Homework 3

Jason Jiang
Dong Fang

February 13, 2019

This algorithm consists 3 procedures. The Get_Min_Inv is the main function that will call the recursive function, Divide_And_Swap, with the initial count of 0.

Then after the recursion, the Divide_And_Swap will call Merge_And_Count to count the number of inversions and decide whether to swap the two child.

The algorithm starts on Page 2.

The Merge_And_Count firstly start iterate the L and R with two different pointers. When find an inversion, add one to the counter c. Then if more than half of the situations leads to an inversion, we simply swap the two child and change the value of c to the better one.
Then if we swapped them, return an merged array with R before L. Else, return one with L before R.

The main recursion part of the program. The base case is n = 2 and we just need to compare the two number, keep the sequence or simply swap them if that could leads to a better situation. It is impossible to have an inversion because we can always solve that by swapping.
Then, we divide the array in two parts and call itself recursively. After reach the base case, we call the Merge function to calculate the inversions.

```
1:  Input: L[1..m] and R[1...m]
2:  Output: Merged array of L and R, count of inversions
3:  procedure MERGE_AND_COUNT(L,R)
4:      i ← 1
5:      j ← 1
6:      k ← 1
7:      c ← 0
8:
9:      while i ≤ m do
10:         while j ≤ m do
11:             if L[i] > R[j] and i <j then
12:                 c ← c + 1
13:             j ← j + 1
14:         i ← i + 1
15:     if c > m^2/2 then
16:         c ← m^2 − c
17:         i ← 1
18:         while i ≤ m do
19:             Merged[k] ← R[i]
20:             k ← k + 1
21:             i ← i + 1
22:         i ← 1
23:         while i ≤ m do
24:             Merged[k] ← L[i]
25:             k ← k + 1
26:             i ← i + 1
27:     else
28:         i ← 1
29:         while i ≤ m do
30:             Merged[k] ← L[i]
31:             k ← k + 1
32:             i ← i + 1
33:         i ← 1
34:         while i ≤ m do
35:             Merged[k] ← R[i]
36:             k ← k + 1
37:             i ← i + 1
38:     return Merged, c
```

```
1: Input: A[1...n] and non-negative integer c
2: Output: Merged array and count of inversions given A
3: procedure DIVIDE_AND_SWAP(A, c)
4:     if n = 2 then
5:         if A[1] > A[2] then
6:             Swap(A, 1, 2)
7:             return A, c
8:     else
9:         m ← n/2
10:        L, c ← Divide_And_Swap(A[1....m], c)
11:        R, c ← Divide_And_Swap(A[m + 1....n], c)
12:    Merged, inv_count ← Merge_And_Count(L, R)
13:    c ← c + inv_count
14:    return Merged, c
```

```
1: Input: A[1....n], with n > 0 and n%2 = 0
2: Output: The number of inversions
3: procedure GET_MIN_INV(A)
4:     Merged, count ← Divide_And_Swap(A[1....n], 0)
5:     return count
```