1 Exercise 4

1.1 (7 point) Design a pseudocode for a recursive insertion sort algorithm named Insertion_sort_recursive(A, n) and describe how it works.

Algorithm 1 Recursive insertion sort

```
1:  procedure INSERTION_SORT_RECURSIVE(A, n)                ▷ A[1...n]
2:      if n < 2 then
3:          return
4:      end if
5:      Insertion_sort_recursive(A, n − 1)
6:      key ← A[n]
7:      i ← n − 1
8:      while i > 0 and A[i] > key do
9:          A[i + 1] ← A[i]
10:         i ← i − 1
11:     end while
12:     A[i + 1] ← key
13: end procedure
```

삽입 정렬은 배열의 모든 elements를 왼쪽에서 부터 비교하여, key의 알맞은 위치를 찾고 이를 삽입해가며 final sorted array를 만들어 내는 알고리즘이다. 이러한 개념에 따르면, 알고리즘 수행 과정에서 key 왼쪽의 배열은 이미 정렬 되어있어야 한다. 이를 재귀적으로 접근하면,

길이가 n인 배열 A를 정렬해보자.

1. n 번째 element 까지 정렬되기 위해서는 먼저 $n-1$ 까지 정렬되어있어야 한다.
2. $n-1$ 번째 element 까지 정렬되기 위해서는 $n-2$ 까지 정렬되어있어야 한다.
3. ...
4. 3 번째 element 까지 정렬되기 위해서는 2 까지 정렬되어있어야 한다.

위와 같이 생각해 볼 수 있다.

따라서 위의 개념을 응용하면, 삽입 정렬 procedure 내에는 parameter n을 1 씩 줄여나가며 자기 자신을 호출하는 명령이 있어야 한다. 이를 5번 라인에 표현해 주었다. 또한 n이 2 보다 작을 경우(대표적으로 n=1)에는 정렬할 필요가 없기에 재귀의 종료조건에 해당한다. 이를 2번 라인에 표현해 주었다.

6번 라인부터는 for-loop을 활용한 삽입 정렬과 같은 코드이다. key값이 왼쪽에 있는 요소들 보다 작은지 비교하여, 올바른 위치에 삽입하는 역할을 한다.

위에 서술한 재귀적 접근의 개념이 실제 함수의 호출에 의해 이루어졌다고 가정하면, call stack 개념을 통해 호출과 반대인 $2, 3, \ldots, n-1, n$순서로 정렬될 것이다.

이러한 원리에 의해 위의 의사코드는 재귀적 방법을 통해 삽입 정렬을 수행 할 수 있다.

1.2 (8 point) Write a recurrence for the running time of this recursive version of insertion sort and explain how you get the recurrence. (You don't have to solve the recurrence.)

$$T(n) = \begin{cases} O(1), & \text{if } n = 1, \\ T(n-1) + O(n), & \text{if } n > 1 \end{cases}$$

재귀호출을 사용하는 삽입 정렬 알고리즘은 아래와 같은 로직을 가진다.

1. 부분 배열 A[1...n-1]을 정렬한다
2. A[n]을 1번에서 정렬된 부분 배열 A[1..n-1]에서 올바은 곳에 삽입한다.

만약 n이 1이라면 부분 배열을 가지지 않기 때문에 $T(1) = O(1)$이다. n이 1 보다 크다면, $A[1..n-1]$ 부분 배열을 가지기 때문에 $T(n-1)$이 걸릴 것이고, $A[n]$을 삽입할 곳을 찾는 과정에서 $O(n)$이 걸릴 것이다. 따라서 n이 1 보다 클 때는 $T(n-1) + O(n)$으로 표현할 수 있다.