ALGORITHMS

Assignment 1 Report

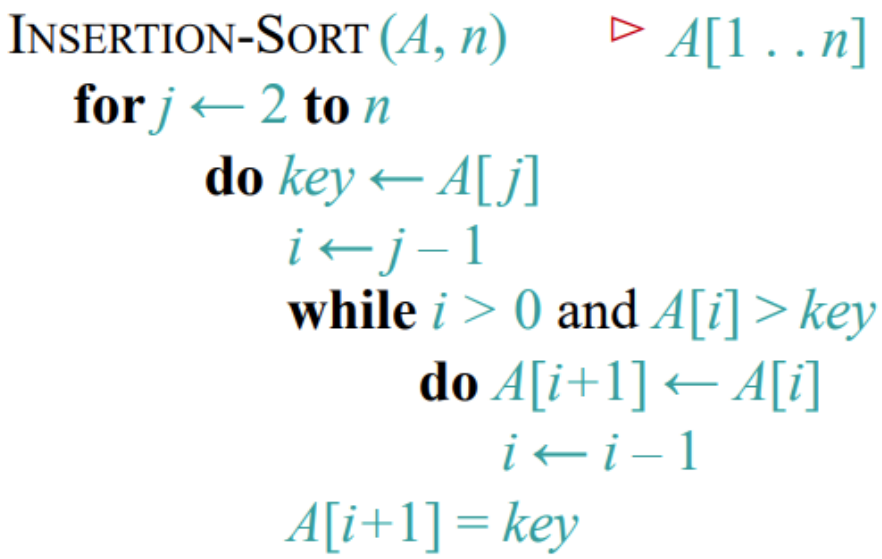
Group: 6

Student ID: 202020681 name: 성민규
Student ID: 202020673 name: 안관우
Student ID: 201921166 name: 정의철

March 19, 2021

1 Exercise 1

1.1 Describe how insertion sort(A, n) works.



0: input parameter 중 A에는 정렬할 수들이 담긴 배열이 들어가며, n에는 배열 A의 크기가 들어간다.
1: 비교 대상이 될 index를 나타내는 j의 초기값은 2이다. A배열의 j번째 원소와 j-1번째 원소를 비교하기 때문에 j가 1인 경우, 1번째와 0번째를 비교하는데, 0번째는 존재하지 않으므로 j의 초기값은 2가 된다.
2: 비교 대상이 되는 값이 들어가는 key에 A의 j번째 원소를 대입한다.
3: key와 비교하게 되는 index인 i에는 j-1을 대입한다.
4: i가 0보다 크고, A의 i번째 원소와 key를 비교하여 A의 i번째 원소가 더 크다면,
5: A의 i번째 원소를 A의 i+1 번째에 대입한다.
6: 그리고 i를 1 감소한다.
i가 0보다 크고, A의 i번째 원소와 key를 비교하여 A의 i번째 원소가 더 크다는 조건을 만족하지 않을 때 까지 이 과정을 반복한다.
7: 이후 A의 i+1번째 자리에 key값을 대입한다.
이제 j번까지는 오름차순으로 정렬된 상태가 되었다.
j를 1 증가시키며 2 8 과정을 반복한다.
j에 n을 대입하여 위 과정을 수행하면 배열 A가 오름차순으로 정렬된 상태가 되며, Insertion Sort가 종료된다.

1.2 Show the correctness of insertion sort(A, n) using loop invariants.

Initialization:
The first iteration begins by assuming that a list of size 1 is already in sorted order, and starts off by assigning j a value 2. A list of size 1 is always in sorted order.

Maintenance:
Our examination of the behavior of the inner loop shows that if the j th element of the array is out of order when entering the inner loop, then it will be in the correct order when exiting the loop.
The inner loop is basically all that the outer loop does during one iteration.

Termination:
The loop terminates when j = n + 1. Just prior to that point, while j = n, all j (and thus all n) of the elements of the array are in sorted order.
So, the termination condition of the loop preserves the sorted order of the array.

1.3 Analyse the worst case run time of insertion sort(A,n).

insertion sort의 the worst case는 배열이 내림차순으로 정렬되어 있는 경우에 발생한다.
예를 들어 A = [5,4,3,2,1] 인 경우, key는 j-1번의 비교가 발생한다.
이를 T(n)으로 나타내면, $T(n) = c_1 \cdot n + c_2(n-1) + c_4(n-1) + c5 \cdot \sum_{j=2}^{n} t_j + c_6 \cdot \sum_{j=2}^{n}(t_j - 1) + c_7 \cdot \sum_{j=2}^{n}(t_j - 1) + c_8 \cdot (n-1)$ 이다.

여기서 $\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} j$라 하면, $\sum_{j=2}^{n} t_j = \sum_{j=1}^{n} j - 1 = \frac{n(n+1)}{2} - 1$ 이고
$k = j - 1$이라 하면,
$\sum_{j=2}^{n}(t_j - 1) = \sum_{j=2}^{n}(j-1) = \sum_{k=1}^{n-1} k = \frac{(n-1)n}{2}$ 이므로
$T(n) = c_1 \cdot n + c_2(n-1) + c_4(n-1) + c5 \cdot (\frac{n(n+1)}{2} - 1) + c6 \cdot (\frac{(n-1)n}{2}) + c7 \cdot (\frac{(n-1)n}{2}) + c_8 \cdot (n-1) = (\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2})n^2 + (c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8)n - (c_2 + c_4 + c_5 + c_8)$
$= an^2 + bn + c$ (단, a, b, c 는 constant)이다.

여기서 $an^2 + bn + c$ 를 간단히 n^2으로 나타낼 수 있으며, the worst case run time of insertion sort(A,n)은 $O(n^2)$이라고 말할 수 있다.

2 Exercise 2

Big O notation For two functions f , g : $N \to N$ we write $f \in O(g)$ if there exists a positive constant c and $n_0 \geq 0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n0$. For each of the following pairs of functions f and g, show $f \in O(g)$ and $g \in O(f)$

2.1 $f(n) = 2n^3 + 10n^2$ and $g(n) = 3n^3$

To show $f \in O(g)$,
We claim that there exist c > 0 and $n_0 \geq 0$.
That make inequality $2n^3 + 10n^2 \leq 3n^3$ holds for all $n \geq n_0$.

We choose $c = 4$ and $n_0 = 1$.
Then we have $f(n) = 2n^3 + 10n^2 \leq c \cdot g(n) = 12n^3$
$\frac{1+5}{n} \leq 6$,
$\frac{5}{n} \leq 5$,
$1 \leq n$
Thus, $f(n) \leq c \cdot g(n)$ for all $n \geq 1$

To show $g \in O(f)$,
We claim that there exist c > 0 and $n_0 \geq 0$.
That make inequality $3n^3 \leq 2n^3 + 10n^2$ holds for all $n \geq n_0$.

We choose $c = \frac{3}{2}$ and $n_0 = 1$.
Then we have $g(n) = 3n^3 \leq c \cdot f(n) = 3n^3 + 15n^2$
$1 \leq 1 + \frac{5}{n}$,
$0 \leq \frac{5}{n}$,
$0 < n$
Thus, $g(n) \leq c \cdot f(n)$ for all $n \geq 1$

2.2 $f(n) = n^3 + 100n^2$ and $g(n) = n^3$

To show $f \in O(g)$,
We claim that there exist c > 0 and $n_0 \geq 0$.
That make inequality $n^3 + 100n^2 \leq n^3$ holds for all $n \geq n_0$.

We choose $c = 101$ and $n_0 = 1$.
Then we have $f(n) = n^3 + 100n^2 \leq c \cdot g(n) = 101n^3$
$\frac{1+100}{n} \leq 101$,

$\frac{100}{n} \leq 100$,
$1 \leq n$
Thus, $f(n) \leq c \cdot g(n)$ for all $n \geq 1$

To show $g \in O(f)$,
We claim that there exist c > 0 and $n_0 \geq 0$.
That make inequality $n^3 \leq n^3 + 100n^2$ holds for all $n \geq n_0$.

We choose $c = 1$ and $n_0 = 1$.
Then we have $g(n) = 3n^3 \leq c \cdot f(n) = 3n^3 + 15n^2$
$1 \leq 1 + \frac{100}{n}$,
$0 \leq \frac{100}{n}$,
$0 < n$
Thus, $g(n) \leq c \cdot f(n)$ for all $n \geq 1$

2.3 $f(n) = \frac{1}{100}n^2$ and $g(n) = 50n^3 - 10^{10}n$

To show $f \in O(g)$,
We claim that there exist c > 0 and $n_0 \geq 0$.
That make inequality $\frac{1}{100}n^2 \leq 50n^2 - 10^{10}n$ holds for all $n \geq n_0$.

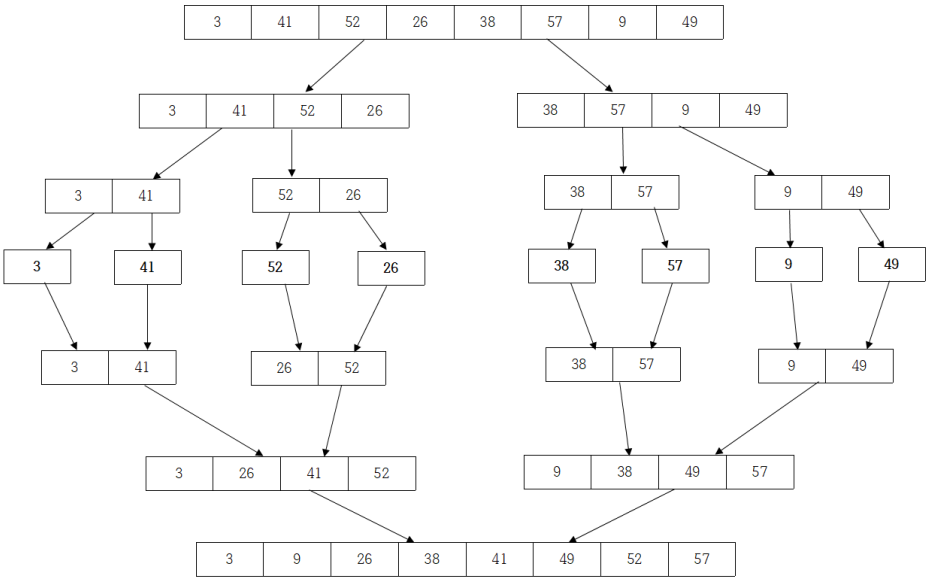
We choose c = 1 and $n_0 = 10^9$.
Then we have $f(n) = \frac{1}{100}n^2 \leq c \cdot g(n) = 50n^2 - 10^{10}n$
$n \leq 5000n - 10^{12}$,
$10^{12} \leq 4999n$,
$\frac{10^{12}}{4999} \leq n$, $\frac{10^{12}}{4999} \leq 10^9$
Thus, $f(n) \leq c \cdot g(n)$ for all $n \geq 10^9$

To show $g \in O(f)$,
We claim that there exist c > 0 and $n_0 \geq 0$.
That make inequality $50n^3 - 10^{10} \leq \frac{1}{100}n^2$ holds for all $n \geq n_0$.

We choose $c = 5000$ and $n_0 = 1$.
Then we have $g(n) = 50n^3 - 10^{10} \leq c \cdot f(n) = 50^2$
$1 - \frac{10^{10}}{50n} \leq 1$,
$0 \leq \frac{10^{10}}{50n}$,
$0 < n$
Thus, $g(n) \leq c \cdot f(n)$ for all $n \geq 1$

3 Exercise 3

3.1 illustrate the operation of merge-sort on the array A = [3, 41, 52, 26, 38, 57, 9, 49] and describe how merge-sort works.



merge-sort는 divide and conquer 방식으로 작동한다.

MERGE-SORT (*A*[1 . . *n*], p, r)

1. If p < r,
2. q ←⌊*p* + *r* / 2⌋
3. MERGE-SORT(A, p, q)
4. MERGE-SORT(A, q+1, r)
5. MERGE(A, p, q, r)

MERGE-SORT(A[1..n], p, r)
우선 정렬할 배열 A[1..n]과 배열의 시작을 나타낼 p와 끝을 나타낼 r을 인풋값으로 가진다.
p<r을 만족시킬 때, p와 r의 중간값인 q를 이용해 배열을 분할해 배열의 원소의 개수가 1개 이하가 될 때까지 분할되게 한다.
3번째 줄에서 p, q를 사용해 왼쪽배열으로 나누고, 4번째 줄에서 q+1, r을 이용해 오른

쪽배열로 나눠진다. p<r을 만족시키지 못하는, 원소의 개수가 1개 이하까지 재귀적으로 호출되며 분할이 끝나면 MERGE(A, p, q, r)을 이용해 정렬된 배열들을 합병해 하나의 배열로 만든다.

MERGE(A, p, q, r)

```
1   n1 ← (q - p) + 1
2   n2 ← (r - q)
3   create arrays L[1..n1+1] and R[1..n2+1]
4   for i ← 1 to n1 do
5     L[i] ← A[(p + i) -1]
6   for j ← 1 to n2 do
7     R[j] ← A[q + j]
8   L[n1 + 1] ← ∞
9   R[n2 + 1] ← ∞
10  i ← 1
11  j ← 1
12  for k ← p to r do
13    if L[I] <= R[j]
14          then A[k] ← L[i]
15               i ←i + 1
16          else A[k] ← R[j]
17               j ← j + 1
```

MERGE(A, p, q, r)
1 - 2 : n1에 왼쪽L 배열의 길이인 (q-p) + 1, n2에 오른쪽R 배열의 길이인 (r-q)를 대입한다.
3 - 7 : L, R 배열을 생성하고 반복문을 사용해 분할됐던 왼쪽, 오른쪽 배열으로 채운다.
8 - 11 : 끝을 표시하기 위해 L, R배열의 끝에 ∞, 시작을 표시하기 위해 i=1, j=1을 추가한다.
12 - 17 : 반복문을 이용해 L[i]와 R[j]를 비교하며 더 작은 값을 A 배열에 넣어준다. 이때 L에서 작은 값이 나왔다면 i에 1을 더해 L배열의 다음 인덱스와 R[j]를 비교할 수 있게 하고, 같은 방식으로 R에서 작은 값이 나왔다면 j에 1을 더해준다.
반복문 종료 후 길이가 (r-p) + 1인 정렬된 A배열이 결과로 나오게된다.

3.2 prove the correctness of merge-sort on the array A

배열 A를 인풋으로 가지는 merge-sort의 correctness를 증명하려면, merge 과정에서 12-17 줄에 해당하는 루프 시작 전, 루프 중, 루프 종료 시의 sub-array A[p..k-1]가 올바르게 정렬되어 있음을 보이면 된다. 이때의 sub-array A[p..k-1]는 루프가 진행되면서 L배열과 R배열에서 가장 작은 값들을 가져와 k-p번째까지 쌓여지는 배열을 뜻한다.

초기조건 :
루프 시작 전에는 sub-array A[p..k-1]가 비어있을 것이다.

반복문이 진행되며 L과 R배열을 비교해서 작은 값을 A에 넣는 과정이 나오는데 반복문이 시작되지 않았기 때문에 A는 비어있다.
비어있는 sub-array A는 L배열과 R배열의 원소 중 가장 작은 원소를 가지는 것이 보장된다고 말할 수 있다. 즉, sub-array A는 올바르게 정렬되어 있다.
유지조건 :
루프가 진행되며 이미 L과 R배열에서 k-p번째까지 가장 작은 수들이 정렬된 sub-array A에 L과 R배열을 비교해 가장 작은 값을 넣게될 것이다.
L[i] $\leq$ R[j] 일때, 이미 정렬이 되어있는 sub-array A에 L[i]값이 추가될 것이고
L[i] $\geq$ R[j] 라면, 이미 정렬이 되어있는 sub-array A에 R[j]값이 추가될 것이다.
결과적으로 A는 k-p+1번째로 작은 원소까지 올바르게 정렬된 배열이 된다.
종료조건 :
루프가 진행되다가 k = r + 1이 되면 종료된다.
즉, r = k -1가 되고 루프가 종료되면 결과적으로 전체 배열 A[p..r]가 나오게 된다.
유지조건을 통해 루프가 진행되고 난 후의 A 배열이 올바르게 정렬되어진다는 것을 알 수 있었고 결국 마지막 루프가 진행된 후 나오게된 A[p..r]도 올바르게 정렬되어 있음을 알 수 있다.

4 Exercise 4

4.1 (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 1씩 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.

4.2 (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.