

Project 3: Multithreaded Sorting Application and Fork-Join Sorting Application

CS307-Operating System (D), CS356-Operating System Course Design, Chentao Wu, Spring 2020.

Name: 方泓杰(Hongjie Fang) Student ID: 518030910150 Email: galaxies@sjtu.edu.cn

1 Multithreaded Sorting Application

Write a multithreaded sorting program that works as follows: A list of integers is divided into two smaller lists of equal size. Two separate threads (which we will term *sorting threads*) sort each sublist using a sorting algorithm of your choice. The two sublists are merged by a third thread - a *merging thread* - which merges the two sublists into a single sorted list.

Because global data are shared across all threads, perhaps the easiest way to set up the data is to create a global array. Each sorting thread will work on one half of this array. A second global array of the same size as the unsorted integer array will also be established. The merging thread will then merge the two sublists into this second array. Graphically, this program is structured as follows (Fig. 1).

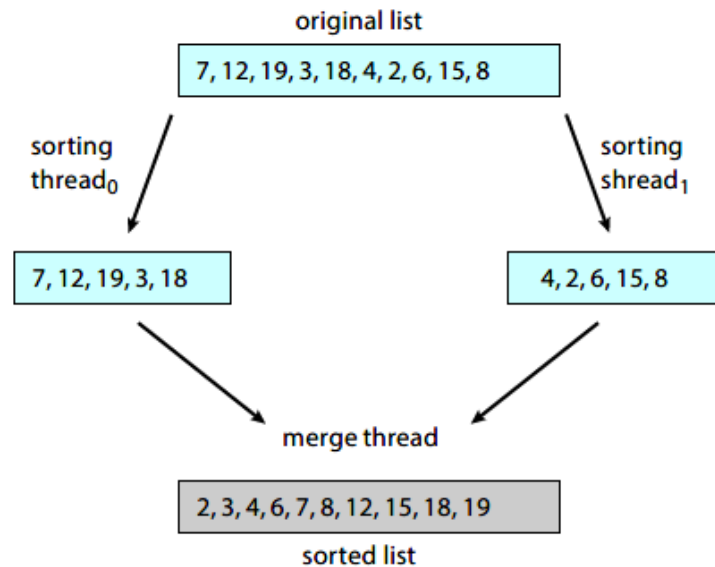


Figure 1: Multithreaded sorting

This programming project will require passing parameters to each of the sorting thread. In particular, it will be necessary to identify the starting index from which each thread is to begin sorting. Refer to the instructions in the last project for details on passing parameters to a thread.

The parent thread will output the sorted array once all sorting threads have exited.

Solution. Here is my methods to implement the multithreaded sorting program (`multithreaded-sort.c`).

- There are two global arrays `arr` and `res`. `arr` stores the initial array and the result of sorting threads; and `res` stores the result of merging thread, which is also the final result array.
- Suppose n is the length of the array. We pass the parameters (`begin`, `end`) to the sorting threads, which indicates that this thread is used to sort the sub-array `arr[begin ... end]`.
 - We pass the parameters (`0`, `n/2`) to sorting thread 0;
 - We pass the parameters (`n/2 + 1`, `n - 1`) to sorting thread 1.

- In each sorting thread, we use the Quicksort to sort the sub-array. The Quicksort algorithm is provided in the `<stdlib.h>` library, so we can call it directly using `qsort(...)` instruction.
- We pass the parameters (`begin`, `mid`, `end`) to the merging thread, which indicates the two sorted subarrays are `arr[begin ... mid]` and `array[mid + 1 ... end]`. Then we use the merge algorithm in the Mergesort to merge two sorted array into one sorted array `res`.

Here is the specific implementation of multithreaded sort (`multithreaded-sort.c`).

```

1 // Note: remember to add '-g -lpthread' when compiling.
2 # include <stdio.h>
3 # include <pthread.h>
4 # include <stdlib.h>
5
6 // the size of the array.
7 int n;
8 // original array.
9 int *arr;
10 // result array.
11 int *res;
12
13 struct parameters {
14     // the sorting thread will sort arr[begin ... end]
15     int begin, end;
16 };
17
18 struct merge_parameters {
19     // the merging thread will merge arr[begin ... mid] and arr[mid + 1, end]
20     int begin, mid, end;
21 };
22
23 int qsort_cmp(const void *a, const void *b) {
24     return * (int *) a - * (int *) b;
25 }
26
27 void* sorting_routine(void *arg) {
28     struct parameters *data = (struct parameters *) arg;
29
30     if (data -> end - data -> begin < 0) return NULL;
31
32     // Quick sort provided by <stdlib.h>.
33     qsort(arr + data -> begin, data -> end - data -> begin + 1, sizeof(int),
34           qsort_cmp);
35
36     // Return.
37     return NULL;
38 }
39
40 void* merging_routine(void *arg) {
41     struct merge_parameters *data = (struct merge_parameters *) arg;
42     // Merge two sorted arrays.
43     // |-- pos of result

```

```

43 int res_pos = data -> begin;
44 // |-- pos of two arrays.
45 int pos0 = data -> begin, pos1 = data -> mid + 1;
46
47 // |-- comparing & merging.
48 while (pos0 <= data -> mid && pos1 <= data -> end) {
49     if (arr[pos0] <= arr[pos1]) res[res_pos++] = arr[pos0++];
50     else res[res_pos++] = arr[pos1++];
51 }
52 // |-- the rest of the elements.
53 while (pos0 <= data -> mid)
54     res[res_pos++] = arr[pos0++];
55 while (pos1 <= data -> end)
56     res[res_pos++] = arr[pos1++];
57
58 // Check the result .
59 if (res_pos != data -> end + 1) {
60     printf("Error: unexpected error occurs when merging.\n");
61     exit(1);
62 }
63
64 // Return.
65 return NULL;
66 }
67
68 int main(void) {
69     int err;
70
71     // Input
72     printf("Please input the length of the array (0 <= n <= 1000000): ");
73     scanf("%d", &n);
74
75     if(n < 0 || n > 1000000) {
76         printf("Error: n should be in range [0, 1000000]!\n");
77         exit(1);
78     }
79
80     // Allocate spaces for arrays.
81     arr = (int *) malloc (n * sizeof(int));
82     res = (int *) malloc (n * sizeof(int));
83
84     // Input the elements or Generate the elements.
85     char opt[5];
86     printf("Do you want to generate the random elements automatically (y/n): ");
87     scanf("%s", opt);
88     if(opt[0] == 'y') {
89         for (int i = 0; i < n; ++ i)
90             arr[i] = rand() % 1000;
91         printf("The original array: \n");
92         for (int i = 0; i < n; ++ i)

```

```

93     printf("%d ", arr[i]);
94     printf("\n");
95 } else if (opt[0] == 'n') {
96     printf("Please input the array elements: \n");
97     for (int i = 0; i < n; ++ i)
98         scanf("%d", &arr[i]);
99 } else {
100     printf("Error: invalid input!\n");
101     exit(1);
102 }
103
104 // Prepare parameters for sorting threads.
105 struct parameters param[2];
106 // |-- thread 0 parameters
107 param[0].begin = 0;
108 param[0].end = n / 2;
109 // |-- thread 1 parameters
110 param[1].begin = n / 2 + 1;
111 param[1].end = n - 1;
112
113 // Create sorting threads & passing parameters.
114 pthread_t sorting_thread[2];
115 for (int i = 0; i < 2; ++ i) {
116     err = pthread_create(&sorting_thread[i], NULL, sorting_routine, &param[i]);
117     if (err) {
118         printf("Error: create thread failed!\n");
119         exit(1);
120     }
121 }
122
123 // Sorting threads end.
124 void *output;
125 for (int i = 0; i < 2; ++ i) {
126     err = pthread_join(sorting_thread[i], &output);
127     if (err) {
128         printf("Error: thread join failed!\n");
129         exit(1);
130     }
131 }
132
133 // Prepare parameters for merging thread.
134 struct merge_parameters m_param;
135 m_param.begin = 0;
136 m_param.mid = n / 2;
137 m_param.end = n - 1;
138
139 // Create merging thread & passing parameters.
140 pthread_t merging_thread;
141 err = pthread_create(&merging_thread, NULL, merging_routine, &m_param);
142 if (err) {

```

```

143     printf("Error: create thread failed!\n");
144     exit(1);
145 }
146
147 // Merging thread end.
148 err = pthread_join(merging_thread, &output);
149 if (err) {
150     printf("Error: thread join failed!\n");
151     exit(1);
152 }
153
154 // Print the result (Output).
155 printf("The array after sorting: \n");
156 for (int i = 0; i < n; ++ i)
157     printf("%d ", res[i]);
158 printf("\n");
159
160 // Release the spaces.
161 free(arr);
162 free(res);
163
164 return 0;
165 }

```

We can test the multithreaded program by entering the following instructions. Rememeber to add `-g -lpthread` because we need to use `pthread` API.

```

1 gcc multithreaded-sort.c -o ./sort -g -lpthread
2 ./sort

```

Thus, you can enter the array or use the random array generated automatically to test the multithreaded sorting program.

Here are some testing examples (Fig. 2).

```

galaxies@ubuntu:~/CS307-Projects/Project3/multithreaded-sort$ gcc multithreaded-sort.c -o ./sort -g -lpthread
galaxies@ubuntu:~/CS307-Projects/Project3/multithreaded-sort$ ./sort
Please input the length of the array (0 <= n <= 1000000): 100
Do you want to generate the random elements automatically (y/n): y
The original array:
383 886 777 915 793 335 386 492 649 421 362 27 690 59 763 926 540 426 172 736 211 368 567 429 782 530 862 123 67 1
35 929 802 22 58 69 167 393 456 11 42 229 373 421 919 784 537 198 324 315 370 413 526 91 980 956 873 862 170 996 2
81 305 925 84 327 336 505 846 729 313 857 124 895 582 545 814 367 434 364 43 750 87 808 276 178 788 584 403 651 75
4 399 932 60 676 368 739 12 226 586 94 539
The array after sorting:
11 12 22 27 42 43 58 59 60 67 69 84 87 91 94 123 124 135 167 170 172 178 198 211 226 229 276 281 305 313 315 324 3
27 335 336 362 364 367 368 370 373 383 386 393 399 403 413 421 426 429 434 456 492 505 526 530 537 539 540
545 567 582 584 586 649 651 676 690 729 736 739 750 754 763 777 782 784 788 793 802 808 814 846 857 862 862 873 8
86 895 915 919 925 926 929 932 956 980 996
galaxies@ubuntu:~/CS307-Projects/Project3/multithreaded-sort$ ./sort
Please input the length of the array (0 <= n <= 1000000): 20
Do you want to generate the random elements automatically (y/n): n
Please input the array elements:
2 5 8 0 3 6 9 10 28 27 19 27 77 55 44 33 2 0 19 -2
The array after sorting:
-2 0 0 2 2 3 5 6 8 9 10 19 19 27 27 28 33 44 55 77

```

Figure 2: Testing examples of the multithreaded sorting program

□

2 Fork-Join Sorting Application

Implement the preceding project (Multithreaded Sorting Application) using Java's fork-join parallelism API. This project will be developed in two different versions. Each version will implement a different divide-and-conquer sorting algorithm: Quicksort, Mergesort.

The Quicksort implementation will use the Quicksort algorithm for dividing the list of elements to be sorted into a *left half* and a *right half* based on the position of the `pivot` value. The Mergesort algorithm will divide the list into two evenly sized halves. For both the Quicksort and Mergesort algorithms, when the list to be sorted falls within some threshold value (for example, the list is size 100 or fewer), directly apply a simple algorithms such as the Selection or Insertion sort. Most data structures texts describe these two well-know, divide-and-conquer sorting algorithms.

The class `SumTask` shwon in Section 4.5.2.1 extends `RecursiveTask`, which is a result-bearing `ForkJoinTask`. As this assignment will involve sorting the array that is passed to the task, but not returning any values, you will instead create a class that extends `RecursiveAction`, a non result-bearing `ForkJoinTask`.

The objects passed to each sorting algorithms are required to implement Java's `Comparable` interface, and this will need to be reflected in the class definition for each sorting algorithm. The source code download for this text includes Java code that provides the foundation for beginning this project.

Solution. Here is my methods to implement the Fork-Join sorting program (`Mergesort.java` and `Quicksort.java`).

- We use the `RecursiveAction` in the `ForkJoinTask` to implement the sorting process;
- We implement the sorting process in `compute()` function. We do not discuss the details of the sort algorithms here and you can refer to the source codes yourself.
- We use the java class `Integer` as the object type. Our object is required to implement `Comparable` interface because we use the `compareTo()` function in the `Comparable` interface, instead of the usual comparators `<`, `<=`, `>`, `>=`,

Here is the specific implementation of the fork-join Mergesort version (`Mergesort.java`).

```
1 import java.util.Arrays;
2 import java.util.Scanner;
3 import java.util.concurrent.*;
4
5 public class Mergesort extends RecursiveAction {
6     static final int THRESHOLD = 10;
7
8     private int begin;
9     private int end;
10    private Integer[] array;
11
12    public Mergesort(int begin, int end, Integer[] array) {
13        this.begin = begin;
14        this.end = end;
15        this.array = array;
16    }
17
18    protected void compute() {
```

```

19     if (end - begin + 1 <= THRESHOLD) {
20         // Sort directly using Bubble Sort
21         for (int i = end; i >= begin + 1; -- i)
22             for (int j = begin; j < i; ++ j)
23                 if (array[j].compareTo(array[j + 1]) > 0) {
24                     Integer temp = array[j];
25                     array[j] = array[j + 1];
26                     array[j + 1] = temp;
27                 }
28     } else {
29         // divide stage
30         int mid = begin + (end - begin) / 2;
31
32         Mergesort leftTask = new Mergesort(begin, mid, array);
33         Mergesort rightTask = new Mergesort(mid + 1, end, array);
34
35         leftTask.fork();
36         rightTask.fork();
37
38         leftTask.join();
39         rightTask.join();
40
41         Integer[] temp = new Integer [end - begin + 1];
42
43         int pos1 = begin, pos2 = mid + 1, k = 0;
44         while (pos1 <= mid && pos2 <= end) {
45             if (array[pos1].compareTo(array[pos2]) <= 0) temp[k ++] = array[pos1 ++];
46             else temp[k ++] = array[pos2 ++];
47         }
48         while (pos1 <= mid) temp[k ++] = array[pos1 ++];
49         while (pos2 <= end) temp[k ++] = array[pos2 ++];
50
51         for (int i = 0; i < k; ++ i)
52             array[i + begin] = temp[i];
53     }
54 }
55
56 public static void main(String[] args) {
57     ForkJoinPool pool = new ForkJoinPool();
58     Scanner sc = new Scanner(System.in);
59
60     System.out.print("Please input the length of the array (0 <= n <= 1000000): ")
61         ;
62     int n = sc.nextInt();
63     if (n < 0 || n > 1000000) {
64         System.out.println("Error: n should be in range [0, 1000000]!");
65         System.exit(1);
66     }
67
68     Integer[] array = new Integer[n];

```

```

68
69 System.out.print("Do you want to generate the random elements automatically (y
    /n): ");
70 char opt = sc.next().charAt(0);
71 if (opt == 'y') {
72     // create SIZE random integers between 0 and 999
73     java.util.Random rand = new java.util.Random();
74     for (int i = 0; i < n; ++ i)
75         array[i] = rand.nextInt(1000);
76     System.out.print("The original array: ");
77     System.out.println(Arrays.toString(array));
78 } else if (opt == 'n') {
79     System.out.println("Please input the array elements: ");
80     for (int i = 0; i < n; ++ i)
81         array[i] = sc.nextInt();
82 } else {
83     System.out.println("Error: invalid input!");
84     System.exit(1);
85 }
86
87 // use fork-join parallelism to sum the array
88 Mergesort task = new Mergesort(0, n - 1, array);
89
90 pool.invoke(task);
91
92 System.out.print("The array after sorting: ");
93 System.out.println(Arrays.toString(array));
94 }
95 }

```

We can test the fork-join Mergesort program by entering the following instructions.

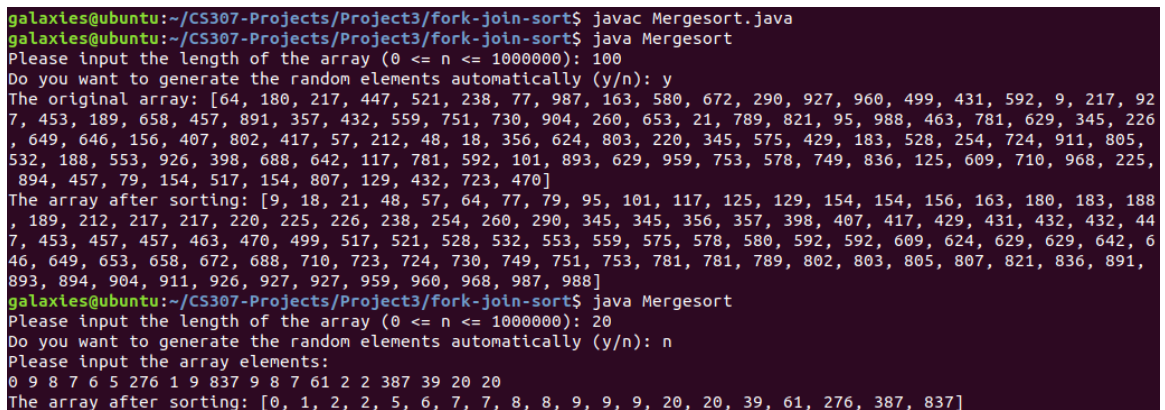
```

1 javac Mergesort.java
2 java Mergesort

```

Thus, you can enter the array or use the random array generated automatically to test the fork-join Mergesort program.

Here are some testing examples (Fig. 3).



```

galaxies@ubuntu:~/CS307-Projects/Project3/fork-join-sort$ javac Mergesort.java
galaxies@ubuntu:~/CS307-Projects/Project3/fork-join-sort$ java Mergesort
Please input the length of the array (0 <= n <= 1000000): 100
Do you want to generate the random elements automatically (y/n): y
The original array: [64, 180, 217, 447, 521, 238, 77, 987, 163, 580, 672, 290, 927, 960, 499, 431, 592, 9, 217, 92
7, 453, 189, 658, 457, 891, 357, 432, 559, 751, 730, 904, 260, 653, 21, 789, 821, 95, 988, 463, 781, 629, 345, 226
, 649, 646, 156, 407, 802, 417, 57, 212, 48, 18, 356, 624, 803, 220, 345, 575, 429, 183, 528, 254, 724, 911, 805,
532, 188, 553, 926, 398, 688, 642, 117, 781, 592, 101, 893, 629, 959, 753, 578, 749, 836, 125, 609, 710, 968, 225,
894, 457, 79, 154, 517, 154, 807, 129, 432, 723, 470]
The array after sorting: [9, 18, 21, 48, 57, 64, 77, 79, 95, 101, 117, 125, 129, 154, 154, 156, 163, 180, 183, 188
, 189, 212, 217, 217, 220, 225, 226, 238, 254, 260, 290, 345, 345, 356, 357, 398, 407, 417, 429, 431, 432, 432, 44
7, 453, 457, 457, 463, 470, 499, 517, 521, 528, 532, 553, 559, 575, 578, 580, 592, 592, 609, 624, 629, 629, 642, 6
46, 649, 653, 658, 672, 688, 710, 723, 724, 730, 749, 751, 753, 781, 781, 789, 802, 803, 805, 807, 821, 836, 891,
893, 894, 904, 911, 926, 927, 927, 959, 960, 968, 987, 988]
galaxies@ubuntu:~/CS307-Projects/Project3/fork-join-sort$ java Mergesort
Please input the length of the array (0 <= n <= 1000000): 20
Do you want to generate the random elements automatically (y/n): n
Please input the array elements:
0 9 8 7 6 5 276 1 9 837 9 8 7 61 2 2 387 39 20 20
The array after sorting: [0, 1, 2, 2, 5, 6, 7, 7, 8, 8, 9, 9, 9, 20, 20, 39, 61, 276, 387, 837]

```

Figure 3: Testing examples of the fork-join Mergesort program

Here is the specific implementation of the fork-join Quicksort version (Quicksort.java).

```
1 import java.util.Arrays;
2 import java.util.Scanner;
3 import java.util.concurrent.*;
4
5 public class Quicksort extends RecursiveAction {
6     static final int THRESHOLD = 10;
7
8     private int begin;
9     private int end;
10    private Integer[] array;
11
12    public Quicksort(int begin, int end, Integer[] array) {
13        this.begin = begin;
14        this.end = end;
15        this.array = array;
16    }
17
18    protected void compute() {
19        if (end - begin + 1 <= THRESHOLD) {
20            // Sort directly using Bubble Sort
21            for (int i = end; i >= begin + 1; -- i)
22                for (int j = begin; j < i; ++ j)
23                    if (array[j].compareTo(array[j + 1]) > 0) {
24                        Integer temp = array[j];
25                        array[j] = array[j + 1];
26                        array[j + 1] = temp;
27                    }
28        } else {
29            // divide stage
30            Integer pivot = array[begin];
31            int low = begin, high = end;
32            while (low < high) {
33                while (low < high && array[high].compareTo(pivot) >= 0) -- high;
34                if (low < high) array[low++] = array[high];
35                while (low < high && array[low].compareTo(pivot) <= 0) ++ low;
36                if (low < high) array[high--] = array[low];
37            }
38            array[low] = pivot;
39
40            Quicksort leftTask = new Quicksort(begin, low - 1, array);
41            Quicksort rightTask = new Quicksort(low + 1, end, array);
42
43            leftTask.fork();
44            rightTask.fork();
45
46            leftTask.join();
47            rightTask.join();
48        }
49    }
```

```

50
51 public static void main(String[] args) {
52     ForkJoinPool pool = new ForkJoinPool();
53     Scanner sc = new Scanner(System.in);
54
55     System.out.print("Please input the length of the array (0 <= n <= 1000000): ")
56         ;
57     int n = sc.nextInt();
58     if (n < 0 || n > 1000000) {
59         System.out.println("Error: n should be in range [0, 1000000]!");
60         System.exit(1);
61     }
62
63     Integer[] array = new Integer[n];
64
65     System.out.print("Do you want to generate the random elements automatically (y
66         /n): ");
67     char opt = sc.next().charAt(0);
68     if (opt == 'y') {
69         // create SIZE random integers between 0 and 999
70         java.util.Random rand = new java.util.Random();
71         for (int i = 0; i < n; ++ i)
72             array[i] = rand.nextInt(1000);
73         System.out.print("The original array: ");
74         System.out.println(Arrays.toString(array));
75     } else if (opt == 'n') {
76         System.out.println("Please input the array elements: ");
77         for (int i = 0; i < n; ++ i)
78             array[i] = sc.nextInt();
79     } else {
80         System.out.println("Error: invalid input!");
81         System.exit(1);
82     }
83
84     // use fork-join parallelism to sum the array
85     Quicksort task = new Quicksort(0, n - 1, array);
86
87     pool.invoke(task);
88
89     System.out.print("The array after sorting: ");
90     System.out.println(Arrays.toString(array));
91 }

```

We can test the fork-join Quicksort program by entering the following instructions.

```

1 javac Quicksort.java
2 java Quicksort

```

Thus, you can enter the array or use the random array generated automatically to test the fork-join Quicksort program.

Here are some testing examples (Fig. 4).

```
galaxies@ubuntu:~/CS307-Projects/Project3/fork-join-sort$ javac Quicksort.java
galaxies@ubuntu:~/CS307-Projects/Project3/fork-join-sort$ java Quicksort
Please input the length of the array (0 <= n <= 1000000): 100
Do you want to generate the random elements automatically (y/n): y
The original array: [960, 698, 579, 765, 253, 567, 366, 547, 120, 951, 783, 492, 307, 345, 281, 787, 681, 713, 745,
, 710, 343, 452, 389, 388, 493, 211, 803, 304, 784, 317, 292, 130, 855, 861, 411, 396, 256, 859, 742, 1, 919, 214,
0, 577, 138, 423, 512, 786, 28, 217, 721, 353, 596, 148, 647, 270, 153, 176, 523, 653, 292, 487, 899, 41, 154, 93
9, 337, 211, 565, 183, 326, 992, 830, 434, 59, 56, 225, 829, 692, 523, 20, 222, 354, 862, 989, 301, 995, 43, 344,
132, 991, 359, 522, 163, 218, 340, 788, 969, 185, 103]
The array after sorting: [0, 1, 20, 28, 41, 43, 56, 59, 103, 120, 130, 132, 138, 148, 153, 154, 163, 176, 183, 185
, 211, 211, 214, 217, 218, 222, 225, 253, 256, 270, 281, 292, 292, 301, 304, 307, 317, 326, 337, 340, 343, 344, 34
5, 353, 354, 359, 366, 388, 389, 396, 411, 423, 434, 452, 487, 492, 493, 512, 522, 523, 523, 547, 565, 567, 577, 5
79, 596, 647, 653, 681, 692, 698, 710, 713, 721, 742, 745, 765, 783, 784, 786, 787, 788, 803, 829, 830, 855, 859,
861, 862, 899, 919, 939, 951, 960, 969, 989, 991, 992, 995]
galaxies@ubuntu:~/CS307-Projects/Project3/fork-join-sort$ java Quicksort
Please input the length of the array (0 <= n <= 1000000): 20
Do you want to generate the random elements automatically (y/n): n
Please input the array elements:
9 8 7 62 298 7 6 52 10 9 88 77 66 55 44 2 9 70 71 999
The array after sorting: [2, 6, 7, 7, 8, 9, 9, 9, 10, 44, 52, 55, 62, 66, 70, 71, 77, 88, 298, 999]
```

Figure 4: Testing examples of the fork-join Quicksort program

□

3 Personal Thoughts

During the project, I've gained basic experiences about using the threads and writing a multithreaded program in both C and Java. And I am familiar with the `pthread` API in C and the `ForkJoinTask` in Java, which is helpful in multithreaded programming. What's more, I understand the multithreaded programming better after some tries.

By the way, you can [find all the source codes in the "src" folder](#). You can also refer to [my github](#) to see my codes of this project, and they are in the `Project3` folder.