Project3: ***Multithreaded Sorting Application*** & ***Fork-Join Sorting Application***

王凯俊 520021910282

Project3-1: Multithreaded Sorting Application

1. 1 Introduction

In this program, we will conduct a program which can using multithreaded methods to sort the arrays and merge them into one array. The program is structured as the figure below:

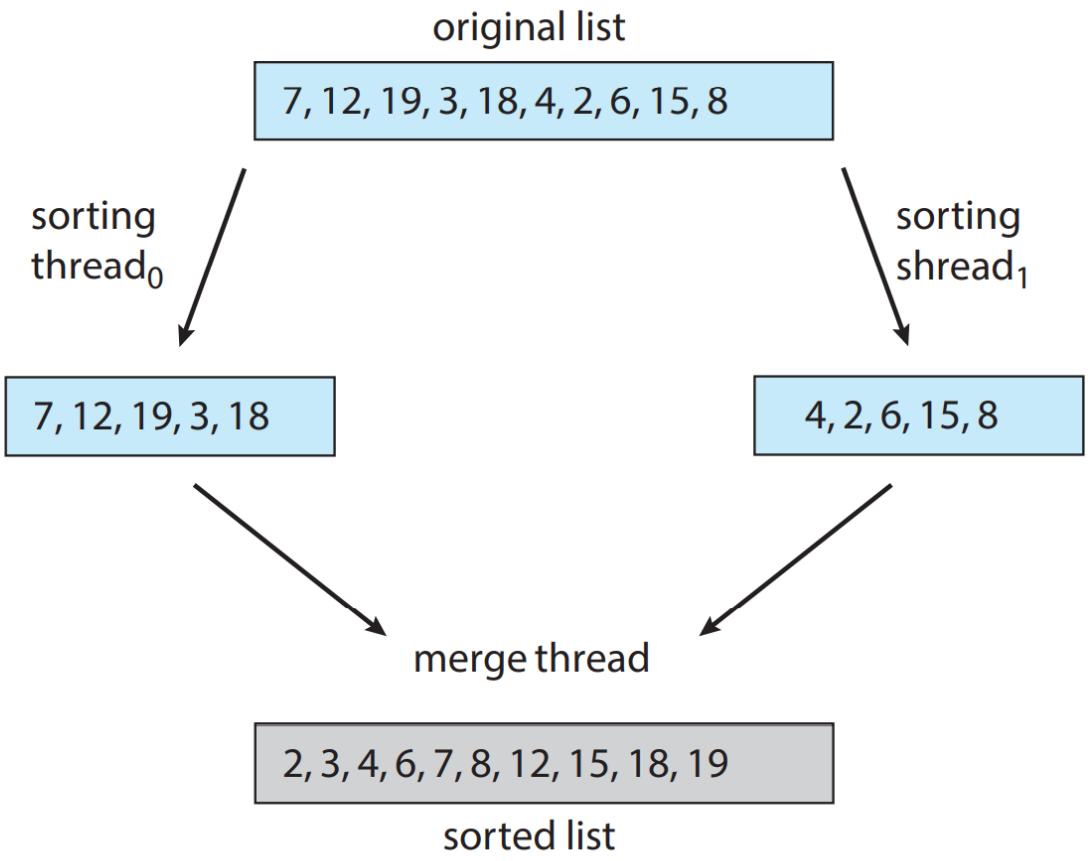


Figure 4.27 Multithreaded sorting.

2. 2 Implementation Details

2.1. 2-1 The code framework

To finish this project, there are 333 parts to be finished:

1. init the array
2. init the threads and sort the arrays
3. merge and sort the arrays

2.2. 2-2 Init the array

We should first set the array and waiting the input. The detailed code is as below:

```
#define MAX_NUM 100
int array[MAX_NUM];
size_t array_size = 0;
// set the array
printf("Please enter the number of elements:");
    scanf("%ld", &array_size);
    for(size_t i = 0; i != array_size; ++i) {
        scanf("%d", &array[i]);
```

```
// input data and store in the array
```

2.3. 2-3 Init the threads and sort the arrays

To sort two arrays, we should set 2 threads. The init code is as below:

```
pthread_t tid[2];
pthread_attr_t attr;
```

Then we want to achieve our sorting goals, we use the "runner" as the function to sort the arrays.

The runners function is as below:

```
void *runner(void *param) {
    size_t low, high;
    low = ((parameters *)param)->low;
    high = ((parameters *)param)->high;
    qsort(array + low, high - low, sizeof(int), cmp);
    // cmp means the compare function
    pthread_exit(0);
}
int cmp(const void  *a, const void *b) {
    return *((int *)a) - *((int *)b);
}
```

We want to call the *runner* function in the main fuction with 2 threads, the detailed code is as below:

```
    pthread_attr_init(&attr);
    /* create two threads to sort the two halves of the array */
    for(size_t i = 0; i != 2; ++i) {
        pthread_create(&tid[i], &attr, runner, &data[i]);
    }
    /* now wait for the thread to exit */
    for(size_t i = 0; i != 2; ++i) {
        pthread_join(tid[i], NULL);
    }
```

2.4. 2-4 Merge and sort the arrays

After sort two arrays respectively, we want to merge and sort these two arrays into one array. The detailed code is as below:

```
void merge_array(int *result) {
    size_t low1 = 0, high1 = array_size / 2;
    size_t low2 = array_size / 2, high2 = array_size;
    size_t i = 0;
    while(low1 < high1 && low2 < high2) {
        if(array[low1] < array[low2]) {
            result[i++] = array[low1++];
        } else {
            result[i++] = array[low2++];
        }
    }
    if(low2 < high2) {
        low1 = low2, high1 = high2;
    }
    while(low1 < high1) {
        result[i++] = array[low1++];
    }
}
```

2.5. 2-5 The main fuction

After the steps above, the basic framework has been made.And we want to connect these modes using the main function. The code in main function is as below:

```
int main() {
    pthread_t tid[2];
    pthread_attr_t attr; //set the threads
    init_array();
    printf("Original array:\n");
    print_array(array, array_size);
    parameters data[2];
    data[0].low = 0;
    data[0].high = array_size / 2;
    data[1].low = array_size / 2;
    data[1].high = array_size;
    pthread_attr_init(&attr);
    // create two threads to sort the two halves of the array
    for(size_t i = 0; i != 2; ++i) {
        pthread_create(&tid[i], &attr, runner, &data[i]);
    }
    // wait for the thread to exit */
    for(size_t i = 0; i != 2; ++i) {
        pthread_join(tid[i], NULL);
    }
    printf("Thread 0:\n");
    print_array(array, array_size / 2);
    printf("Thread 1:\n");
    print_array(array + array_size / 2, array_size - array_size / 2);
    int *sorted_array = malloc(sizeof(int) * array_size);
    merge_array(sorted_array);
```

```
    printf("After merging:\n");
    print_array(sorted_array, array_size);
    return 0;
}
```

3. 3 Program Results

Complie the program:

```
gcc multithread_sorting.c -o multithread_sorting -l pthread
```

Then type **./multithread_sorting** to execute it.

And the result is as below:

Input 10 elements:

```
jackiewang@ubuntu:~/Desktop/Project3$ gcc multithread_sorting.c -o multithread_sorting
-l pthread
jackiewang@ubuntu:~/Desktop/Project3$ ./multithread_sorting
Please enter the number of elements:10
1 3 2 5 7 6 12 45 62 3
Original array:
1 3 2 5 7 6 12 45 62 3
Thread 0:
1 2 3 5 7
Thread 1:
3 6 12 45 62
After merging:
1 2 3 3 5 6 7 12 45 62
jackiewang@ubuntu:~/Desktop/Project3$
```

Input 20 elements:

```
jackiewang@ubuntu:~/Desktop/Project3$ ./multithread_sorting
Please enter the number of elements:20
1 5 34 32 5 34 23 87 35 2 453 435 43 78 34 23 54 23 78 0
Original array:
1 5 34 32 5 34 23 87 35 2 453 435 43 78 34 23 54 23 78 0
Thread 0:
1 2 5 5 23 32 34 34 35 87
Thread 1:
0 23 23 34 43 54 78 78 435 453
After merging:
0 1 2 5 5 23 23 23 32 34 34 34 35 43 54 78 78 87 435 453
jackiewang@ubuntu:~/Desktop/Project3$
```

Project3-2: Fork-Join Sorting Application

4. 1 Introduction

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:

1. Quicksort
2. Mergesort

We want to divide the data and 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 algorithm such as the Selection or Insertion sort.

There has been an example about sum function, we need to modify it to achieve our sorting goals.

5. 2 Implementation Details

5.1. 2-1 The code framework

We first see the **sum** function. then we realise that we need to conduct the two sorting functions in the large part.And then we should use the selection sort or something to achieve the small part conquer.And apparently, we need use the Java api to achieve the thread pool.

So there are total 3 steps to achieve:

1. finish the 2 sorting function
2. achieve the small part conquer
3. add the threads

5.2. 2-2 Finish the 2 sorting function

We should achieve the quicksort and mergesort using the Java grammer.

Luckily, there are not much difference between C and Java. So we can achieve it using a C-likely way. The detaeled code is as below:

The quicksort:

```
    int i=low;
    int j=high;
    int pivot = array[low];
    if(low>=high)
        return;
    while(low<high)
    {
        while(low<high && pivot<=array[high])
            high--;
        if(pivot > array[high])
        {
            int tmp;
            tmp = array[low];
            array[low] = array[high];
            array[high] =tmp;
```

```
            ++low;
        }
        while (low < high && pivot >= array[low])
        {
        ++low;
        }
        if (pivot < array[low])
        {
            int tmp;
            tmp = array[low];
            array[low] = array[high];
            array[high] =tmp;
            --high;
        }
    }
    quick_sort_pivot(i, low-1);
    quick_sort_pivot(low+1, j);
```

The merge sort:

```
        int current1=left;
        int current2=mid+1;
        int copy_id =left;
        int[] array_copy = new int[SIZE];
        while(current1<=mid && current2<=right)
        {
            if(array[current1]<=array[current2])
                {
                    array_copy[copy_id]=array[current1];
                    current1++;
                    copy_id++;
                }
            else
            {
                array_copy[copy_id]=array[current2];
                current2++;
                copy_id++;
            }
        }
        if(copy_id==right)
        {
            for(int i=left;i<=right;i++)
                array[i]=array_copy[i];
        }
        else if(current1<mid)
        {
            for(int i=current1;i<=mid;i++)
                {
                    array_copy[copy_id]=array[i];
                    copy_id++;
                }
            for(int i=left; i<=right;i++)
```

```
                array[i]=array_copy[i];
        }
        else if(current2<right)
        {
                for(int i=current2;i<=right;i++)
                {
                    array_copy[copy_id]=array[i];
                    copy_id++;
                }
                for(int i=left; i<=right;i++)
                    array[i]=array_copy[i];
        }

    }
```

5.3. 2-3 Achieve the small part conquer

We want to achieve the conquer part using the selection sort.

The detailed code is as below:

```
private void selection_sort(int low, int high)
    {
        for(int i=low;i<high;i++)
        {
            int min = array[i];
            for(int j=i+1;j<=high;j++)
                {
                    if(array[j]<min)
                    {
                        min = array[j];
                        array[j] = array[i];
                        array[i] = min;
                    }
                }
        }
    }
```

5.4. 2-4 Add the threads

We use fork-join parallelism to get the array.

```
        SortTask task = new SortTask(0, SIZE-1, array);
        pool.invoke(task);
```

5.5. 2-5 The main function

We want to input some random numbers to test our program. So we use the **rand.nextInt()** to achieve it.

Then we want to do our function and output the results on the screen.

The detailed code is as below:

```
    public static void main(String[] args)
    {
        ForkJoinPool pool = new ForkJoinPool();
        int[] array = new int[SIZE];

        // create SIZE random integers between 0 and 100
        java.util.Random rand = new java.util.Random(100);

        System.out.println("Init array is: ");

        for (int i = 0; i < SIZE; i++) {
            array[i] = rand.nextInt(10);

            System.out.print(array[i]);
            System.out.print(' ');
        }

        SortTask task = new SortTask(0, SIZE-1, array);

        pool.invoke(task);
        System.out.println(' ');

        System.out.println("Sorted array is: ");
        for (int i = 0; i < SIZE; i++) {
            System.out.print(array[i]);
            System.out.print(' ');
        }
    }
```

6. 3 Program Results

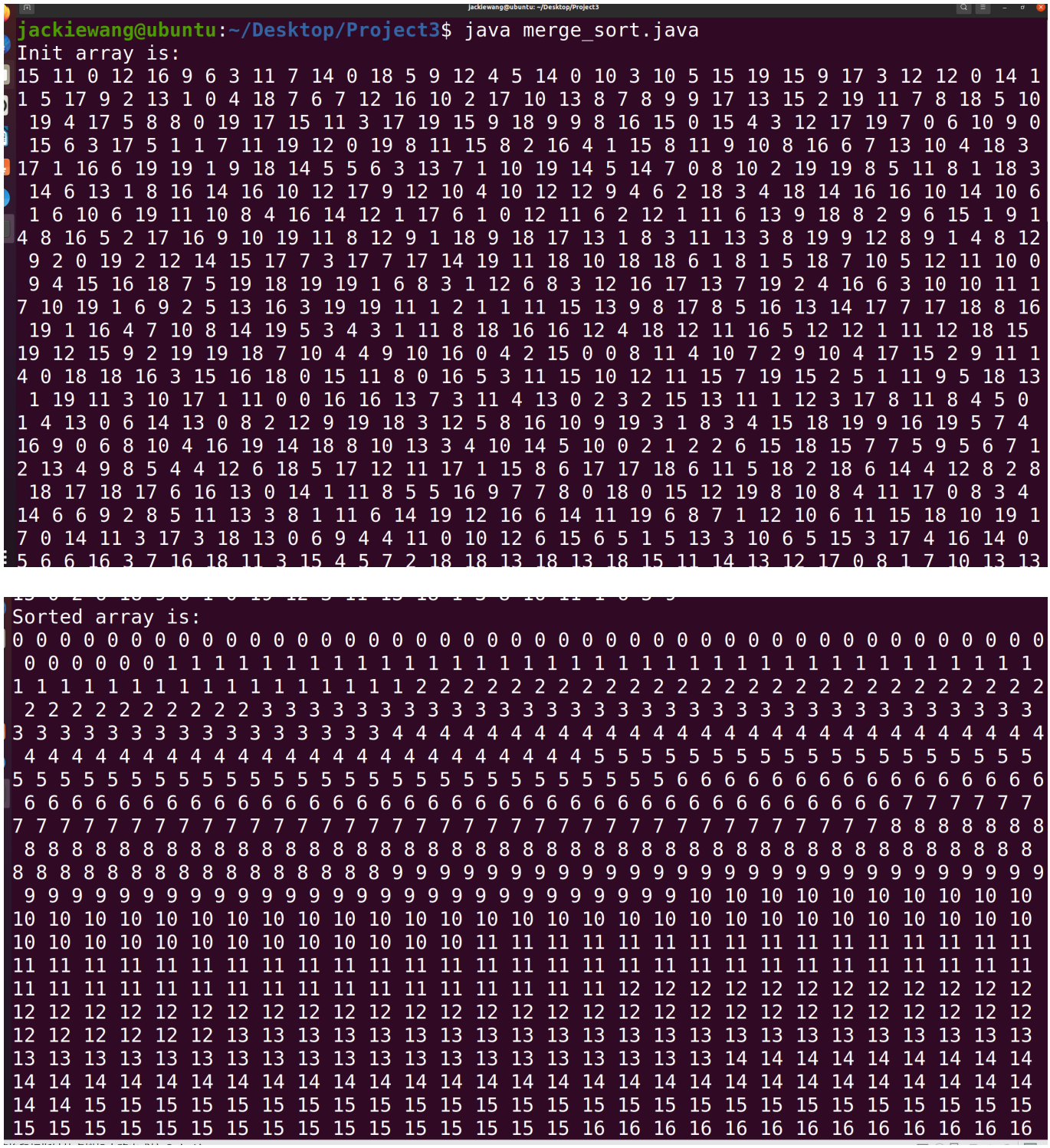
The result is as below:

The **quick_sort** with random **from 0 to 100**:

```
jackiewang@ubuntu:~/Desktop/Project3$ java quick_sort.java
Init array is:
95 89 98 71 36 75 42 7 79 80 49 78 96 54 5 91 99 59 51 11 87 2 39 11 25 1 63 37 53 23 8
4 2 22 56 10 91 29 67 66 79 91 44 86 14 35 75 72 32 16 77 12 85 73 87 60 64 16 48 18 86
 41 16 93 47 42 91 37 16 44 63 1 23 73 86 60 32 82 16 68 33 41 2 2 42 20 5 62 71 75 42
20 95 63 63 77 57 73 54 52 6 99 37 32 3 59 54 95 72 97 50 21 10 2 54 95 18 67 84 63 84
21 57 16 43 43 43 92 10 78 47 11 64 80 78 84 49 48 56 30 30 31 47 91 84 0 0 80 0 91 45
94 62 35 59 19 84 73 98 22 38 5 35 37 8 20 39 19 68 1 19 97 6 61 92 70 26 94 72 4 94 83
 87 87 53 82 12 40 44 97 60 24 83 39 65 46 17 86 39 79 89 97 12 65 20 19 66 27 4 80 96
64 26 69 58 34 78 79 5 0 57 74 47 72 28 39 71 31 74 20 22 46 50 52 2 74 24 61 36 20 35
51 54 63 94 67 66 87 32 54 58 91 53 57 36 51 66 45 6 36 68 88 68 42 94 98 63 31 66 3 90
 2 11 45 75 75 0 90 31 62 60 1 11 79 81 66 48 89 63 77 22 72 20 39 53 74 30 64 48 17 85
 28 4 9 98 56 86 24 17 58 81 60 29 98 43 47 97 9 70 94 91 96 78 59 92 65 66 49 13 28 12
 87 87 1 43 33 75 4 4 79 17 75 25 51 7 52 71 25 9 9 70 63 9 61 99 87 25 65 73 33 88 17
9 33 38 42 97 59 59 79 37 43 44 28 33 24 43 55 90 33 17 27 76 30 25 92 44 41 95 33 40 9
7 2 76 92 57 19 19 44 21 48 43 24 25 34 62 43 43 56 76 21 40 93 36 80 56 91 90 54 20 2
71 58 88 31 87 12 72 39 95 59 0 49 3 34 57 11 13 14 40 70 62 64 34 60 93 46 70 89 5 41
57 21 44 59 96 88 65 94 53 36 7 46 45 95 0 7 26 23 69 70 40 18 77 10 74 20 73 27 50 92
7 41 35 58 63 3 52 60 86 78 93 67 27 1 90 93 5 50 76 33 38 28 73 76 45 60 40 7 34 2 98
78 72 53 56 92 94 54 12 16 55 24 11 76 15 36 33 76 2 80 61 75 47 74 48 83 27 46 77 67 4
1 90 24 10 47 6 0 82 3 61 63 75 7 99 69 87 14 87 76 91 7 12 50 15 9 22 34 23 70 0 77 46
 58 29 6 61 95 15 96 69 39 99 95 41 86 11 39 95 22 63 25 58 33 60 33 43 4 18 53 20 14 9
7 43 9 49 35 62 78 31 29 75 30 82 47 79 11 28 38 34 22 21 79 34 98 38 66 78 53 72 80 54
 16 0 41 80 68 22 14 62 25 8 12 90 35 87 39 79 36 2 94 82 19 60 11 25 86 37 46 8 70 96
```

```
Sorted array is:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 4
 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 9
9 9 9 9 9 9 9 9 9 9 10 10 10 10 10 10 10 10 11 11 11 11 11 11 11 11 11 11 11 11 11 1
2 12 12 12 12 12 12 12 12 12 12 12 12 13 13 13 13 13 13 13 14 14 14 14 14 14 14 15 15 1
5 15 15 15 16 16 16 16 16 16 16 16 16 16 17 17 17 17 17 17 17 17 17 18 18 18 18 18 18 1
8 18 18 18 19 19 19 19 19 19 19 19 19 20 20 20 20 20 20 20 20 20 20 20 20 21 21 21 21 2
1 21 21 21 21 21 21 21 22 22 22 22 22 22 22 22 22 22 22 23 23 23 23 23 24 24 24 24 24 2
4 24 24 24 24 24 24 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 26 26 26 26 26 2
6 27 27 27 27 27 27 27 27 27 28 28 28 28 28 28 28 28 29 29 29 29 29 29 29 30 30 30 30 3
0 31 31 31 31 31 31 31 31 31 31 31 31 31 32 32 32 32 32 32 32 32 33 33 33 33 33 33 33 3
3 33 33 33 33 33 33 34 34 34 34 34 34 34 34 34 34 34 34 34 34 35 35 35 35 35 35 35 35 3
5 36 36 36 36 36 36 36 36 36 36 37 37 37 37 37 37 37 37 37 37 38 38 38 38 38 38 38 38 3
8 39 39 39 39 39 39 39 39 39 39 39 39 39 40 40 40 40 40 40 40 40 40 40 40 40 41 41 4
1 41 41 41 41 41 41 41 41 41 41 41 41 42 42 42 42 42 42 42 43 43 43 43 43 43 43 43 4
3 43 43 43 43 43 44 44 44 44 44 44 44 44 44 44 44 45 45 45 45 45 45 45 45 46 46 46 46 46 4
6 46 46 46 46 46 47 47 47 47 47 47 47 47 47 47 47 47 47 48 48 48 48 48 48 48 48 48 48 4
9 49 49 49 49 49 49 49 49 49 50 50 50 50 50 50 50 50 50 51 51 51 51 51 51 51 52 52 52 5
2 52 52 53 53 53 53 53 53 53 53 53 53 53 54 54 54 54 54 54 54 54 54 55 55 55 55 55 55 5
5 56 56 56 56 56 56 56 56 56 56 56 56 56 57 57 57 57 57 57 57 57 57 57 57 58 58 58 58 58 5
```

The **merge_sort** with random **from 0 to 20**:



Conclusion and Thoughts

In project 3-1, we learned about using C language to implement a multithread program, based on the **pthread** library. In project 3-2, we utilize **fork-join** in Java, similar to the multithread in C.

I learned about the Java grammer and api.

I also know some about the threads' knowledge.

I review the completion of quick sort and merge sort.

All in all, it's a good experience which enhance my coding ability.