2016 Fall EE205

**Project #1 (Small Project, 50 pts):**

**C Practice with Hanoi Tower Puzzle Report**

**이름: 김경만**

**학번: 20150073**

**Ⅰ. 알고리즘 설명**

RECURSIVE

n개의 하노이 탑을 a에서 b를 경유하여 c까지 옮기는 과정을 출력하는 함수를 hanoi\_recursive(n, a, b, c)라 할 때 함수 hanoi\_recursive의 정의 안에 이 함수를 사용함으로써 recursive한 알고리즘입니다. n개의 하노이 탑을 a에서 c까지 옮기는 과정은 n-1개의 하노이 탑을 b로 옮긴 후 마지막 가장 큰 n번째 원반을 c에 옮긴 다음 다시 b에 있는 n-1개의 하노이 탑을 c로 옮기는 것입니다. 즉 hanoi\_recursive(n, a, b, c) 안에 hanoi\_recursive(n-1, a, c, b)와 hanoi\_recursive(n-1, b, a, c)가 사용되는 것입니다. 이러한 알고리즘은 n이 1보다 커야 되기 때문에 항상 함수를 실행하기 전에 n이 1인가를 확인하여 n이 1이라면 가장 작은 첫번째 원반을 옮기는 출력을 해야합니다.

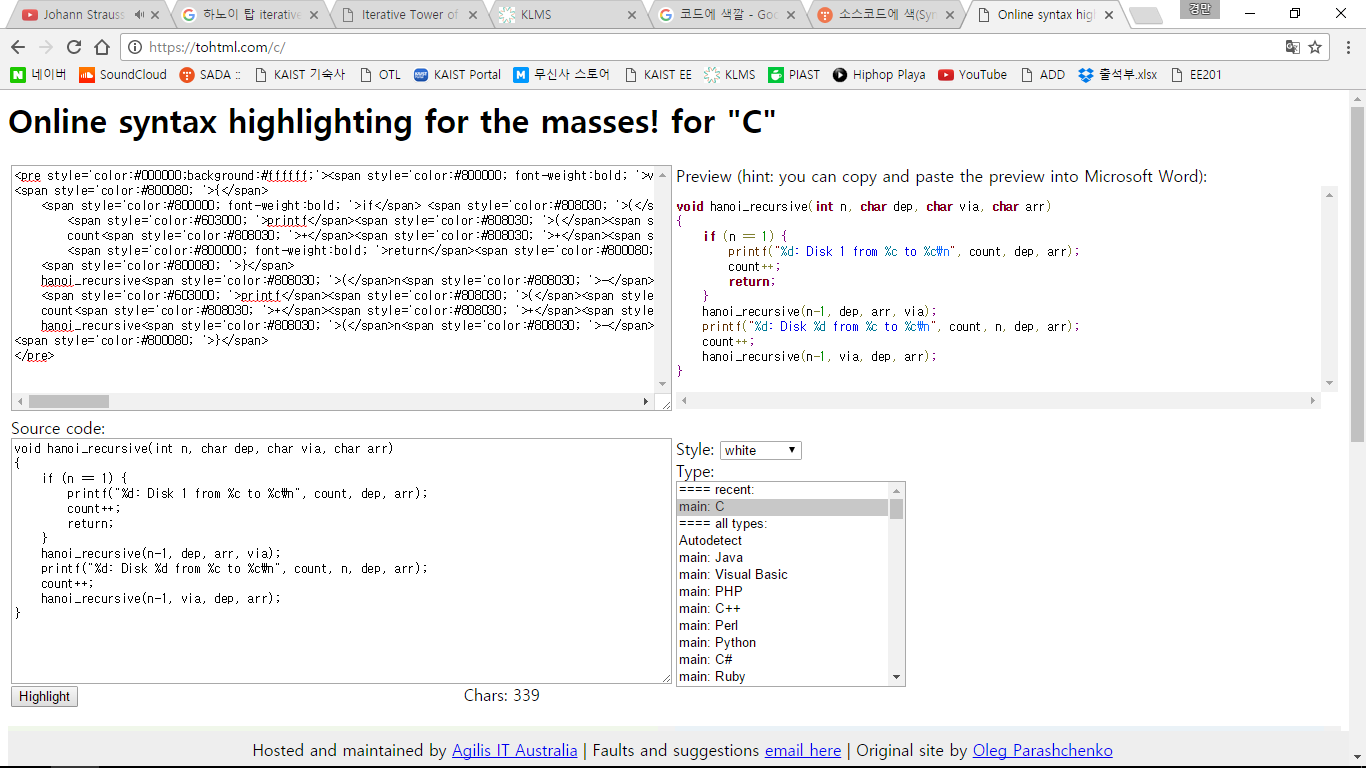
ITERATIVE

가장 작은 첫번째 원판은 항상 세 막대 중 하나에 맨 위에 꽂혀 있습니다. 그렇기 때문에 가장 작은 첫번째 원판이 꽂혀 있는 막대를 제외한 다른 두 막대에서 적절한 움직임이 있었다면 그 다음은 무조건 가장 작은 첫번째 원판이 움직여야 합니다. 왜냐하면 다른 원판을 가장 작은 원판 위에 올려놓지 못하며 방금 움직인 원판은 가장 최적의 움직임으로 하노이 탑을 옮긴다 할 때 또 움직일 수 없기 때문입니다. 방금 움직이지 않고 가장 작지도 않은 원판 역시 방금 움직인 원판 밑에 있었기 때문에 더 크며 그렇기 때문에 어디에도 옮길 수 없습니다. 그러므로 가장 작은 원판은 항상 두 막대 사이에서 적절한 움직임을 하면 한번의 움직임 후 그 다음 기회에서 무조건 움직여야 합니다. 그러므로 항상 막대 사이에서 적절한 움직임을 하면 이 막대 쌍은 총 세 쌍이 있으며 돌아가면서 반복하게 됩니다.

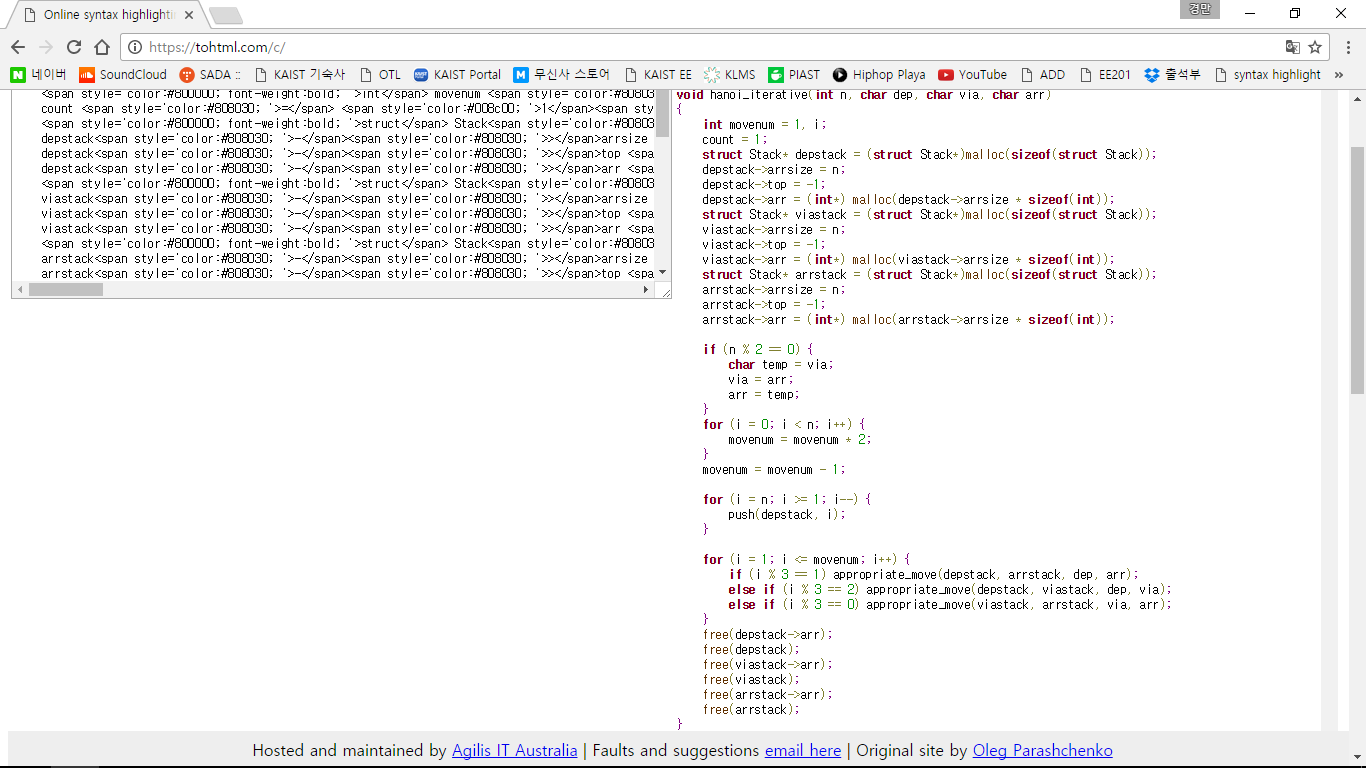
그래서 최적의 움직임이여야 하므로 n개의 원판일 때 (2^n)-1번의 움직임에 대해서 출발-도착, 출발-경유, 경유-도착의 세 쌍에서의 적절한 움직임을 반복해야 합니다. 여기서 적절한 움직임은 두 막대가 있을 때 큰 것이 작은 것 위에 올라가지 않는, 즉 상대적으로 더 작은 막대가 반대편 막대 위에 올라가는 것입니다.

**Ⅱ. 코드**

RECURSIVE



ITERATIVE



appropriate\_move(stack, stack, rod, rod)는 두 막대 사이의 적절한 움직임을 실행하는 코드입니다. 적절한 움직임은 위에 있는 원반 중 상대적으로 더 작은 원반이 반대 막대로 옮겨가는 것 입니다.

**Ⅲ. 결과 분석**

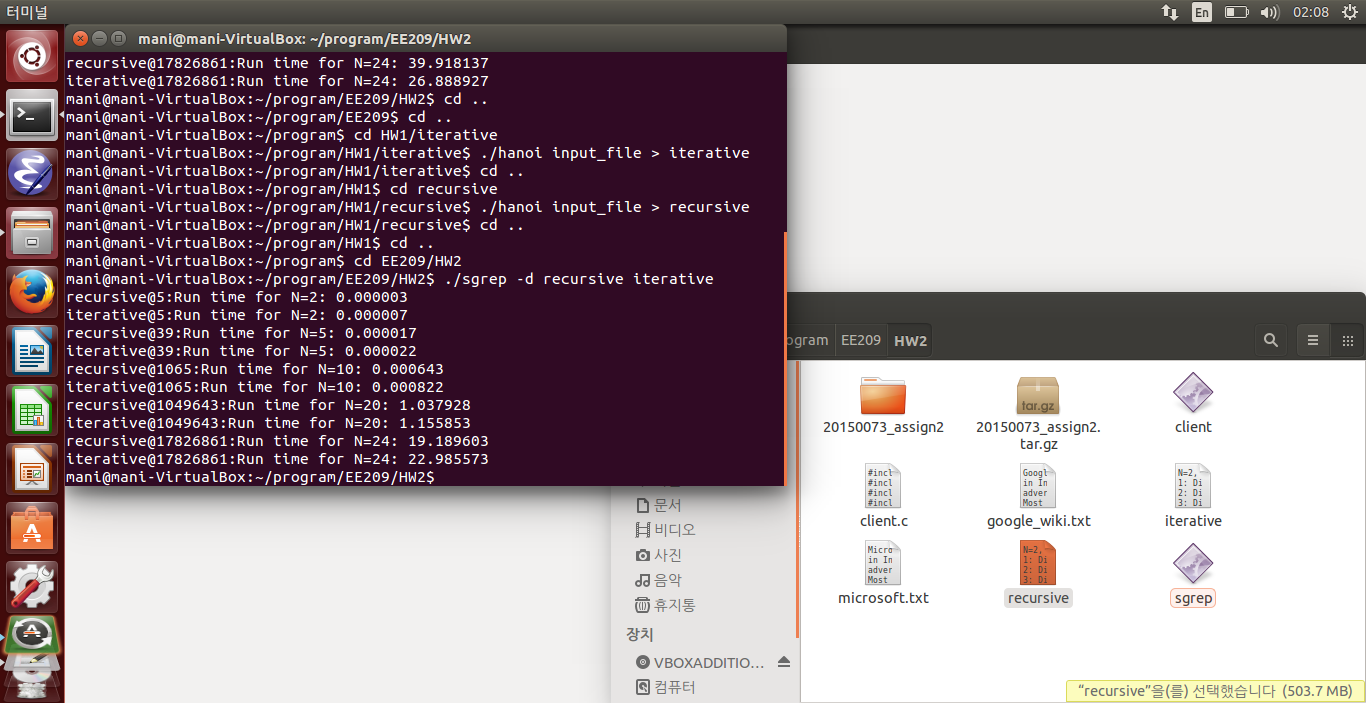
EXPECTED

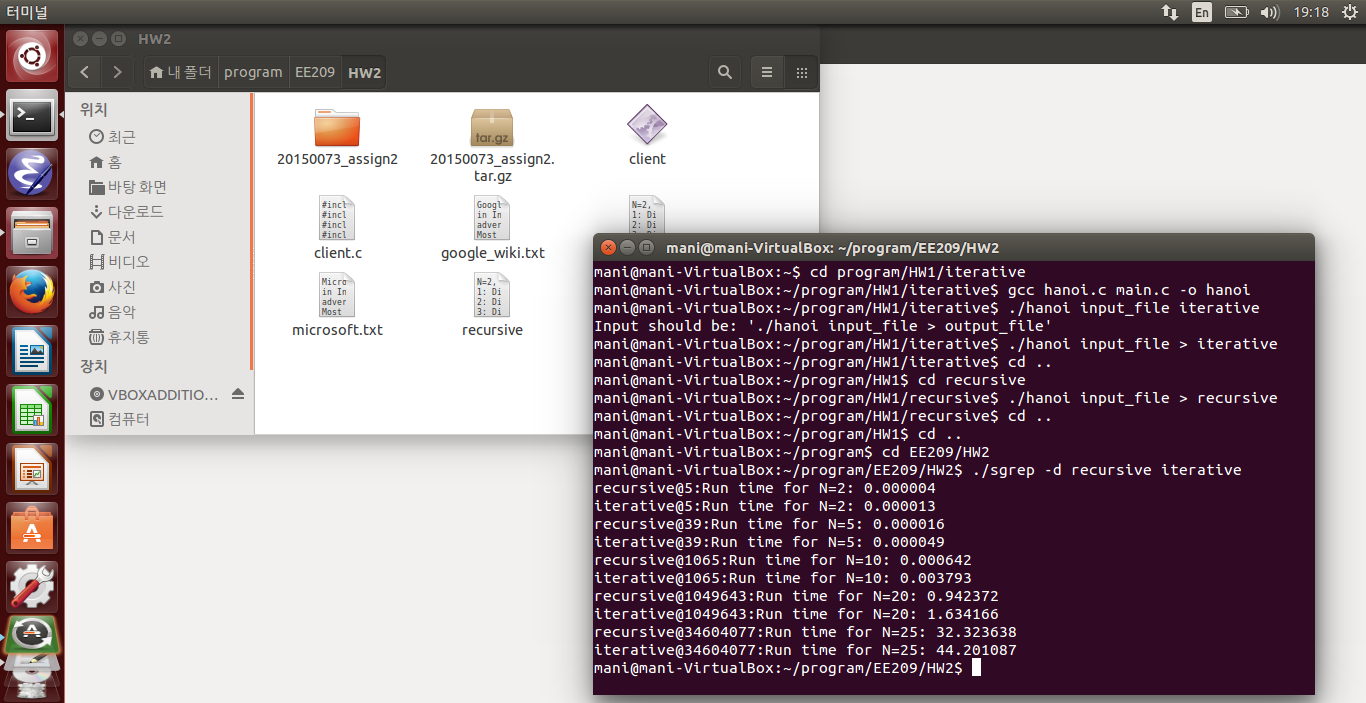
두 알고리즘의 시간 복잡도를 계산하기 위해 primitive operations를 세어보면 우선 recursive는 첫번째 함수 호출에서 두 번의 함수 호출, 그리고 과정 출력과 count 증가로 두 번의 연산을 합니다. 총 2^2번이며 호출된 두 번의 함수가 또 각각 네 번의 함수를 호출하며 과정 출력, count 증가를 하기 때문에 트리처럼 그려 나가며 수를 계산해보면 대략 2^2부터 맨 마지막에 2^n개 함수가 1이 되어 두 번의 연산을 하게 됩니다. 즉, 다 더하면 대략 2^(n+2)정도 됩니다. 시간복잡도를 빅오표기법으로 나타내면 c=4, n0=1일 때 1보다 큰 모든 n에 대해서 2^(n+2)<=4\*(2^n)을 만족하기 때문에 O(2^n)이라고 할 수 있습니다.

iterative는 알고리즘의 핵심인 반복문의 primitive operations를 세어보면 총 반복 횟수는 (2^n)-1이며 이 반복문 안에서 반복 횟수를 3으로 나누었을때의 결과를 비교합니다. 처음엔 한번 비교로 끝나며 그 다음은 두번의 비교, 그 다음은 세번의 비교 끝에 if문을 통과하기 때문에 평균적으로 2번 비교하게 되는 것입니다. 또한 그 후 적절한 움직임을 해주는 함수를 호출해 주기 때문에 한번의 반복문 안에서 비교와 호출, 총 두 번의 연산이 이루어 집니다. 그렇기 때문에 역시 계산해보면 iterative 역시 대략 2^(n+2)의 연산 횟수를 가지고 있으며 빅오표기법으로는 마찬가지로 O(2^n)이 됩니다.

하지만 둘 중에 누가 더 빠른지를 비교해보면 recursive가 될 것입니다. 왜냐하면 iterative는 함수를 호출한 뒤 적절한 움직임을 정하기 위해 appropriate\_move()함수 안에서 몇번의 원판의 크기 비교와 출력 등이 이루어 지기 때문입니다.

RESULT





input\_file에 n을 2, 5, 10, 20, 24를 입력 한 뒤 각각의 알고리즘에 대해 프로그램을 실행했습니다. 그 후 두 output\_file을 두 텍스트 파일을 한 줄씩 비교하여 만약 서로 다르면 몇 번째 줄 인지와 그 줄의 내용이 출력되는 프로그램을 돌렸습니다. 우선 하노이 탑이 옮겨지는 과정은 두 알고리즘 모두 같은 결과를 출력하였기 때문에 터미널에는 두 텍스트 파일의 다른 줄이 Run Time만 있는 것으로 나왔습니다.

또한 Run Time을 비교해보면 큰 차이는 없지만 예상했던 결과와 같이 항상 iterative가 약간 더 오래 걸렸으며 n이 커질수록 iterative와 recursive의 시간 차이가 더 커짐을 볼 수 있었습니다. 이는 iterative와 recursive가 모두 O(n^2)의 시간복잡도를 가지고 있지만 iterative한 알고리즘이 for문 안에서 또 적절한 움직임을 위해 함수를 호출하며 이 함수에서 적절한 움직임을 계산하기 위해 각각의 스택의 가장 위에 있는 원반 크기를 비교하는 과정이 recursive는 없는데 반해 iterative는 추가로 있기 때문이라고 생각합니다.