Homework02

Hwijong Im

Problem01

First, construct the Merge function that compares the left side and the right side to arrange the array. After that, the while method, which in the end of Merge function is crucial, because the condition of the while loop contains ((i<n1) && (j<n2)), and the intersection may left the end values. Therefore, the end of the while functions is essential to consider the remainders.

Problem02

Because of the while loop contains the condition of ((i<n1) && (j<n2)), it may leave the remainders that results can be iterated. Therefore, using the while functions as below, the while functions can be the role of sentinels that consider the end of values.

```
// The role of sentinel
while (i < n1) {
    /*
    the functions help to get the remainders and
    the role of sentinel(The end of values)
    */
    Arr[k] = Left[i];
    i++;
    k++;
}

while (j < n2) {
    Arr[k] = Right[j];
    j++;
    k++;
}
```

Problem03

(1) Both result of functions is power of 2. Even they work different ways, however, they are recursive.

(2) , (3) Case1 F1(n) and Let n=3,

F1(3) = 2*F(2) = 8

F1(2) = 2*F(1) = 4

F1(1) = 2*F(0) = 2*1 = 2

Therefore, O(F1(n)) = 2*(n-1) + 1 = 2n -1

Case2 F2(n) and Let n=3,
F2(3) = 2*F2(2) = 8
F2(2) = 2*(F2(2/2)) = 2F2(1) = 4
F2(1) = 2*F2(0) = 2*1 = 2
Tehrefore, O(F2(n)) = 2*(n/2) + 1
The result of O(n), F2(n) is much faster than F1(n)

Problem04

a. The purpose of PrcedureX is the sort function with ascending order. It places the least integer from i to j at position i, shifting the i pointer every time the first loop occurs. When the i passes every position in the array, the array has been sorted.
b. The worst scenario of time complexity is O(n^2). Because for n elements, each element needs to loop through an array up to size of n. Therefore, the time complexity becomes (n*n), or n^2

Problem05

The recursive of the insertion sort algorithm in pseudo code.

```
Recur_sort(A, i):
   if i=1: return
   Recur_sort(A, i-1)
   Inert_function(A, i-1)
```

The time complexity of this algorithm is O(n^2).
T(n) = T(n-1) + O(n)
= T(n-2) + O(n-1) + O(n)
= T(n-3) +O(n-2) + O(n-1) + O(n) → (It continues this pattern n times)
= O(n*n)
= O(n^2)