알고리즘 Assign 01

과제 제출일자 : 2019/09/19
학번 : 201502085, 이규봉
분반 : 05
과제 : 삽입 정렬, 합병 정렬 코드 작성 및 연습 문제 풀이

Test Environment

과제의 코드를 테스트한 환경은 아래와 같습니다. 따로 사용한 라이브러리는 없습니다.

- Pycharm 2019.02
- Python 3.7.4

과제 결과는 main.py를 실행함으로써, 확인할 수 있습니다. main.py에선 ./data 폴더가 존재한다고 가정하며, 안에 data02.txt 파일이 존재하고 그 안에 한 줄의 숫자의 시퀀스가 존재한다고 가정합니다. 따라서, 이 이외의 상황에선 정상 작동하지 않을 수 있습니다.

연습문제 2.2-2

2.2-2
Consider sorting n numbers stored in array A by first finding the smallest element of A and exchanging it with the element in $A[1]$. Then find the second smallest element of A, and exchange it with $A[2]$. Continue in this manner for the first $n-1$ elements of A. Write pseudocode for this algorithm, which is known as ***selection sort***. What loop invariant does this algorithm maintain? Why does it need to run for only the first $n-1$ elements, rather than for all n elements? Give the best-case and worst-case running times of selection sort in Θ-notation.

선택정렬의 의사 코드를 작성해 본 것은 아래와 같습니다.

```
# Selection Sort

for i = 0 to A.length
    index = i

    for j = i + 1 to A.length
        if A[index] > A[j]
            index = j

    swap(A[i], A[index])
```

위 의사 코드를 C로 바꿔 작성한 후 코드가 어떻게 작동하는지 확인하기 위해 아래와 같이 크기 10의 배열을 이용해 테스트 해 보았습니다. (테스트에 사용한 배열은 { 88, 33, 222, 77, 11, 33, 0, 99, 33 ,66 } 입니다.)
한 번의 step에 한 번 최소값을 찾고 왼쪽 정렬되어 있는 부분에 넣습니다. 또한 한 번의 step에 최소값을 찾기 위해 A.length - i - 1 번의 비교가 필요합니다.

	A
1	0 33 222 77 11 33 88 99 33 66
2	0 11 222 77 33 33 88 99 33 66
3	0 11 33 77 222 33 88 99 33 66
4	0 11 33 33 222 77 88 99 33 66
5	0 11 33 33 33 77 88 99 222 66
6	0 11 33 33 33 66 88 99 222 77
7	0 11 33 33 33 66 77 99 222 88
8	0 11 33 33 33 66 77 88 222 99
9	0 11 33 33 33 66 77 88 99 222

위와 같이 크기 10의 배열을 선택 정렬하기 위해선 9번의 최소값을 찾는 step이 필요한 것을 알 수 있습니다.

이처럼, 10번의 step이 아닌 9번의 step이 필요한 이유는, n-1 번째 step이 종료된 시점에서 마지막 원소는 하나로, 다른 원소와 비교할 것이 없는 상태이며, 이미 배열이 정렬된 상태이기 때문입니다.

선택 정렬 알고리즘의 Invariant는 정렬되지 않은 배열에서 최소값을 찾아 순서대로 이어 붙여 나가면 정렬된 배열이 된다는 것입니다.

위에서 명시한 선택 정렬의 best case는, O(n^2) 입니다. 배열의 크기 -1 번 만큼 step을 밟아야 하고 (N-1) 한 번의 스텝마다 최소값을 찾기 위해 평균적으로 N/2 번 정도의 반복이 필요하기 때문입니다. 이는 best case로 이미 배열이 정렬되어 있는 경우에도 마찬가지로 적용됩니다. 최소값을 찾기 위해선 나머지 정렬되어 있지 않은 원소들을 모두 순회해야 하기 때문입니다.

선택 정렬은 worst case에도 마찬가지로 O(n^2) 입니다. 배열의 크기 -1 번 만큼 step을 밟아야 하고 (N-1) 한 번의 스텝마다 최소값을 찾기 위해 평균적으로 N/2 번 정도 반복한다는 점에서 위와 같은 방식으로 동작합니다.

위에서 의사코드로 명시한 선택정렬은 상한과 하한이 모두 O(n^2)이므로 Θ(n^2) 으로 동작함을 알 수 있습니다.

연습문제 2.3-6

2.3-6
Observe that the **while** loop of lines 5–7 of the INSERTION-SORT procedure in Section 2.1 uses a linear search to scan (backward) through the sorted subarray $A[1 \ldots j-1]$. Can we use a binary search (see Exercise 2.3-5) instead to improve the overall worst-case running time of insertion sort to $\Theta(n \lg n)$?

섹션 2.1의 의사코드 5-7 줄의 선형 탐색 대신 이진 탐색을 사용하면 비교에 걸리는 worst case running time을 lgn으로 줄일 수 있습니다. 1.. j-1 까지의 subarray가 정렬되어 있다는 것이 삽입 정렬의 Invariant이기 때문입니다. 그러나 삽입 정렬의 overall worst case running time이 Θ(nlgn) 이 되진 않습니다.

이진 탐색을 이용해 빠르게 삽입 위치를 찾아도 기존 array의 원소들을 모두 이동시켜야 하기 때문에 최대 n-1 번 정도의 swap 연산이 필요합니다. 따라서, 이진 탐색을 이용한 삽입 정렬 역시 기존 삽입 정렬과 마찬가지로 Θ(n^2) 의 시간 복잡도를 가질 것입니다.