CSC3100 Assignment1 Report

Problem 1

1. First, I used double loops with 2 pointers where pointer i in the outer loop starts from 0 and ends in n - 1, and pointer j in the inner loop starts from i + 1 and ends in n - 1, comparing the pointer i with the pointers j. If arr[i] > arr[j], then the disorder counter adds.It turns out that this simplest method doesn't work for the last two cases with a large amount of numbers.
2. To decrease the time complexity from

$$O(n^2). \tag{1}$$

I need a more efficient way of combining with the algorithm in merge sort whose time complexity is

$$O(nlgn). \tag{2}$$

3. Here is the code for merging

```
    public static long merge(long arr[],int left,int right) {
        int mid = (right + left) / 2;
        int length = right - left + 1;
        long [] temp = new long[length];
        int i = left;//pointer of left array
        int k = 0;//pointer of temp array
        int j = mid + 1;//pointer of right array
        long count = 0;// disorder counter
        while(i <= mid && j <= right && k < length){
            if(arr[i] <= arr[j]){
                temp[k++] = arr[i++];
            }else if(arr[j] < arr[i]){
                temp[k++] = arr[j++];
                count += mid - i + 1; // 3 4 7 2 1
              //when encountering an element in the right array is
  smaller
              //it means that it is smaller than the reminding
  elements of left array
```

```
                //counter adds mid - i + 1
            }
        }
        if((i<=mid || j<=right) && k < length){
            //there is still an array which have remainding elements
            while(i<=mid && k < length){
                temp[k++] = arr[i++];
            }
            while(j<=right && k < length){
                temp[k++] = arr[j++];
            }
        }
        for (int i1 = 0; i1 < length; i1++) {
            arr[left + i1] = temp[i1];
        }
        return count;
    }
}
```

Problem 2

1. Using recursion

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.*in*);
    long n = sc.nextLong();
    int a = sc.nextInt();
    int b = sc.nextInt();
    long f0 = sc.nextInt();
    long f1 = sc.nextInt();
    long m = sc.nextLong();
    long result = starSequence*(n,f0,f1,a,b,m);
    System.out.println(result);
```

```
public static long starSequence(long n,long f0,long f1, int a,int b,long m){
    if (n == 0) return f0 % m;
    if(n == 1) return f1 % m;
        long [] starSequence = new long[(int) n + 1];
        starSequence[0] = f0 % m;
        starSequence[1] = f1 % m;
        for (long i = 2; i < n+1 ; i++) {
            starSequence[(int)i] = (a * starSequence[(int)i - 1]) + (starSequence[(int)i - 2] * b);
        }
        return starSequence[(int)n];
}
```

However, the time complexity of this algorithm is

$$O(n) \tag{3}$$

which cannot solve the problem since the n is too large.

2. Considering matrix

$$\begin{bmatrix} a & b \\ 1 & 0 \end{bmatrix}, \tag{4}$$

left multiply to

$$\begin{bmatrix} f_1 \\ f_0 \end{bmatrix} \tag{5}$$

We get

$$\begin{bmatrix} f_2 \\ f_1 \end{bmatrix} \tag{6}$$

if we multiply n - 1 times

$$\begin{bmatrix} a & b \\ 1 & 0 \end{bmatrix}^{n-1} \tag{7}$$

then we get

$$\begin{bmatrix} f_n \\ f_{n-1} \end{bmatrix} \tag{8}$$

3. Instead of multiplying the matrix by itself `n-1` times, we use a binary exponentiation method to compute it in

$$O(lgn) \tag{9}$$

steps.

Here is the code

```
public static long starSequence(long n,long f0,long f1, int a,int b)
{
    if (n == 0) {
        return f0 % mod;
    }
    if (n == 1) {
        return f1 % mod;
    }

  long [][]transformationMatrix  = {{a,b},
                                    {1,0}};

    long [][] powTransformationMatrix =
quickPowMatrix(transformationMatrix,n-1);

    long fn = powTransformationMatrix[0][0] * f1 +
powTransformationMatrix[0][1] * f0;

    return fn % mod;

}
```

```java
private static long[][] quickPowMatrix(long[][] matrix, long order)
{
    long [][] resultMatrix = new long[matrix.length][matrix.length];
    //Assign identity matrix to resultMatrix first
    for (int i = 0; i < matrix.length; i++) {
        resultMatrix[i][i] = 1;
    }

  while(order != 0){
        if(order % 2 == 1){
            //R = R * A
            resultMatrix =
matrixMultiplication(resultMatrix,matrix);
        }
          //A = A*A
          matrix = matrixMultiplication(matrix,matrix);
          //binary: move right
        order = order/2;
    }
    return resultMatrix;
}
```