<시간복잡도의 개념과 구하는법>

20221057 권관우

시간 복잡도(Time Complexity)는 알고리즘이 특정 크기의 입력을 처리하는 데 걸리는 시간의 상대적 증가율을 나타내는 지표라고 구글에서 알려준다.

대충 입력된 값을 받아. 처리할 때 알고리즘의 구조로 인해 걸리는 시간을 표현한 것이라고 이해했다.
시간 복잡도 표현 방식에는 여러 가지가 있지만
가장 최악의 경우를 보여주는 Big-O 표기법이 대표적이다.

```c
#include <stdio.h>
#include <stdlib.h>
typedef unsigned long ULONG;

ULONG factorial(int n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}
```

위의 팩토리얼 함수는.
한번의 재귀호출로 n이 0이 될 때까지 호출을 한다. n번의 호출이 있기어서
선형으로 진행되기에 O(N)
점화식 표현 T(n) = T(n-1)

```c
ULONG sum(int A[], int n) {
    if (n == 1) return A[0];
    return A[n - 1] + sum(A, n - 1);
}
```

이 함수 또한 배열의 마지막부터 호출하면서 배열의 맨 처음 인덱스에 도착하면 반환하면서 더하는 연산.
이 또한 선형 진행의 O(N)이며
T(n) = T(n-1)이다

```c
ULONG Fibonacci_Iteration(int n) {
    int i;
    ULONG Result;
    ULONG* FibonacciTable;
    if (n == 0 || n == 1) return (ULONG)n;
    FibonacciTable = (ULONG*)malloc(sizeof(ULONG) * (n + 1));
    FibonacciTable[0] = 0;
    FibonacciTable[1] = 1;
    for (i = 2; i <= n; i++)
        FibonacciTable[i] = FibonacciTable[i - 1] + FibonacciTable[i - 2];
    Result = FibonacciTable[n];
    free(FibonacciTable);
    return Result;
}
```

피보나치수열을 반복문으로 표현한 함수로서
배열을 동적 생성하고 채우는등의 O(1)의 연산이 있지만
이 또한 1회의 반복문으로 선형진행 O(N)이다.
T(n) = 그냥 O(n)이다 반.복.문

```c
ULONG Fibonacci_Recursion(int n) {
    ULONG Result;
    ULONG* FibonacciTable;
    if (n == 0 || n == 1) return (ULONG)n;
    FibonacciTable = (ULONG*)malloc(sizeof(ULONG) * (n + 1));
    FibonacciTable[0] = 0;
    FibonacciTable[1] = 1;
    if (n >= 2) {
        FibonacciTable[n] = Fibonacci_Recursion(n - 1) + Fibonacci_Recursion(n - 2);
    }
    Result = FibonacciTable[n];
    free(FibonacciTable);
    return Result;
}
```

피보나치 수열을 구하는 재귀함수이다.
동적 생성을 재귀함수 내부에서 하는 비효율적인 연산이 있긴 한데
T(n) = T(n-1) + T(n-2)의 함수로서 O(2^n)이다.
사실상 실제 연산을 따져보면 2^n은 절대 나오지 않지만 표현가능한 bigO 표기가 이게 제일 적합하다.

```c
void dividelist(int A[], int p, int r) {
    if (p >= r) {
        printf("%d %d\n", p, r);
    }
    else {
        int mid = (p + r) / 2;
        printf("%d %d %d\n", p, mid, r);
        dividelist(A, p, mid - 1);
        dividelist(A, mid + 1, r);
    }
}
```

리스트를 나누는 연산으로서.
T(n) = T(n/2)+T(n/2) 라서
T(n) = 2T(n/2)이고 이는 리프호출까지 나누면서 내려가지만 사실상 모든 인덱스를 거치기 때문에 O(n)이다.

```c
int binarysearch(int A[], int toFind, int start, int end) {
    int mid = start + (end - start) / 2;
    if (start > end) return -1;
    else if (A[mid] == toFind) return mid;
    else if (A[mid] > toFind) return binarysearch(A, toFind, start, mid - 1);
    else return binarysearch(A, toFind, mid + 1, end);
}
```

이진 탐색 트리로서 정렬된 배열에서 사용되는 탐색 방법인데.
조건보다 크거나 작으면 과감하게 버리고 조건에 맞는 부분만 탐색하는 코드이다.
때문에 재귀가 진행될수록 탐색 크기가 줄어든다.
T(n) = T(n/2)로서 O(log n)이다.

```c
int search(int A[], int toFind, int count) {
```

```
    return binarysearch(A, toFind, 0, count - 1);
}


void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod) {
    if (n == 1) {
        printf("\nMove disk 1 from rod %c to rod %c", from_rod, to_rod);
        return;
    }
    towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
    printf("\nMove disk %d from rod %c to rod %c", n, from_rod, to_rod);
    towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
}
```

호출 횟수로만 보면 T(n) = 2T(n-1)로서

T(n) = 2T(n-1)
 = 2(2T(n-2))
 = 4T(n-2)
 = 8T(n-3)

시간복잡도는 O(2^n)이 된다.

그러나 아직까지 하노이타워 알고리즘의 설계를 완벽히 이해하지는 못했다.

```
int main(void) {

    int A[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
    int N = 8;
    //Test 1: factorial recursion
    printf("Factorial(%d) = %lu\n", 4, factorial(4));
    printf("end of test\n");
    //Test 2: integersum_recursion
    printf("sum(1 .. %2d) = %lu\n", N, sum(A, 8));
    printf("end of test\n");
    //Test 3: fibonacci_recursion vs fibonacci_iteration
    printf("fib_재귀(%d) = %lu\n", 10, Fibonacci_Recursion(10));
    printf("fib_반복(%d) = %lu\n", 10, Fibonacci_Iteration(10));
    printf("end of test\n");
    //test 4: integerlist
    dividelist(A, 0, N - 1);
    printf("end of test\n");
    //test 5: binary search
    printf("search(%d) = %d\n", 7, search(A, 7, 8));
    printf("end of test\n");
    //test 6: towerofHanoi_recursion
    towerOfHanoi(4, 'A', 'C', 'B');
    printf("\nend of test\n");
    return 0;
}
```