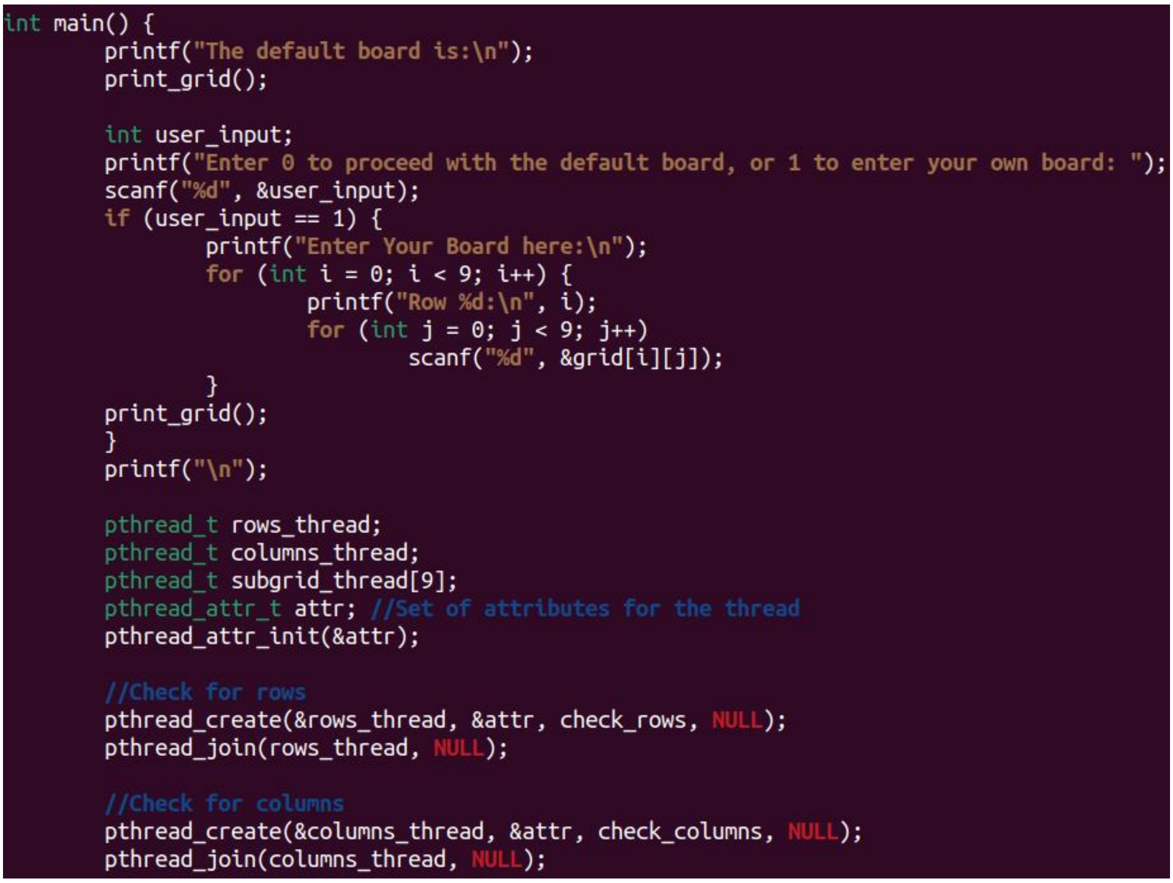
Report of Projects

Description:

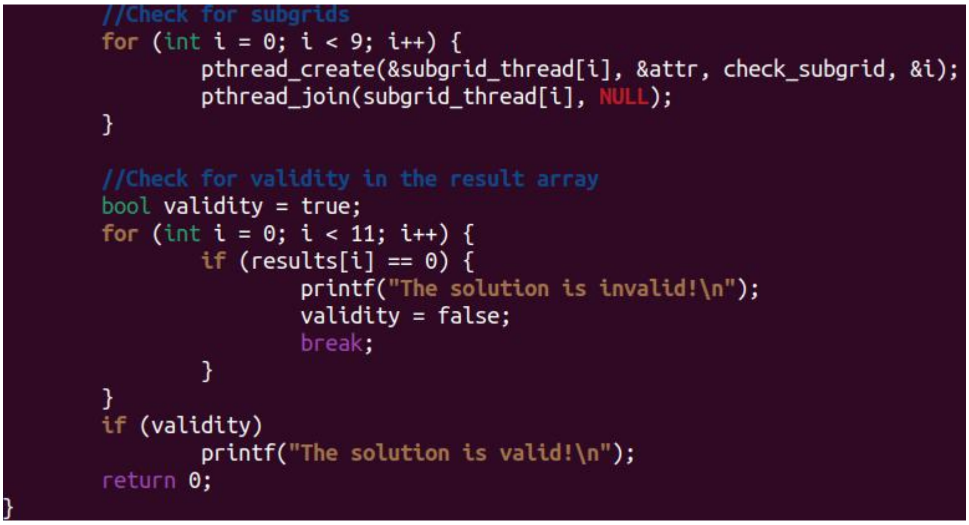
There are 2 projects: Sudoku Solution Validator, and Fork-Join Sorting.
And I will show you the codes, results, and explanation of each project.

Project 1: Sudoku Solution Validator.

Main Function:



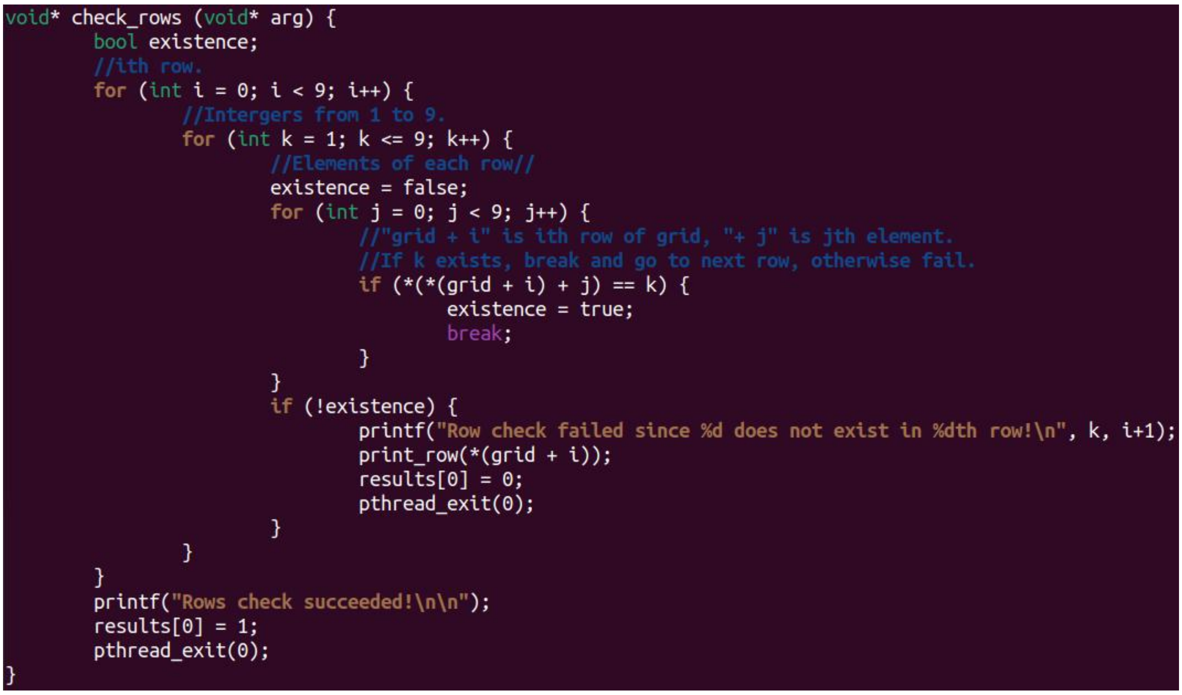
```
int main() {
        printf("The default board is:\n");
        print_grid();

        int user_input;
        printf("Enter 0 to proceed with the default board, or 1 to enter your own board: ");
        scanf("%d", &user_input);
        if (user_input == 1) {
                printf("Enter Your Board here:\n");
                for (int i = 0; i < 9; i++) {
                        printf("Row %d:\n", i);
                        for (int j = 0; j < 9; j++)
                                scanf("%d", &grid[i][j]);
                }
        print_grid();
        }
        printf("\n");

        pthread_t rows_thread;
        pthread_t columns_thread;
        pthread_t subgrid_thread[9];
        pthread_attr_t attr; //Set of attributes for the thread
        pthread_attr_init(&attr);

        //Check for rows
        pthread_create(&rows_thread, &attr, check_rows, NULL);
        pthread_join(rows_thread, NULL);

        //Check for columns
        pthread_create(&columns_thread, &attr, check_columns, NULL);
        pthread_join(columns_thread, NULL);
```



```
        //Check for subgrids
        for (int i = 0; i < 9; i++) {
                pthread_create(&subgrid_thread[i], &attr, check_subgrid, &i);
                pthread_join(subgrid_thread[i], NULL);
        }

        //Check for validity in the result array
        bool validity = true;
        for (int i = 0; i < 11; i++) {
                if (results[i] == 0) {
                        printf("The solution is invalid!\n");
                        validity = false;
                        break;
                }
        }
        if (validity)
                printf("The solution is valid!\n");
        return 0;
}
```

1) Firstly, I will show you the default board of a sudoku solution, and let you decide whether to choose it for proceeding or use your board.
2) Then I created 9 threads (One for checking all rows, one for checking all columns, and 9 for checking each of the subgrids), and used them to check the validity separately by thread, and join them after finished.
3) Then I stored the result of each thread in an array results[11], and check the validity of the solution by the results[].

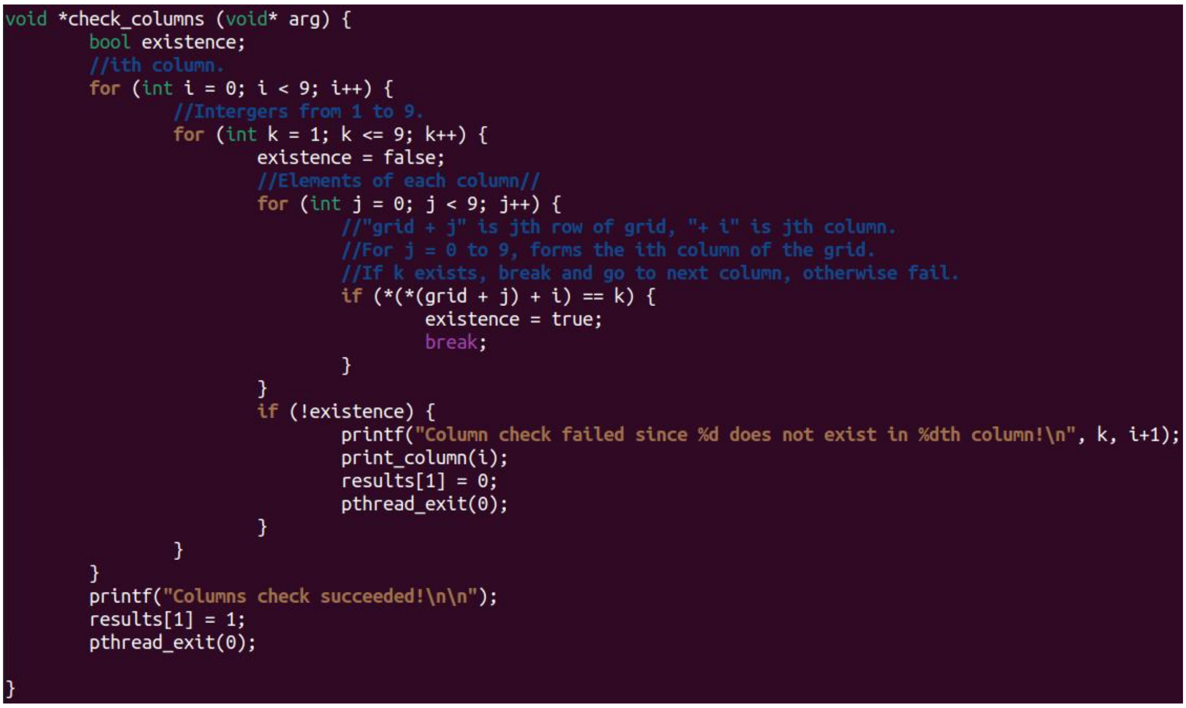
Check Rows:



```
void* check_rows (void* arg) {
        bool existence;
        //ith row.
        for (int i = 0; i < 9; i++) {
                //Intergers from 1 to 9.
                for (int k = 1; k <= 9; k++) {
                        //Elements of each row//
                        existence = false;
                        for (int j = 0; j < 9; j++) {
                                //"grid + i" is ith row of grid, "+ j" is jth element.
                                //If k exists, break and go to next row, otherwise fail.
                                if (*(*(grid + i) + j) == k) {
                                        existence = true;
                                        break;
                                }
                        }
                        if (!existence) {
                                printf("Row check failed since %d does not exist in %dth row!\n", k, i+1);
                                print_row(*(grid + i));
                                results[0] = 0;
                                pthread_exit(0);
                        }
                }
        }
        printf("Rows check succeeded!\n\n");
        results[0] = 1;
        pthread_exit(0);
}
```

1) In this function, I checked the validity of all rows by for(int i = 0; i < 9; i++).
2) For each row, I checked whether all the integers from 1 to 9 exists in the row.
3) In order to check, I used *(*(grid + i) + j) == k, to loop for all elements of the row, and once there is a row that an integer k can’t be found from the row, I will directly set the result of checking all rows to false.
4) Only if there is no false until the loops finished, the result could be true.

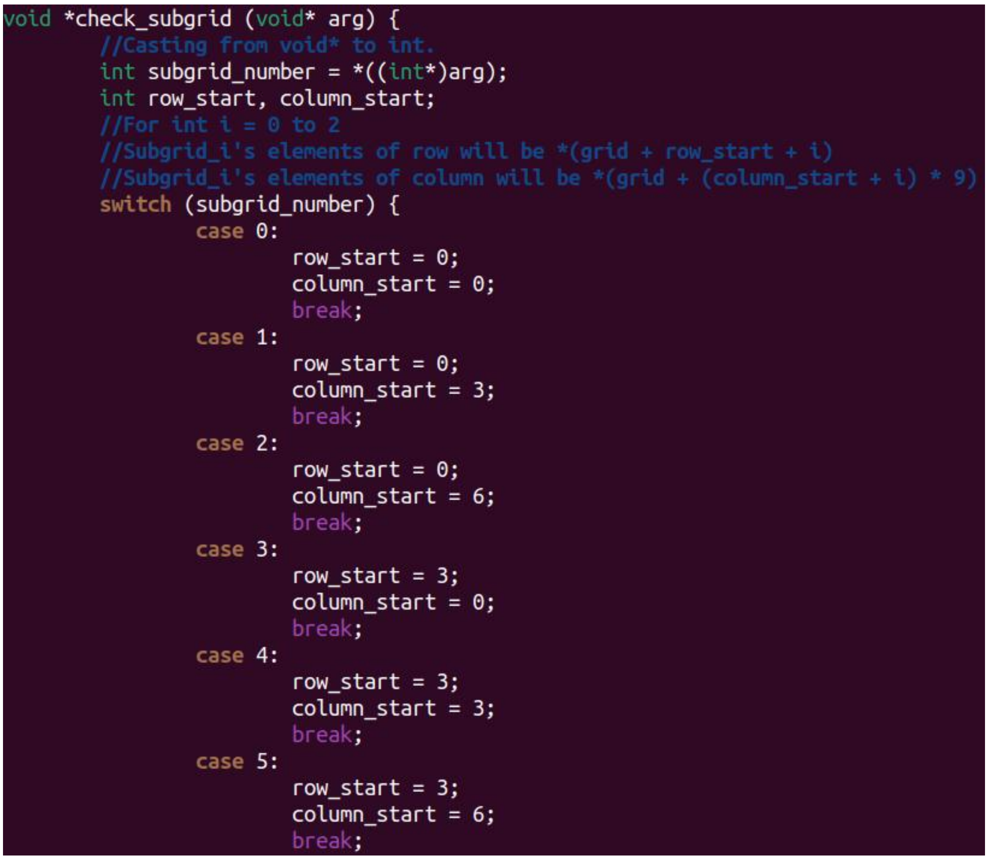
Check Columns:



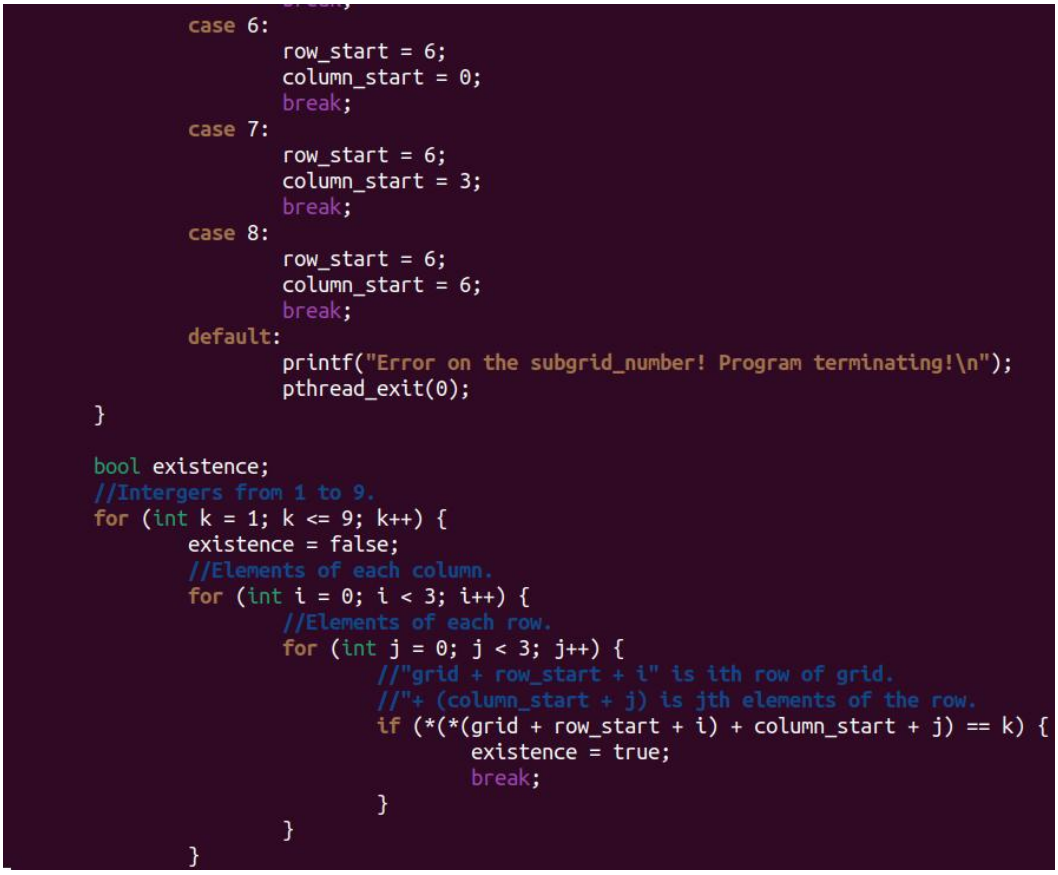
```
void *check_columns (void* arg) {
        bool existence;
        //ith column.
        for (int i = 0; i < 9; i++) {
                //Intergers from 1 to 9.
                for (int k = 1; k <= 9; k++) {
                        existence = false;
                        //Elements of each column//
                        for (int j = 0; j < 9; j++) {
                                //"grid + j" is jth row of grid, "+ i" is jth column.
                                //For j = 0 to 9, forms the ith column of the grid.
                                //If k exists, break and go to next column, otherwise fail.
                                if (*(*(grid + j) + i) == k) {
                                        existence = true;
                                        break;
                                }
                        }
                        if (!existence) {
                                printf("Column check failed since %d does not exist in %dth column!\n", k, i+1);
                                print_column(i);
                                results[1] = 0;
                                pthread_exit(0);
                        }
                }
        }
        printf("Columns check succeeded!\n\n");
        results[1] = 1;
        pthread_exit(0);

}
```

1) Almost the same idea with check_rows, just switch the order to change the rows to columns.

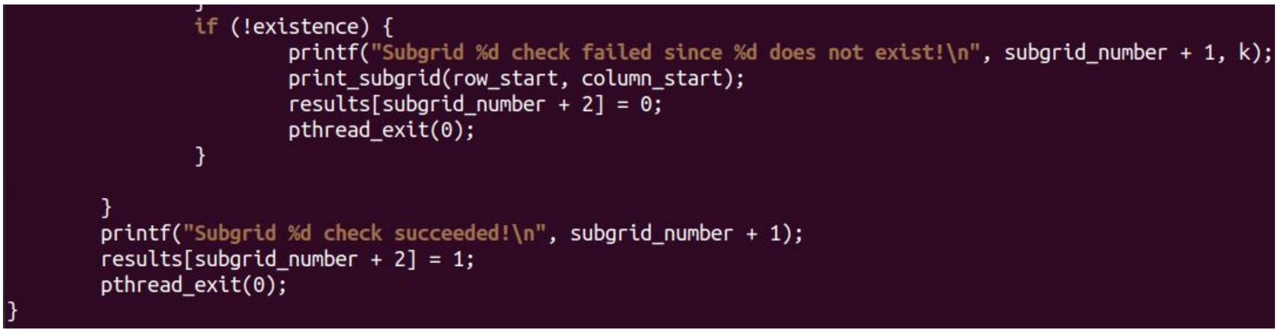
Check Subgrid:



```
void *check_subgrid (void* arg) {
        //Casting from void* to int.
        int subgrid_number = *((int*)arg);
        int row_start, column_start;
        //For int i = 0 to 2
        //Subgrid_i's elements of row will be *(grid + row_start + i)
        //Subgrid_i's elements of column will be *(grid + (column_start + i) * 9)
        switch (subgrid_number) {
                case 0:
                        row_start = 0;
                        column_start = 0;
                        break;
                case 1:
                        row_start = 0;
                        column_start = 3;
                        break;
                case 2:
                        row_start = 0;
                        column_start = 6;
                        break;
                case 3:
                        row_start = 3;
                        column_start = 0;
                        break;
                case 4:
                        row_start = 3;
                        column_start = 3;
                        break;
                case 5:
                        row_start = 3;
                        column_start = 6;
                        break;
```



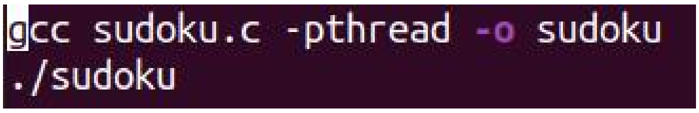
```
                case 6:
                        row_start = 6;
                        column_start = 0;
                        break;
                case 7:
                        row_start = 6;
                        column_start = 3;
                        break;
                case 8:
                        row_start = 6;
                        column_start = 6;
                        break;
                default:
                        printf("Error on the subgrid_number! Program terminating!\n");
                        pthread_exit(0);
        }

        bool existence;
        //Intergers from 1 to 9.
        for (int k = 1; k <= 9; k++) {
                existence = false;
                //Elements of each column.
                for (int i = 0; i < 3; i++) {
                        //Elements of each row.
                        for (int j = 0; j < 3; j++) {
                                //"grid + row_start + i" is ith row of grid.
                                //"+ (column_start + j) is jth elements of the row.
                                if (*(*(grid + row_start + i) + column_start + j) == k) {
                                        existence = true;
                                        break;
                                }
                        }
                }
```



```
                if (!existence) {
                        printf("Subgrid %d check failed since %d does not exist!\n", subgrid_number + 1, k);
                        print_subgrid(row_start, column_start);
                        results[subgrid_number + 2] = 0;
                        pthread_exit(0);
                }

        }
        printf("Subgrid %d check succeeded!\n", subgrid_number + 1);
        results[subgrid_number + 2] = 1;
        pthread_exit(0);
}
```

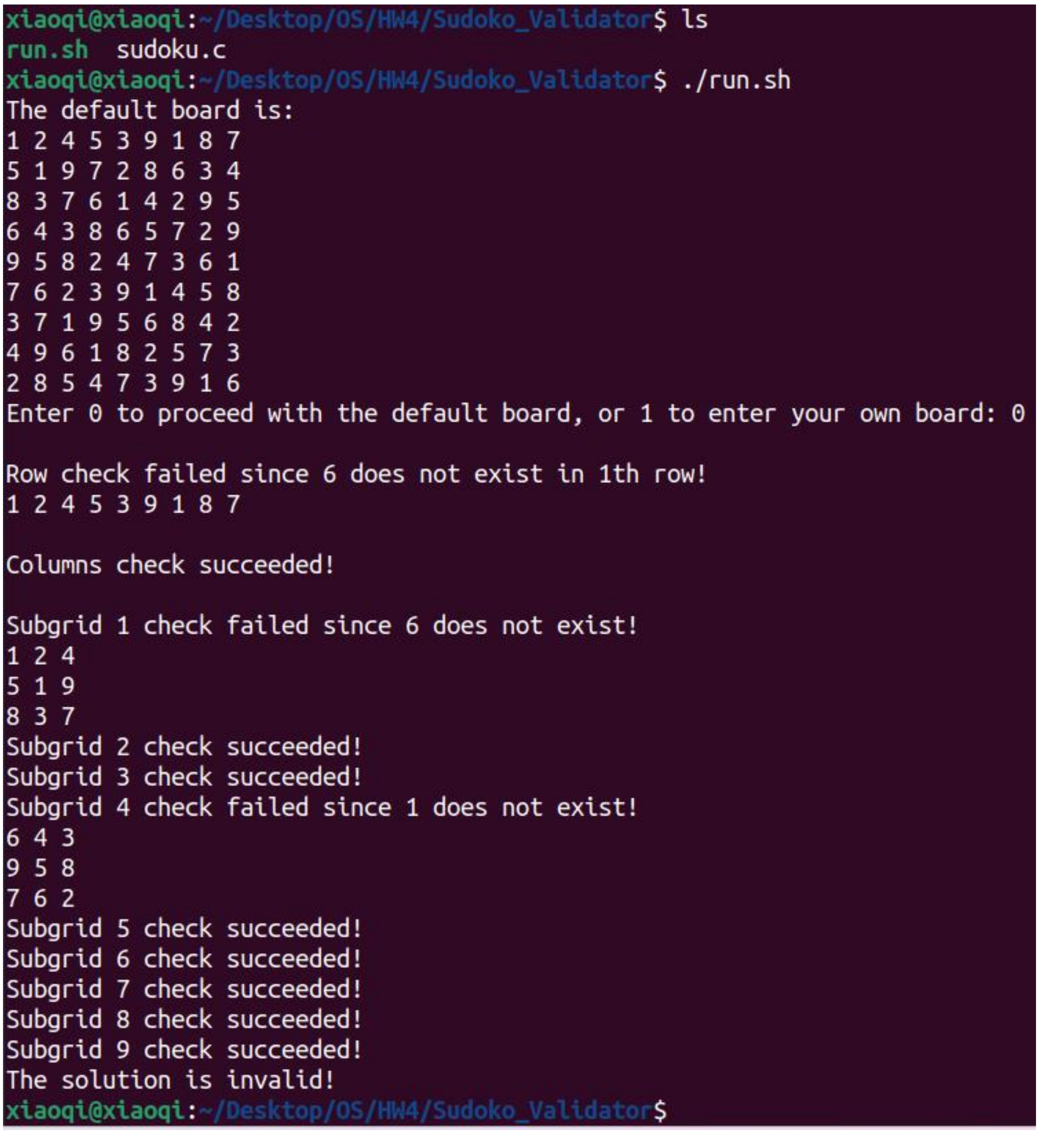
1) I didn't allocate the values of grid to 9 separate subgrids, instead, I used 2 variables "row_start", and "column_start", and I used "switch() case" to set the 2 variables to implement different subgrids.
2) For looping each subgrid, I used *(*(grid + row_start + i) + column_start + j), for i and j from 0 to 2.
3) Finally, after looping the subgrid and get a result, I put the result in the array results[subgrid_number + 2];

Compile (run.sh):



```
gcc sudoku.c -pthread -o sudoku
./sudoku
```

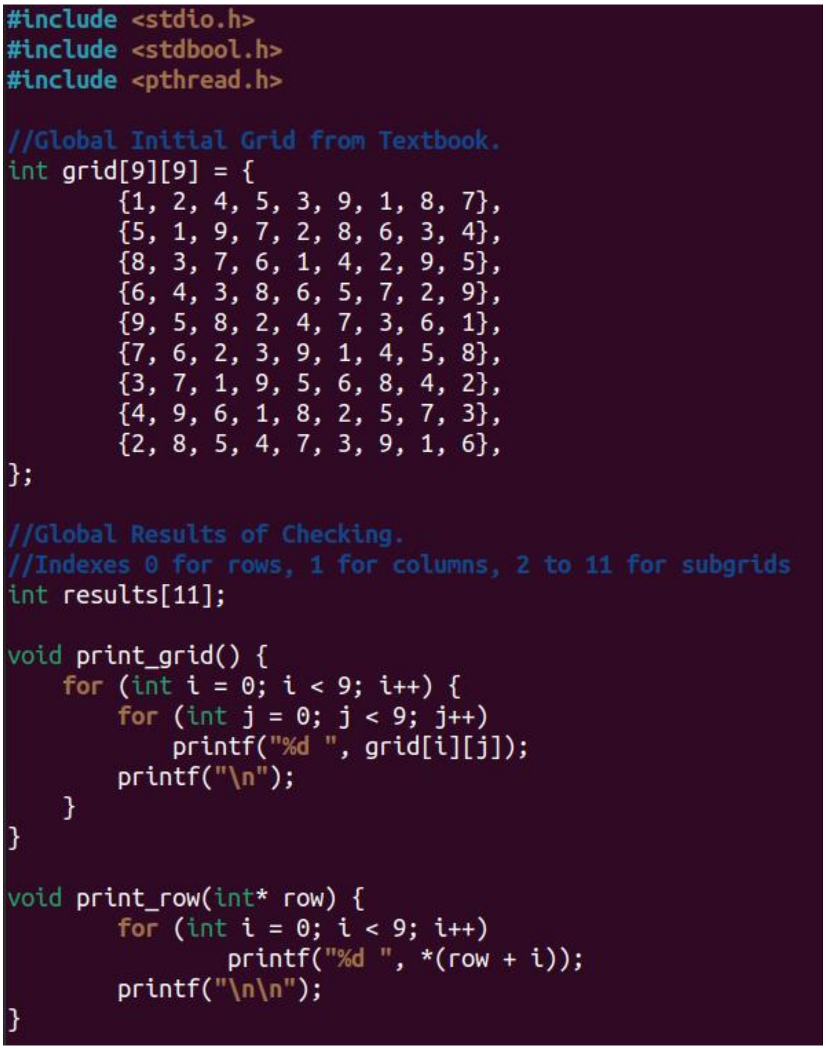
Results:



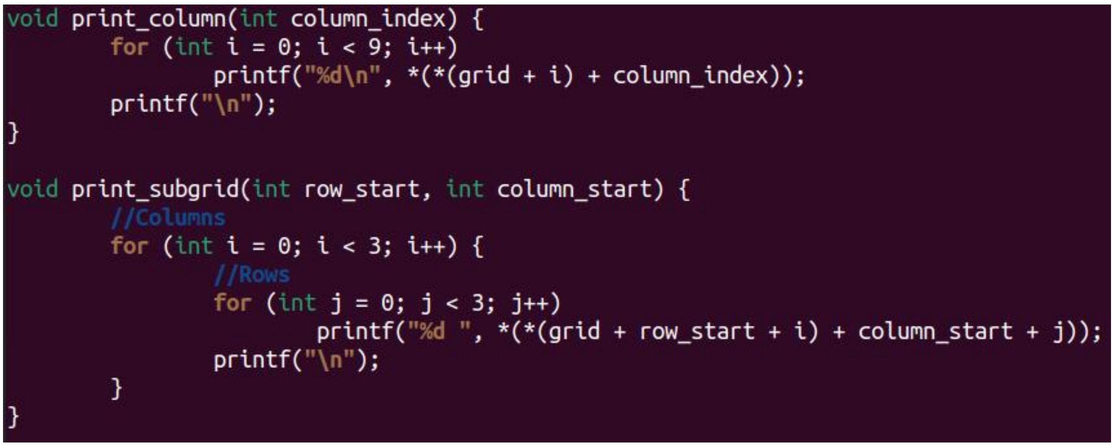
```
xiaoqi@xiaoqi:~/Desktop/OS/HW4/Sudoko_Validator$ ls
run.sh  sudoku.c
xiaoqi@xiaoqi:~/Desktop/OS/HW4/Sudoko_Validator$ ./run.sh
The default board is:
1 2 4 5 3 9 1 8 7
5 1 9 7 2 8 6 3 4
8 3 7 6 1 4 2 9 5
6 4 3 8 6 5 7 2 9
9 5 8 2 4 7 3 6 1
7 6 2 3 9 1 4 5 8
3 7 1 9 5 6 8 4 2
4 9 6 1 8 2 5 7 3
2 8 5 4 7 3 9 1 6
Enter 0 to proceed with the default board, or 1 to enter your own board: 0

Row check failed since 6 does not exist in 1th row!
1 2 4 5 3 9 1 8 7

Columns check succeeded!

Subgrid 1 check failed since 6 does not exist!
1 2 4
5 1 9
8 3 7
Subgrid 2 check succeeded!
Subgrid 3 check succeeded!
Subgrid 4 check failed since 1 does not exist!
6 4 3
9 5 8
7 6 2
Subgrid 5 check succeeded!
Subgrid 6 check succeeded!
Subgrid 7 check succeeded!
Subgrid 8 check succeeded!
Subgrid 9 check succeeded!
The solution is invalid!
xiaoqi@xiaoqi:~/Desktop/OS/HW4/Sudoko_Validator$
```

1) Here I changed the default board of solution a little bit in order to show my program could detect the errors. I swaped the grid[0][0] with grid[3][0] in order to make a situation that the check of columns will succeed while the check of rows and subgrid will fail.
2) As the result showed, the check of row failed because 6 doesn’t exist in the 1st row. Check of Subgrid 1 and 4 also failed because 6 and 1 doesn’t exist in the subgrid respectively.
3) I also added the function that it will print the corresponding data when it fails.

Other Codes:



```
#include <stdio.h>
#include <stdbool.h>
#include <pthread.h>

//Global Initial Grid from Textbook.
int grid[9][9] = {
        {1, 2, 4, 5, 3, 9, 1, 8, 7},
        {5, 1, 9, 7, 2, 8, 6, 3, 4},
        {8, 3, 7, 6, 1, 4, 2, 9, 5},
        {6, 4, 3, 8, 6, 5, 7, 2, 9},
        {9, 5, 8, 2, 4, 7, 3, 6, 1},
        {7, 6, 2, 3, 9, 1, 4, 5, 8},
        {3, 7, 1, 9, 5, 6, 8, 4, 2},
        {4, 9, 6, 1, 8, 2, 5, 7, 3},
        {2, 8, 5, 4, 7, 3, 9, 1, 6},
};

//Global Results of Checking.
//Indexes 0 for rows, 1 for columns, 2 to 11 for subgrids
int results[11];

void print_grid() {
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++)
            printf("%d ", grid[i][j]);
        printf("\n");
    }
}

void print_row(int* row) {
        for (int i = 0; i < 9; i++)
                printf("%d ", *(row + i));
        printf("\n\n");
}
```

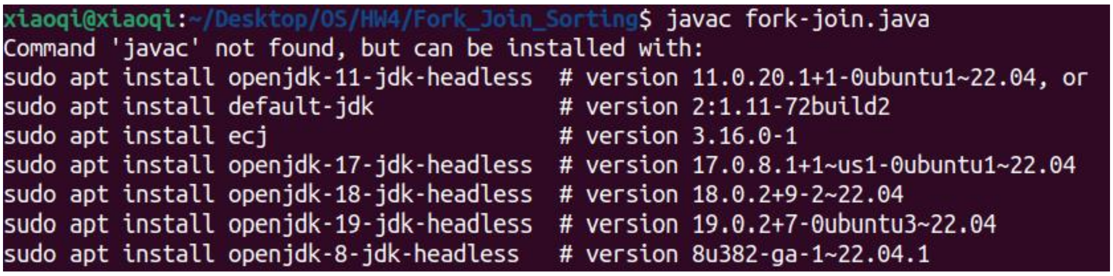


```
void print_column(int column_index) {
        for (int i = 0; i < 9; i++)
                printf("%d\n", *(*(grid + i) + column_index));
        printf("\n");
}

void print_subgrid(int row_start, int column_start) {
        //Columns
        for (int i = 0; i < 3; i++) {
                //Rows
                for (int j = 0; j < 3; j++)
                        printf("%d ", *(*(grid + row_start + i) + column_start + j));
                printf("\n");
        }
}
```

Project 2: Fork-Join Sorting Application.

Notice:

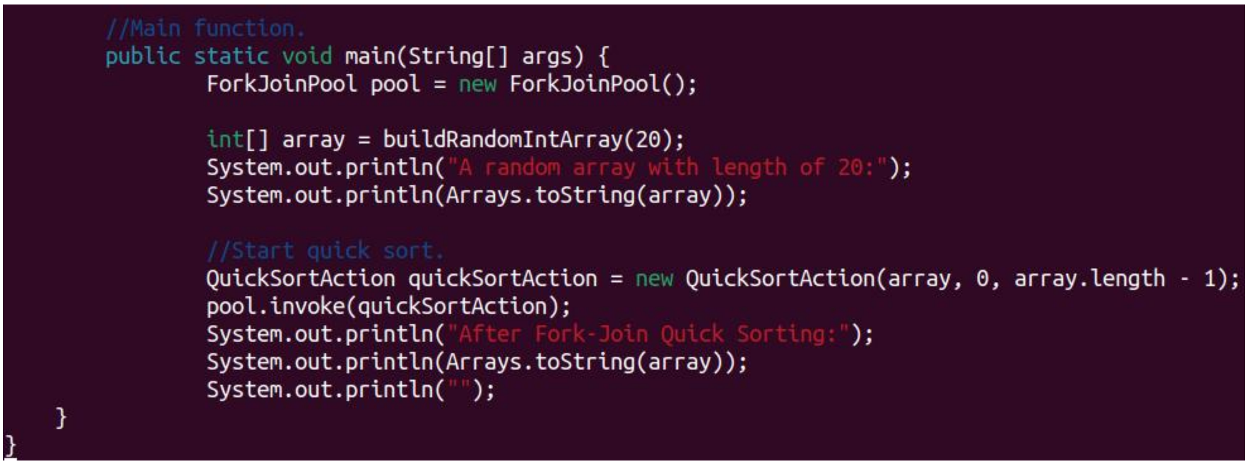
1) Since this project asks us to use Fork-Join technique to implement sorting application, and Fork-Join is a specific technique in Java, the program I wrote is based on Java.
2) If this is the first time that you use Java, please install the environment of Java first in order to compile it.



```
xiaoqi@xiaoqi:~/Desktop/OS/HW4/Fork_Join_Sorting$ javac fork-join.java
Command 'javac' not found, but can be installed with:
sudo apt install openjdk-11-jdk-headless  # version 11.0.20.1+1-0ubuntu1~22.04, or
sudo apt install default-jdk              # version 2:1.11-72build2
sudo apt install ecj                      # version 3.16.0-1
sudo apt install openjdk-17-jdk-headless  # version 17.0.8.1+1~us1-0ubuntu1~22.04
sudo apt install openjdk-18-jdk-headless  # version 18.0.2+9-2~22.04
sudo apt install openjdk-19-jdk-headless  # version 19.0.2+7-0ubuntu3~22.04
sudo apt install openjdk-8-jdk-headless   # version 8u382-ga-1~22.04.1
```

Application 1: Quick Sort:

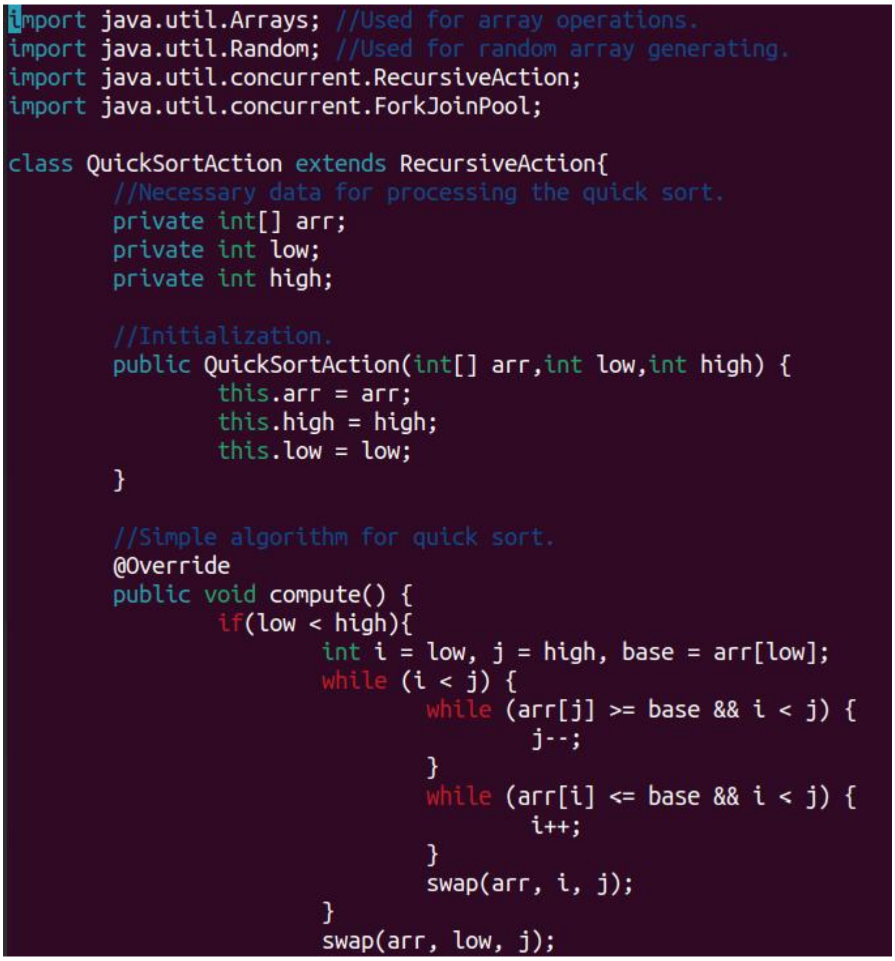
Main Function:



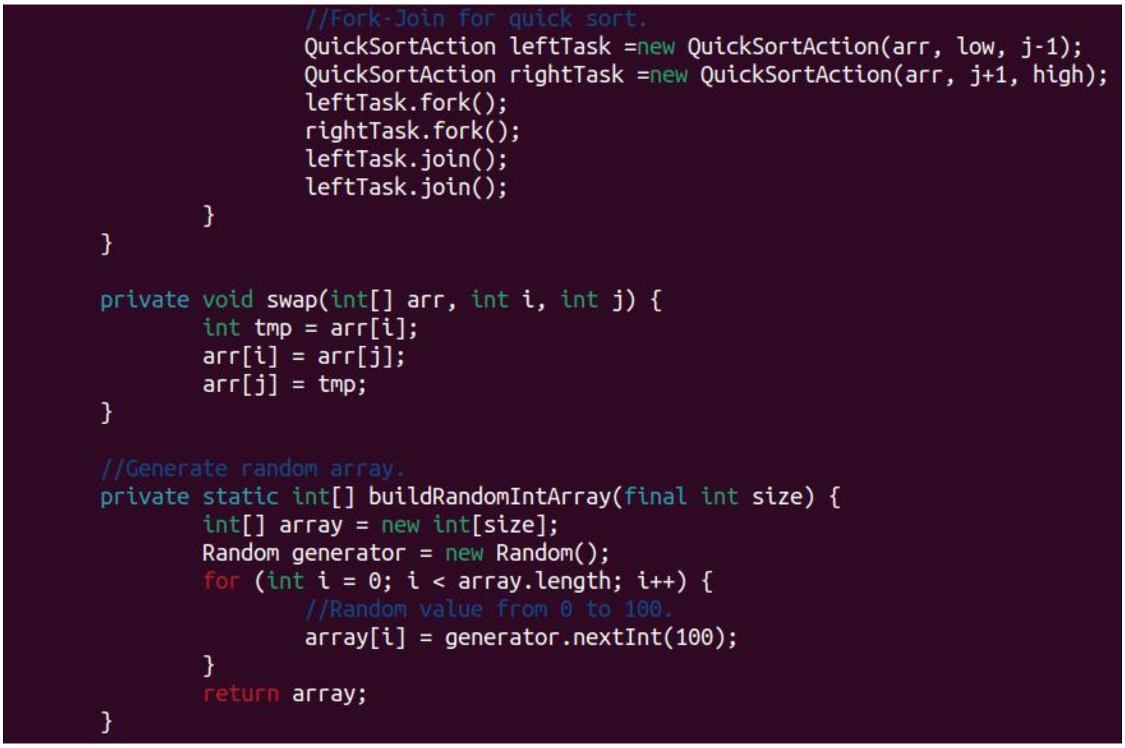
```
        //Main function.
        public static void main(String[] args) {
                ForkJoinPool pool = new ForkJoinPool();

                int[] array = buildRandomIntArray(20);
                System.out.println("A random array with length of 20:");
                System.out.println(Arrays.toString(array));

                //Start quick sort.
                QuickSortAction quickSortAction = new QuickSortAction(array, 0, array.length - 1);
                pool.invoke(quickSortAction);
                System.out.println("After Fork-Join Quick Sorting:");
                System.out.println(Arrays.toString(array));
                System.out.println("");
    }
}
```

1) The “ForkJoinPool” process and “pool.invoke()” is the necessary steps if you want to use Fork-Join in Java.
2) I built a random array with 20 elements first, then I used the Fork-Join Quick Sort to sort it. I also printed the array respectively.

Other Codes:



```
import java.util.Arrays; //Used for array operations.
import java.util.Random; //Used for random array generating.
import java.util.concurrent.RecursiveAction;
import java.util.concurrent.ForkJoinPool;

class QuickSortAction extends RecursiveAction{
        //Necessary data for processing the quick sort.
        private int[] arr;
        private int low;
        private int high;

        //Initialization.
        public QuickSortAction(int[] arr,int low,int high) {
                this.arr = arr;
                this.high = high;
                this.low = low;
        }

        //Simple algorithm for quick sort.
        @Override
        public void compute() {
                if(low < high){
                        int i = low, j = high, base = arr[low];
                        while (i < j) {
                                while (arr[j] >= base && i < j) {
                                        j--;
                                }
                                while (arr[i] <= base && i < j) {
                                        i++;
                                }
                                swap(arr, i, j);
                        }
                        swap(arr, low, j);
```

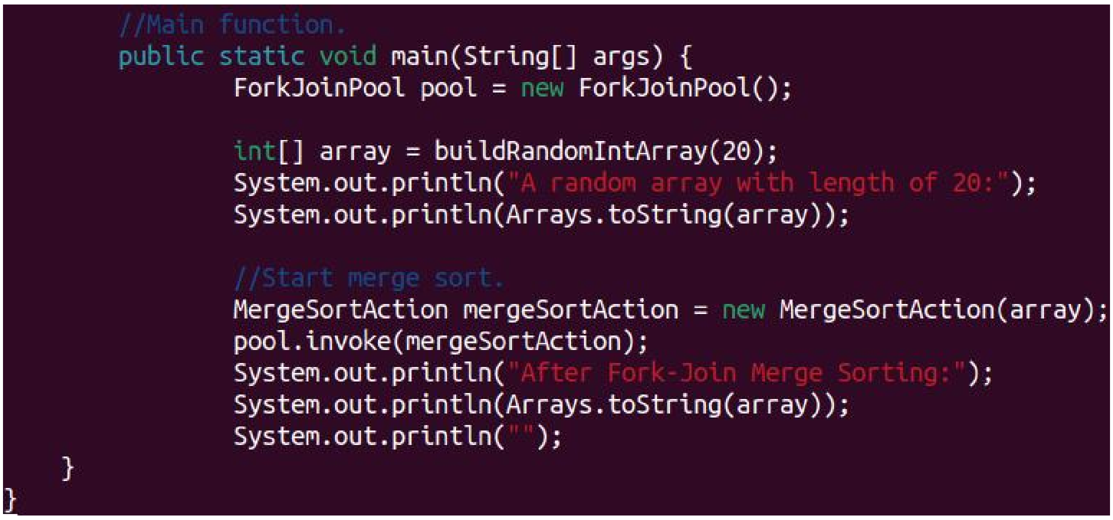


```
                        //Fork-Join for quick sort.
                        QuickSortAction leftTask =new QuickSortAction(arr, low, j-1);
                        QuickSortAction rightTask =new QuickSortAction(arr, j+1, high);
                        leftTask.fork();
                        rightTask.fork();
                        leftTask.join();
                        leftTask.join();
                }
        }

        private void swap(int[] arr, int i, int j) {
                int tmp = arr[i];
                arr[i] = arr[j];
                arr[j] = tmp;
        }

        //Generate random array.
        private static int[] buildRandomIntArray(final int size) {
                int[] array = new int[size];
                Random generator = new Random();
                for (int i = 0; i < array.length; i++) {
                        //Random value from 0 to 100.
                        array[i] = generator.nextInt(100);
                }
                return array;
        }
```

1) Explanation are those blue comments.

Application 2: Merge Sort:

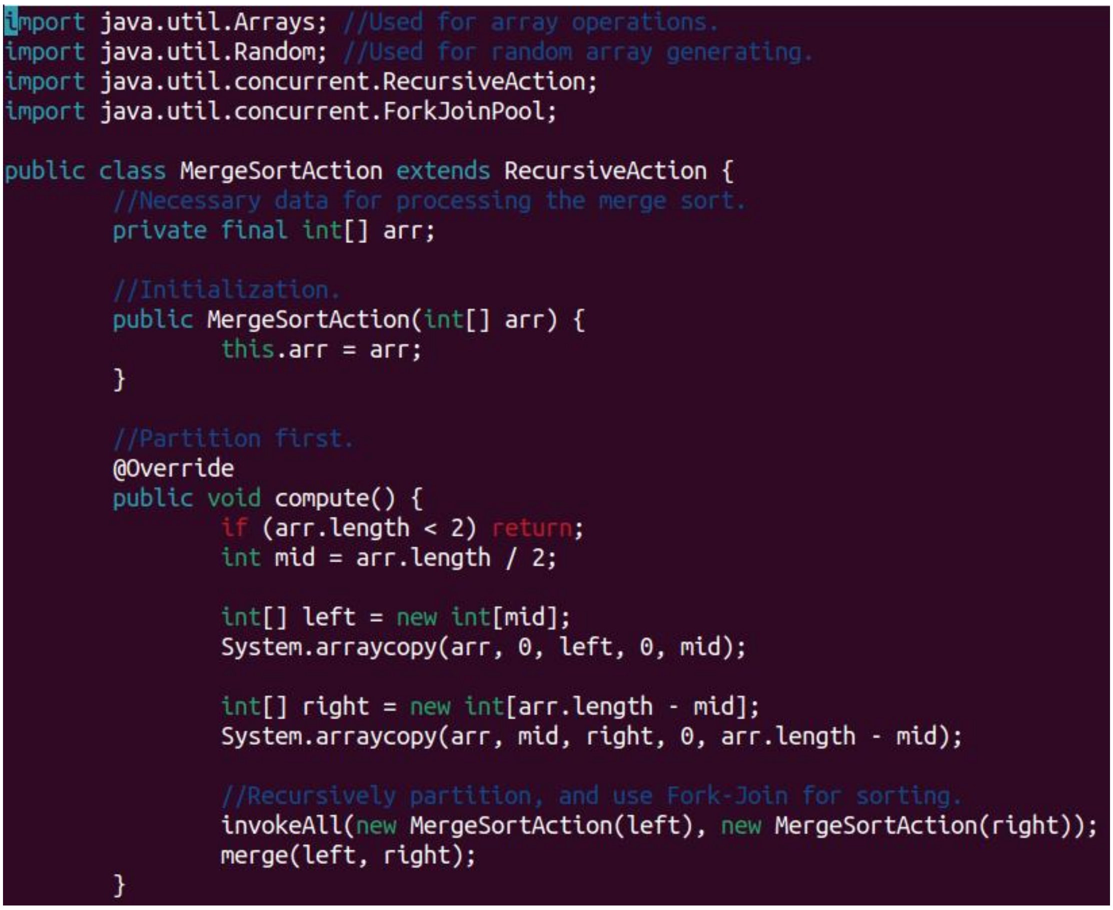
Main Function:



```
        //Main function.
        public static void main(String[] args) {
                ForkJoinPool pool = new ForkJoinPool();

                int[] array = buildRandomIntArray(20);
                System.out.println("A random array with length of 20:");
                System.out.println(Arrays.toString(array));

                //Start merge sort.
                MergeSortAction mergeSortAction = new MergeSortAction(array);
                pool.invoke(mergeSortAction);
                System.out.println("After Fork-Join Merge Sorting:");
                System.out.println(Arrays.toString(array));
                System.out.println("");
    }
}
```

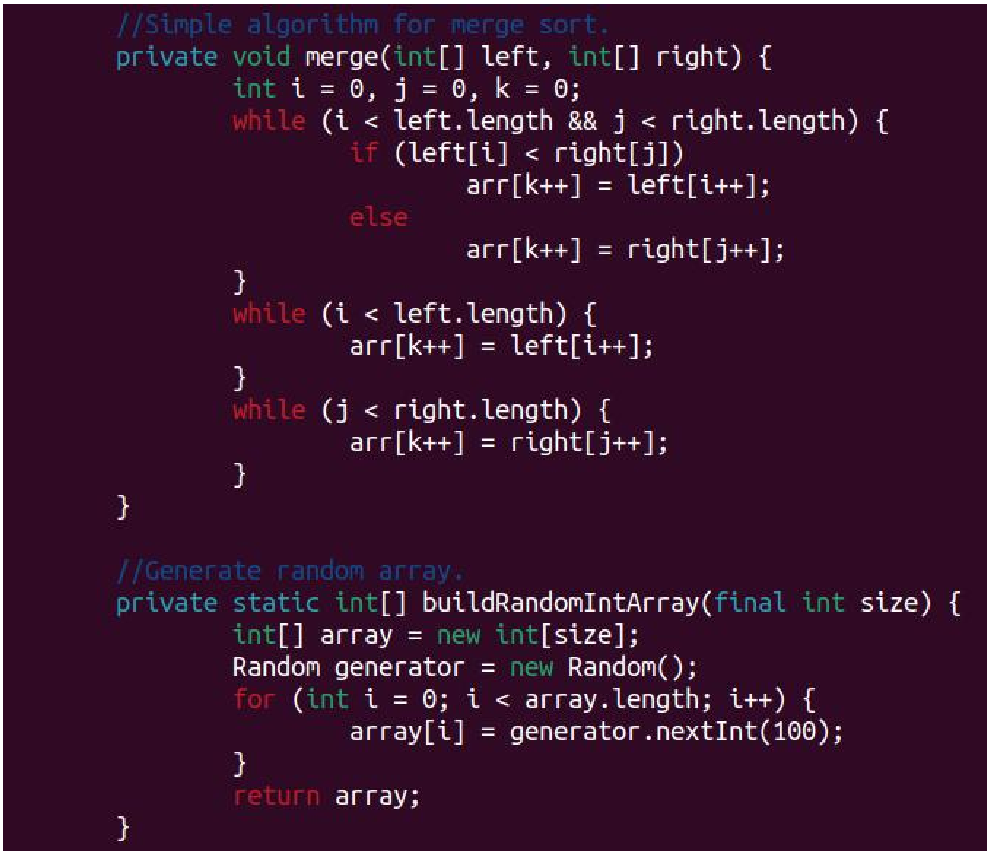
1) Almost the same idea with application 1 Quick Sort.

OtherCodes:



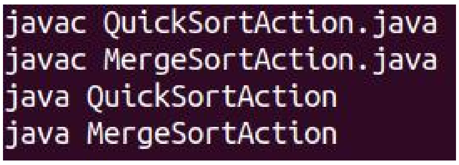
```
import java.util.Arrays; //Used for array operations.
import java.util.Random; //Used for random array generating.
import java.util.concurrent.RecursiveAction;
import java.util.concurrent.ForkJoinPool;

public class MergeSortAction extends RecursiveAction {
        //Necessary data for processing the merge sort.
        private final int[] arr;

        //Initialization.
        public MergeSortAction(int[] arr) {
                this.arr = arr;
        }

        //Partition first.
        @Override
        public void compute() {
                if (arr.length < 2) return;
                int mid = arr.length / 2;

                int[] left = new int[mid];
                System.arraycopy(arr, 0, left, 0, mid);

                int[] right = new int[arr.length - mid];
                System.arraycopy(arr, mid, right, 0, arr.length - mid);

                //Recursively partition, and use Fork-Join for sorting.
                invokeAll(new MergeSortAction(left), new MergeSortAction(right));
                merge(left, right);
        }
```

1) Recursively partition first, and use Fork-Join for sort by “merge(left, right)”.



```
	//Simple algorithm for merge sort.
	private void merge(int[] left, int[] right) {
		int i = 0, j = 0, k = 0;
		while (i < left.length && j < right.length) {
			if (left[i] < right[j])
				arr[k++] = left[i++];
			else
				arr[k++] = right[j++];
		}
		while (i < left.length) {
			arr[k++] = left[i++];
		}
		while (j < right.length) {
			arr[k++] = right[j++];
		}
	}

	//Generate random array.
	private static int[] buildRandomIntArray(final int size) {
		int[] array = new int[size];
		Random generator = new Random();
		for (int i = 0; i < array.length; i++) {
			array[i] = generator.nextInt(100);
		}
		return array;
	}
```

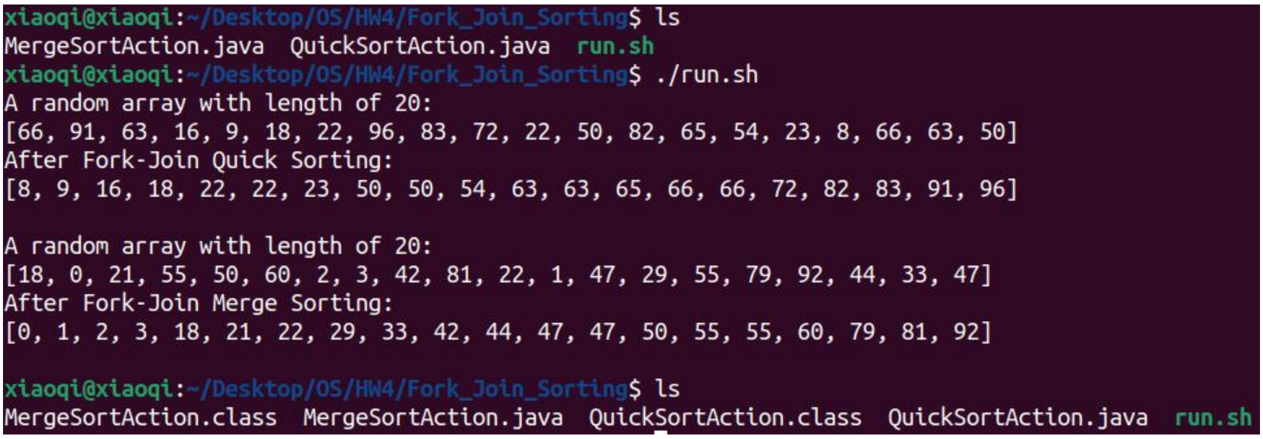
Compile (run.sh):



```
javac QuickSortAction.java
javac MergeSortAction.java
java QuickSortAction
java MergeSortAction
```

1) The 2 programs are compiled by a single file.
2) In Java, you need to use “javac” first to get a FileName.class file, then you can use “java” to run the program.
3) You can just use “./run.sh” to run the program, if you don’t have the permission to run it, try to use “chmod u+x FileName” to add your permission of running.

Results of 2 programs:



```
xiaoqi@xiaoqi:~/Desktop/OS/HW4/Fork_Join_Sorting$ ls
MergeSortAction.java  QuickSortAction.java  run.sh
xiaoqi@xiaoqi:~/Desktop/OS/HW4/Fork_Join_Sorting$ ./run.sh
A random array with length of 20:
[66, 91, 63, 16, 9, 18, 22, 96, 83, 72, 22, 50, 82, 65, 54, 23, 8, 66, 63, 50]
After Fork-Join Quick Sorting:
[8, 9, 16, 18, 22, 22, 23, 50, 50, 54, 63, 63, 65, 66, 66, 72, 82, 83, 91, 96]

A random array with length of 20:
[18, 0, 21, 55, 50, 60, 2, 3, 42, 81, 22, 1, 47, 29, 55, 79, 92, 44, 33, 47]
After Fork-Join Merge Sorting:
[0, 1, 2, 3, 18, 21, 22, 29, 33, 42, 44, 47, 47, 50, 55, 55, 60, 79, 81, 92]

xiaoqi@xiaoqi:~/Desktop/OS/HW4/Fork_Join_Sorting$ ls
MergeSortAction.class  MergeSortAction.java  QuickSortAction.class  QuickSortAction.java  run.sh
```

1) As you can see, after running the compiler “run.sh”, it will print the random array it made first, and then print the sorted array of Quick Sort and Merge Sort which uses the Fork-Join technique.

That’s the end of this report, thank you very much for your attention!
Xiaoqi LIU 999009335