실습과제 1.1

[과제 설명]

n x n 행렬이 주어졌을 때 이 행렬의 전치 행렬을 구하는 알고리즘을 의사코드로 기술하고, 해당 알고리즘의 시간 복잡도를 계산하여 빅오 표기법으로 표현한다. main부터 알고리즘 전체를 기술하는 것이 아닌, 조건에 있는 함수 2개 중 Transpose_Mat 함수에 대한 알고리즘만 기술한다.

함수의 입력은 정수 값으로 구성된 정방행렬이고, 출력은 이 행렬의 행과 열이 바뀐 정방행렬이어야 한다.

[알고리즘(의사 코드)]

```
Algorithm Transpose_Mat(A[][배열의 크기], B[][배열의 크기])
Input: 5 x 5의 배열 A
Output: A의 전치행렬 B

for int i <- 0 to 4 do
  for int j <-0 to 4 do
    B[i][j] <- A[j][i]
```

[시간 복잡도]

O(n^2) = (n+1)+n*(n+1)+n*n = 2n^2+2n+2

for문 이외에는 다른 변수를 초기화하거나 수행하는 코드는 없기 때문에, 시간 복잡도는 for문에 대해서만 체크하면 된다.

배열의 크기를 n으로 볼 때, 가장 바깥의 for문은 조건을 검사하고 중지될 때까지 n+1번이 실행되고 s/e는 1이므로 n+1이 된다. 안쪽의 반복문은 바깥의 반복문이 1번 돌아갈 때마다 n+1번이 돌아가기 때문에 n(n+1)만큼 돌아가고, 가장 안쪽의 실행문은 반복문이 각각 n번씩 돌아갈 때마다 1번씩 수행되기 때문에 횟수는 n*n번이 된다. 모두 s/e는 1이기 때문에 시간 복잡도는 총 2n^2+2n+2이 된다.

n^2가 최고차항이기 때문에 O(n^2)로 표기했다.

[과제에 대한 고찰]

for문을 작성할 때 보통 for(int i = 0; i<행렬의 크기; i++)로 해서 의사코드를 적을 때도 for i <-0 to 5 do로 할 수도 있지만, 실제로는 0부터 4까지 돌아가기 때문에 for i <-0 to 4 do 로 하는 게 더 명확하다. 또한 교안에 있는 예제처럼 반복문에 쓰일 i 같은 변수를 for문 안쪽에 따로 선언할 수도 있지만, for문 안에 같이 초기화시키는 방식을 많이 쓰기 때문에 for int i <-0 to 4 do의 방식으로 기술했다.

앞서 서술한 것도 그렇지만, 이렇게 실제로 코드와 적을 때와 의사 코드로 적을 때는 다소 다른 방식으로 기술한다. 게다가 평소에는 알고리즘을 딱히 적지 않고 바로 코드로 짜고 나서 실행이 제대로 되지 않으면 보충하는 방식으로 했었기 때문에 이번에 알고리즘을 정형화된 방식으로 나타내는 것이 힘들었다. 하지만 알고리즘을 한 번 적고 나서 코드를 작성하는 방식으로 하면 실수를 줄이게 되고 어디가 틀렸는지 등을 조금 더 쉽게 파악할 수 있는 것 같다.

이번은 간단한 문제여서 오히려 알고리즘을 적는 게 난해하고 불필요하게 느껴졌을 수도 있다. 하지만 복잡한 문제일 경우 알고리즘을 의사코드로 따로 적는 것이 도움이 될 것 같다는 걸 알게 되었다.