu:~/Desktop/submit/project3/Part1$ make
gcc -c Multithreaded.c
gcc -o Multithreaded Multithreaded.o -lpthread
psyduckliu@ubuntu:~/Desktop/submit/project3/Part1$ ./Multithreaded
Please enter the number of array elements:
10
Please enter the array elements:
9 1 0 2 8 7 5 3 4 6
The sorted array is:
0 1 2 3 4 5 6 7 8 9
psyduckliu@ubuntu:~/Desktop/submit/project3/Part1$ ./Multithreaded
Please enter the number of array elements:
10
Please enter the array elements:
7 12 19 3 18 4 2 6 15 8
The sorted array is:
2 3 4 6 7 8 12 15 18 19
```

图 1: Part 1-Multithreaded Sorting Application

3.2 Part 2-Fork-Join Sorting Application-MergeSort

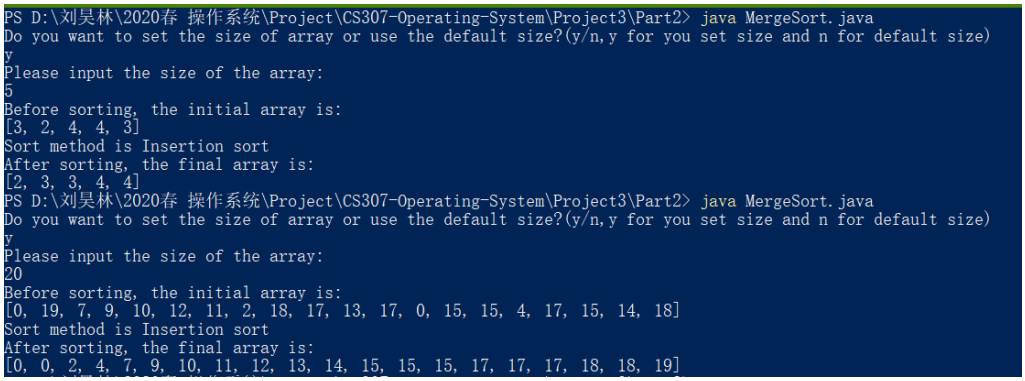

图 2: Fork-Join Sorting Application-MergeSort(1)

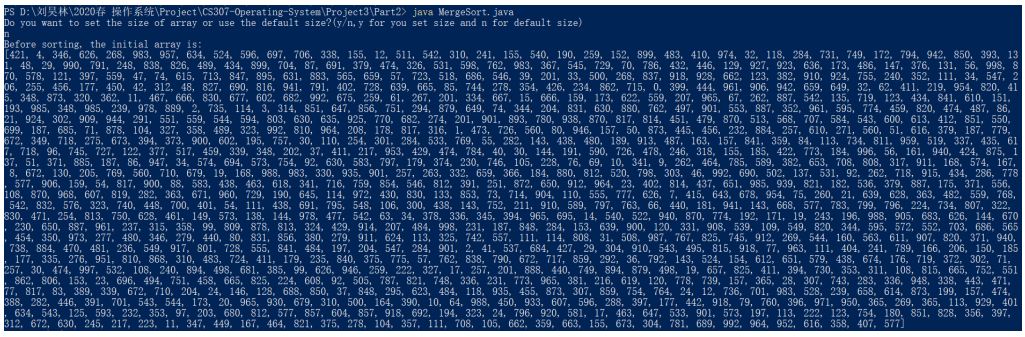

图 3: Fork-Join Sorting Application-MergeSort(2)

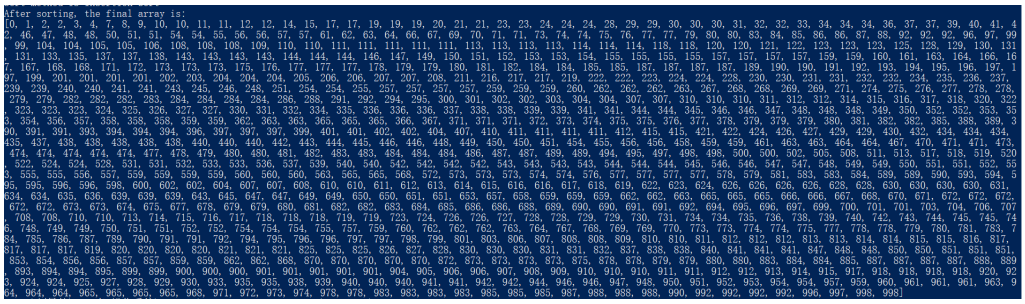

图 4: Fork-Join Sorting Application-MergeSort(3)

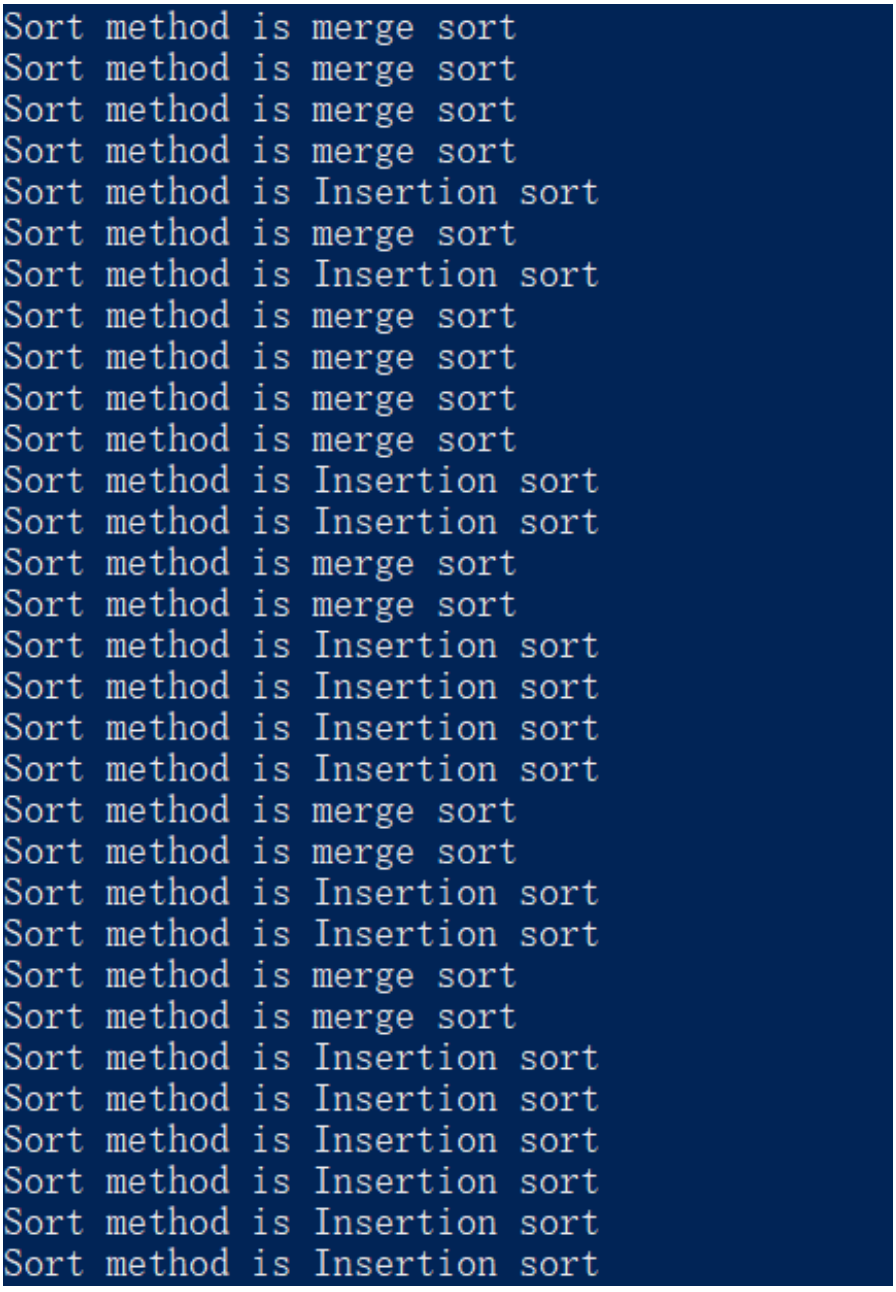

图 5: Fork-Join Sorting Application-MergeSort(4)

3.3 Part 2-Fork-Join Sorting Application-QuickSort

```
PS D:\刘昊林\2020春 操作系统\Project\CS307-Operating-System\Project3\Part2> java QuickSort.java
Do you want to set the size of array or use the default size?(y/n,y for you set size and n for default size)
y
Please input the size of the array:
5
Before sorting, the initial array is:
[0, 3, 4, 3, 2]
Sort method is Insertion sort
After sorting, the final array is:
[0, 2, 3, 3, 4]
PS D:\刘昊林\2020春 操作系统\Project\CS307-Operating-System\Project3\Part2> java QuickSort.java
Do you want to set the size of array or use the default size?(y/n,y for you set size and n for default size)
y
Please input the size of the array:
20
Before sorting, the initial array is:
[14, 2, 6, 13, 4, 11, 8, 7, 10, 8, 4, 10, 3, 13, 12, 12, 7, 1, 4, 3]
Sort method is Insertion sort
After sorting, the final array is:
[1, 2, 3, 3, 4, 4, 4, 6, 7, 7, 8, 8, 10, 10, 11, 12, 12, 13, 13, 14]
```

图 6: Fork-Join Sorting Application-QuickSort(1)

```
PS D:\刘昊林\2020春 操作系统\Project\CS307-Operating-System\Project3\Part2> java QuickSort.java
Do you want to set the size of array or use the default size?(y/n,y for you set size and n for default size)
n
Before sorting, the initial array is:
[453, 125, 462, 676, 438, 875, 396, 426, 954, 656, 189, 295, 123, 824, 114, 593, 996, 297, 80, 854, 343, 532, 138, 998, 502, 552, 516, 10, 733, 475, 895, 727, 571, 958, 528, 347, 890, 476,
948, 907, 884, 854, 110, 30, 493, 878, 296, 162, 96, 886, 440, 785, 922, 669, 193, 167, 538, 973, 515, 843, 338, 972, 365, 704, 919, 556, 639, 86, 243, 721, 236, 962, 232, 263, 580, 55, 518
, 286, 295, 55, 659, 923, 29, 562, 867, 252, 387, 889, 462, 871, 497, 340, 774, 991, 894, 622, 82, 637, 812, 338, 373, 519, 553, 371, 694, 215, 206, 212, 424, 963, 996, 971, 769, 36, 229, 5
25, 887, 717, 444, 752, 630, 962, 855, 603, 679, 730, 760, 206, 515, 859, 798, 682, 725, 817, 957, 252, 349, 650, 741, 11, 968, 41, 137, 475, 813, 492, 530, 422, 394, 685, 870, 927, 909, 18
6, 624, 450, 360, 777, 819, 923, 939, 747, 199, 281, 13, 555, 543, 463, 125, 484, 641, 823, 50, 444, 812, 603, 683, 577, 280, 226, 217, 823, 79, 979, 288, 523, 314, 792, 732, 718, 542, 106,
857, 913, 573, 589, 50, 386, 640, 912, 867, 393, 810, 492, 778, 161, 64, 830, 616, 126, 997, 299, 918, 647, 826, 423, 762, 488, 722, 674, 969, 64, 676, 103, 598, 481, 242, 905, 369, 577, 6
2, 949, 11, 4, 204, 615, 917, 572, 626, 394, 864, 694, 971, 743, 199, 50, 132, 216, 848, 399, 174, 458, 90, 16, 634, 693, 57, 223, 210, 858, 316, 517, 804, 464, 352, 619, 914, 466, 77, 231,
638, 24, 470, 831, 872, 17, 718, 635, 631, 234, 854, 274, 728, 594, 619, 84, 282, 73, 148, 63, 759, 722, 137, 217, 798, 493, 614, 224, 297, 235, 239, 769, 820, 519, 639, 35, 453, 925, 129,
447, 1, 680, 55, 599, 821, 496, 184, 896, 859, 807, 187, 661, 783, 223, 986, 417, 817, 658, 835, 688, 805, 804, 832, 363, 278, 945, 334, 415, 265, 119, 882, 193, 109, 895, 759, 984, 169, 6
22, 365, 734, 766, 642, 523, 532, 355, 724, 229, 578, 496, 969, 8, 8, 306, 179, 322, 99, 860, 570, 935, 495, 703, 947, 767, 342, 112, 331, 219, 226, 480, 835, 654, 341, 17, 55, 728, 362, 89
9, 788, 668, 583, 215, 422, 500, 730, 291, 51, 779, 717, 868, 376, 776, 607, 874, 838, 50, 932, 349, 96, 7, 465, 205, 315, 255, 726, 743, 658, 525, 625, 771, 597, 61, 981, 322, 790, 656, 26
7, 445, 545, 293, 364, 553, 805, 323, 731, 999, 730, 499, 662, 296, 793, 718, 761, 29, 530, 348, 15, 108, 463, 150, 463, 566, 971, 267, 527, 878, 135, 483, 551, 284, 271, 263, 748, 500, 230
, 412, 439, 266, 973, 759, 121, 781, 986, 196, 590, 228, 797, 193, 836, 292, 289, 688, 678, 370, 952, 569, 377, 614, 999, 150, 282, 268, 747, 696, 865, 747, 62, 312, 471, 799, 338, 375, 760
, 329, 481, 326, 420, 766, 126, 994, 200, 172, 212, 247, 819, 841, 650, 101, 824, 118, 47, 120, 296, 521, 639, 929, 849, 56, 986, 834, 6, 505, 149, 471, 644, 639, 944, 568, 375, 78, 987, 41
9, 471, 71, 750, 942, 794, 632, 395, 674, 679, 419, 418, 731, 804, 782, 505, 52, 853, 826, 726, 391, 794, 741, 458, 554, 555, 83, 108, 893, 754, 721, 223, 248, 638, 154, 361, 890, 511, 162,
693, 771, 963, 848, 707, 403, 828, 24, 764, 100, 419, 566, 397, 282, 861, 911, 110, 692, 624, 801, 832, 58, 995, 523, 606, 309, 504, 55, 649, 302, 814, 304, 695, 302, 535, 361, 264, 493, 6
66, 779, 628, 823, 758, 231, 524, 96, 677, 291, 851, 764, 832, 942, 521, 498, 540, 896, 69, 430, 945, 748, 867, 830, 443, 827, 25, 454, 271, 434, 866, 165, 916, 651, 102, 696, 35, 63, 906,
229, 652, 620, 799, 468, 310, 373, 755, 42, 619, 948, 102, 919, 550, 477, 947, 888, 66, 369, 924, 903, 240, 64, 480, 326, 4, 779, 284, 858, 53, 720, 609, 868, 95, 624, 189, 867, 911, 426, 7
37, 357, 354, 458, 950, 637, 338, 148, 133, 638, 318, 498, 682, 349, 885, 990, 84, 764, 143, 841, 804, 19, 992, 337, 327, 393, 802, 113, 185, 681, 934, 114, 496, 456, 190, 655, 618, 962, 44
2, 447, 420, 20, 101, 172, 201, 884, 987, 784, 171, 736, 337, 943, 256, 574, 663, 71, 236, 1, 393, 852, 975, 970, 767, 136, 659, 694, 523, 503, 909, 523, 513, 529, 149, 671, 473, 505, 555,
688, 149, 754, 136, 988, 567, 170, 662, 271, 4, 654, 277, 392, 104, 4, 868, 426, 927, 840, 632, 772, 309, 86, 43, 877, 628, 662, 500, 242, 272, 718, 662, 133, 952, 713, 112, 657, 970, 373,
634, 922, 123, 275, 22, 273, 507, 511, 378, 949, 177, 847, 233, 877, 752, 968, 304, 482, 437, 747, 212, 138, 268, 879, 717, 471, 574, 912, 807, 254, 153, 38, 670, 394, 916, 728, 299, 281, 3
90, 287, 23, 775, 938, 919, 258, 942, 968, 240, 561, 631, 707, 963, 266, 890, 50, 720, 973, 210, 366, 140, 953, 212, 599, 319, 351, 453, 35, 947, 687, 123, 916, 537, 454, 684, 69, 334, 393,
823, 377, 379, 112, 489, 280, 762, 897, 655, 446, 68, 696, 186, 249, 447, 129, 408, 208, 397, 306, 822, 883, 392, 207, 436, 947, 409, 835, 632, 462, 608, 631, 607, 662, 421, 467, 970, 120,
397, 703, 405, 668, 73, 876, 635, 792, 871, 659, 501, 831, 948, 593, 857, 704, 465, 7, 751, 989, 114, 266, 55, 434, 887, 931, 342, 517, 833, 501, 976, 559, 566, 949, 71, 56, 616, 475, 129,
270, 135, 293, 392, 690, 447, 106, 498, 199, 964, 378, 753, 255, 634, 319, 725, 444, 73, 903, 397, 360, 893, 841, 685, 506, 67, 866, 314, 374, 472, 475, 736, 319, 868, 193]
```

图 7: Fork-Join Sorting Application-QuickSort(2)

```
After sorting, the final array is:
[1, 1, 4, 4, 4, 4, 6, 7, 7, 8, 8, 10, 11, 11, 13, 15, 16, 17, 17, 19, 20, 22, 23, 24, 24, 25, 29, 29, 30, 35, 35, 35, 36, 38, 41, 42, 43, 47, 50, 50, 50, 50, 50, 51, 52, 53, 55, 55, 55, 55,
55, 55, 56, 56, 57, 58, 61, 62, 62, 63, 64, 64, 64, 65, 66, 67, 68, 69, 69, 71, 71, 71, 72, 73, 73, 77, 78, 79, 80, 82, 83, 84, 84, 86, 86, 90, 95, 96, 96, 99, 100, 101, 101, 102, 102,
103, 104, 106, 106, 108, 108, 109, 110, 110, 112, 112, 112, 113, 114, 114, 114, 118, 119, 120, 120, 121, 123, 123, 123, 125, 125, 125, 126, 126, 129, 129, 129, 132, 133, 133, 135, 136, 136
, 137, 137, 138, 138, 140, 143, 148, 148, 149, 149, 149, 150, 150, 153, 154, 161, 162, 162, 165, 167, 169, 170, 171, 172, 172, 174, 177, 179, 184, 185, 186, 186, 187, 189, 189, 190, 193, 19
3, 193, 193, 196, 199, 199, 199, 200, 200, 201, 204, 206, 206, 207, 208, 210, 210, 212, 212, 213, 214, 215, 215, 215, 216, 217, 217, 219, 223, 223, 223, 224, 225, 226, 228, 229, 229, 229, 2
30, 231, 231, 233, 234, 235, 236, 236, 239, 240, 240, 240, 242, 242, 243, 247, 248, 249, 252, 252, 252, 254, 255, 255, 256, 258, 259, 262, 263, 264, 264, 265, 266, 266, 266, 267, 267, 268,
268, 270, 271, 271, 271, 272, 273, 274, 275, 277, 278, 280, 280, 281, 281, 282, 282, 282, 284, 284, 286, 287, 288, 289, 291, 291, 292, 293, 293, 295, 295, 296, 296, 296, 297, 297, 299, 299,
302, 302, 304, 306, 306, 309, 309, 310, 312, 314, 314, 315, 316, 318, 319, 319, 319, 322, 322, 323, 326, 326, 327, 329, 331, 334, 334, 337, 337, 338, 338, 338, 341, 342, 342, 343, 347, 348
, 349, 349, 351, 352, 354, 355, 357, 360, 360, 361, 361, 362, 363, 364, 365, 365, 366, 369, 369, 370, 371, 373, 373, 373, 374, 375, 376, 377, 377, 378, 379, 386, 387, 390, 391, 392, 39
2, 392, 393, 393, 393, 393, 394, 394, 394, 395, 396, 397, 397, 397, 397, 403, 405, 408, 409, 409, 412, 415, 417, 418, 419, 419, 419, 420, 420, 421, 422, 422, 423, 424, 426, 426, 426, 430, 4
34, 434, 436, 437, 438, 439, 440, 442, 443, 444, 444, 444, 445, 446, 447, 447, 447, 447, 450, 453, 453, 453, 454, 454, 456, 458, 458, 458, 462, 462, 462, 463, 463, 463, 464, 465, 465, 466,
467, 468, 470, 471, 471, 471, 471, 472, 473, 473, 475, 475, 475, 476, 477, 480, 480, 481, 481, 482, 483, 484, 488, 489, 492, 492, 493, 493, 493, 493, 496, 496, 496, 497, 498, 498, 498, 500,
500, 501, 501, 502, 503, 504, 505, 505, 505, 506, 507, 509, 511, 511, 513, 515, 515, 515, 516, 517, 517, 518, 519, 519, 521, 521, 523, 523, 523, 523, 523, 523, 524, 525, 525, 527, 528, 529, 530
, 530, 532, 532, 535, 537, 538, 538, 540, 542, 543, 545, 550, 551, 552, 553, 553, 554, 555, 555, 555, 556, 559, 561, 562, 566, 566, 566, 567, 568, 569, 570, 571, 572, 573, 574, 574, 575, 57
7, 577, 578, 578, 580, 583, 589, 590, 593, 593, 597, 598, 599, 599, 599, 603, 603, 606, 607, 607, 608, 609, 614, 614, 615, 616, 616, 618, 619, 619, 619, 620, 622, 622, 623, 624, 624, 624, 6
28, 626, 628, 628, 628, 630, 631, 631, 631, 632, 632, 632, 634, 634, 634, 635, 635, 637, 637, 638, 638, 639, 639, 639, 639, 640, 641, 642, 644, 647, 649, 650, 650, 651, 652, 654, 654, 655,
655, 656, 656, 657, 658, 658, 659, 659, 659, 661, 662, 662, 662, 662, 662, 662, 665, 666, 668, 668, 669, 670, 671, 674, 674, 676, 676, 677, 678, 679, 679, 680, 681, 682, 682, 683, 684, 685, 685,
687, 688, 688, 688, 690, 692, 693, 694, 694, 694, 695, 696, 696, 696, 703, 703, 704, 704, 706, 707, 707, 713, 717, 717, 717, 718, 718, 718, 718, 720, 720, 721, 721, 722, 722, 724, 725, 725
, 726, 726, 727, 728, 728, 728, 730, 730, 730, 731, 731, 732, 733, 734, 736, 737, 741, 741, 743, 743, 747, 747, 747, 747, 748, 748, 750, 751, 752, 752, 753, 754, 754, 755, 756, 758, 759, 75
9, 759, 760, 761, 762, 762, 764, 764, 766, 766, 767, 767, 769, 769, 771, 771, 772, 774, 775, 776, 777, 778, 779, 779, 779, 781, 782, 782, 783, 784, 785, 788, 790, 792, 793, 794, 794, 797, 7
98, 798, 799, 799, 801, 802, 804, 804, 804, 804, 805, 805, 806, 807, 807, 810, 812, 812, 813, 814, 817, 817, 819, 819, 820, 821, 822, 823, 823, 824, 824, 825, 826, 826, 827, 828, 830, 830,
831, 831, 832, 832, 832, 833, 834, 835, 835, 835, 836, 838, 840, 841, 841, 841, 843, 847, 848, 848, 849, 851, 852, 853, 854, 854, 854, 857, 857, 858, 858, 859, 859, 860, 861, 864, 865, 866,
866, 867, 867, 867, 867, 868, 868, 868, 868, 870, 871, 871, 872, 874, 875, 876, 877, 877, 878, 878, 879, 882, 883, 884, 884, 885, 886, 887, 887, 888, 889, 890, 890, 893, 893, 894, 895
, 895, 896, 896, 897, 899, 903, 903, 905, 907, 909, 909, 911, 911, 912, 912, 913, 914, 916, 916, 916, 917, 918, 919, 919, 919, 922, 922, 923, 923, 924, 925, 927, 927, 929, 931, 934, 935, 93
8, 939, 942, 942, 942, 943, 944, 945, 945, 947, 947, 947, 947, 948, 948, 948, 949, 949, 949, 950, 952, 952, 952, 953, 954, 957, 958, 962, 962, 962, 963, 963, 963, 964, 968, 968, 968, 969, 9
69, 970, 970, 970, 971, 971, 971, 971, 972, 973, 973, 973, 975, 976, 979, 981, 984, 986, 986, 986, 986, 987, 987, 989, 990, 991, 992, 993, 994, 995, 996, 996, 997, 998, 999, 999]
```

图 8: Fork-Join Sorting Application-QuickSort(3)

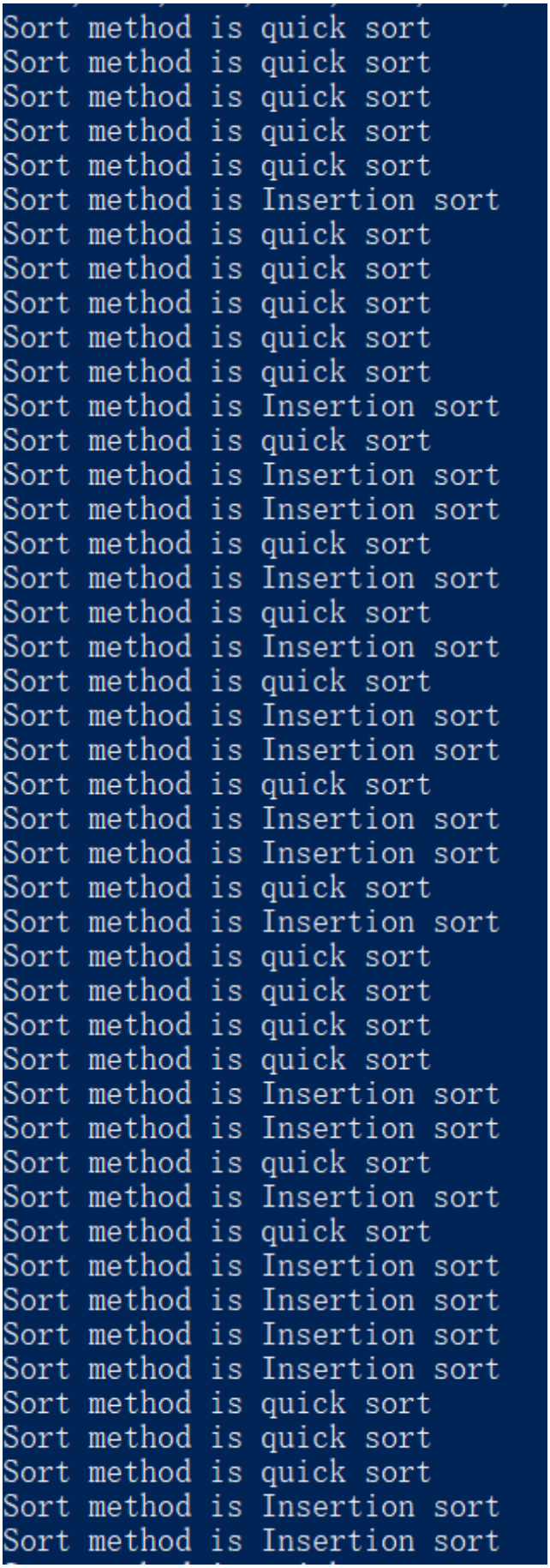

图 9: Fork-Join Sorting Application-QuickSort(4) 仅仅是排序过程中的一小部分!!