자료구조 과제 1

20211533 남정연

1. 코드 설명

```
#include <iostream>
#include <time.h>

using namespace std;

// SWAP 함수
// a 와 b 를 reference 로 입력받아 a 와 b 의 값을 서로 바꾼다.
void SWAP(char &a, char &b, int &temp) {
    temp = a;
    a = b;
    b = temp;
}

// perm 함수
// list 와 시작 인덱스, 끝 인덱스를 입력받아 permutation 을 수행한다.
void perm(char *list, int i, int n) {
    int j, temp;
    // 시작 인덱스와 끝 인덱스가 일치하면 정지
    if(i==n) {
        //for(j=0; j<=n; j++) cout << list[j];
        //cout << "  ";
    } else {
        // 시작 인덱스부터 끝 인덱스까지 순회하면서
        for(j=i; j<=n; j++) {
            // 시작 인덱스와 현재 인덱스의 값을 바꿈
            SWAP(list[i], list[j], temp);
            // 시작 인덱스를 1 늘리고 perm 호출
            perm(list, i+1, n);
            // 바꿨던 인덱스를 원상 복귀
            SWAP(list[i], list[j], temp);
        }
    }
}

int main() {
    // 소요 시간
    double duration;
    // 리스트
    char list[12] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k',
'l'};
    // 1~12 개의 permutation 테스트
    for(int i=0;i<12;i++) {
```

```
        long repetitions = 0;
        clock_t start = clock();
        do {
            repetitions++;
            // permutation 수행
            perm(list, 0, i);
        } while(clock() - start < 1000);
        duration = (double) (clock() - start) / CLOCKS_PER_SEC;
        duration /= repetitions;

        // 반복 횟수 및 소요 시간 출력
        cout << repetitions << " " << duration << endl;
    }
    return 0;
}
```

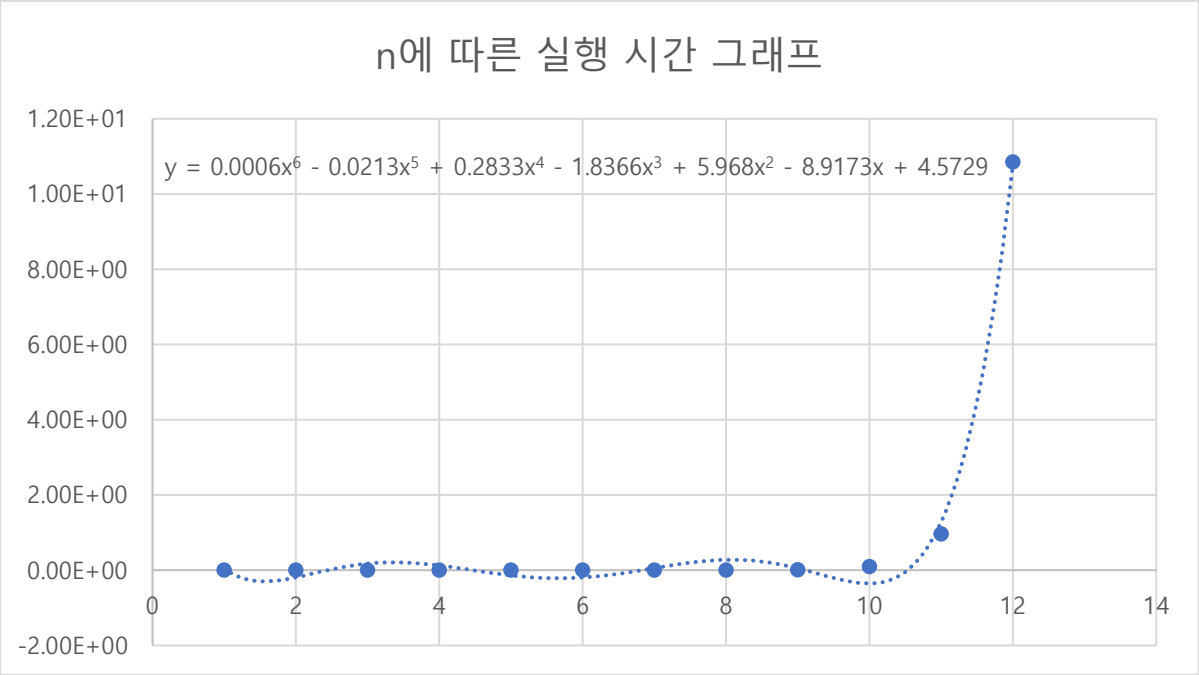
2. 시간 복잡도 계산 과정

perm 함수는 for-loop안에서 자기 자신을 재귀 호출하면서 for문의 반복횟수를 1씩 감소시킨다. 따라서 n개의 데이터에 대해 반복 횟수는 n * (n-1) * (n-2) * ... * 1 = n! 이므로 시간 복잡도 O(n!) 이다.

3. n의 변화에 따른 반복횟수 및 코드 실행시간 표

N	반복 횟수	실행시간
1	1502	6.66E-07
2	1128	8.87E-07
3	1001	1.04E-06
4	640	1.56E-06
5	251	3.98E-06
6	45	2.25E-05
7	10	0.0001106
8	1	0.00111
9	1	0.00834
10	1	0.092411
11	1	0.959238
12	1	10.8503

4. n의 변화에 따른 실행시간 변화 chart



5. 결론

n-permutation을 코드로 구현하고 실행 후 실행 시간을 측정한 결과 n이 1 증가함에 따라 실행 시간이 n배에 가깝게 증가함을 알 수 있었다. 또한 Excel 프로그램의 한계로 n!의 추세선을 그릴 수는 없었지만 6차 테일러 급수로 근사하여 그래프의 추세선을 그려본 결과 데이터들이 추세선과 유사한 양상을 보였고 이를 통하여 계산한 시간 복잡도의 타당성을 확인할 수 있었다.