ADT & Complexity

즐겁고 알찬 자료구조 튜터링

ADT

ADT

- Abstraction Data Types
- 구체적인 기능의 완성과정을 언급하지 않고, 순수하게 기능이 무엇인지를 나열한 것
- 객체의 명세(specification)와 구현(implementation)을 분리해 what과 how를 명확히 구분

Class Calulator

```cpp
class Calulator {
public:
    int add(int a, int b) {
        return a + b;
    }
    int sub(int a, int b) {
        return a - b;
    }
    int mul(int a, int b) {
        return a * b;
    }
    int div(int a, int b) {
        return a / b;
    }
};
```

```cpp
int main() {
    Calulator cal;
    int a = cal.add(3, 4);
    int s = cal.sub(3, 4);
    int m = cal.mul(3, 4);
    int d = cal.div(3, 4);
}
```

Calulator 클래스의 ADT

- int add(int a, int b);
  - 더하기 기능을 하는 함수
  - 두 개의 정수 인자를 더한 값을 반환
- int sub(int a, int b);
  - 빼기 기능을 하는 함수
  - 두 개의 정수 인자를 뺀 값을 반환
- int mul(int a, int b);
- int div(int a, int b);

differential 함수

```
int **differential(int **polynomial, int highest_term) {
    int **diff_polynomial = new int*[highest_term - 1];

    for (int i = 0; i < highest_term - 1; i++) {
        diff_polynomial[i] = new int[2];
        diff_polynomial[i][0] = polynomial[i][0] * polynomial[i][1];
        diff_polynomial[i][1] = polynomial[i][1] - 1;
    }

    return diff_polynomial;
}
```

differentrial 함수의 ADT

- int **differential(int **polynomial, int highest_term)
  - 다항식을 미분하는 함수
  - 인자로 다항식과 최고차항을 받아 다항식의 미분 결과를 반환

differentrial 함수의 ADT

- int **differential(int **polynomial, int highest_term)
  - 다항식을 미분하는 함수
  - 함수에 사용되는 함수는 계수와 차수를 저장하는 2차원 int 형 배열
  - ex) $3x^2 + 4x + 2$
    - int p[3][2]
    - p[0][0] = 3, p[0][1] = 2, p[1][0] = 4, p[1][1] = 1, p[2][0] = 2, p[2][1] = 0
  - polynomial : 미분할 다항식, highest_term : 최고차항

Complexity

Complexity

- 알고리즘을 평가하는 두 가지 요소
  - 시간 복잡도(time complexity) : 속도
  - 공간 복잡도(space complexity) : 메모리

- 복잡도의 세 가지 경우
  - 최선의 경우 : B(n)
  - 평균의 경우 : A(n)
  - 최악의 경우 : W(n)

시간 복잡도

- 시간 복잡도를 구하기 위해 필요한 사항
  - 단위연산(basic instruction) : 명령문이나 명령문의 군(덩어리)
  - 입력크기(input size) : 반복되는 횟수

- T(n) = n
  - 입력크기가 n이면 명령문의 실행 횟수가 n번이 됨

순차탐색의 시간 복잡도

```c
int linear_search(int *arr, int len, int target) {
    for (int i = 0; i < len; i++) {
        if (arr[i] == target) {
            return i;
        }
    }

    return -1;
}
```

순차탐색의 시간 복잡도

- 최선의 경우 분석
  - 단위연산 : 배열 원소와 target과의 비교
  - 입력크기 : 배열 원소의 개수 len
  - len ≥ 1이기에 반복문이 최소 한 번 실행되고, 배열의 첫 원소가 target이라면 len과 상관 없이 반복문이 실행되는 횟수는 1
  - $B(n) = 1$

순차탐색의 시간 복잡도

- 최악의 경우 분석
  - 단위연산 : 배열 원소와 target과의 비교
  - 입력크기 : 배열 원소의 개수 len
  - taget이 배열의 마지막 원소이거나 target이 배열에 없는 경우 단위연산이 len만큼 실행되고, 해당 경우가 단위연산이 가장 많이 발생하는 경우
  - $W(n) = n$

순차탐색의 시간 복잡도

- 평균의 경우 분석
  - 단위연산 : 배열 원소와 target과의 비교
  - 입력크기 : 배열 원소의 개수 len
  - target이 배열의 k번째에 위치할 확률은 1/n이면, target을 찾는데 단위연산의 실행 횟수는 k가 됨
  - A(n) $= \sum_{k=1}^{n}(k \times \frac{1}{n}) = \frac{1}{n} \times \frac{n(n+1)}{2} = \frac{n+1}{2}$

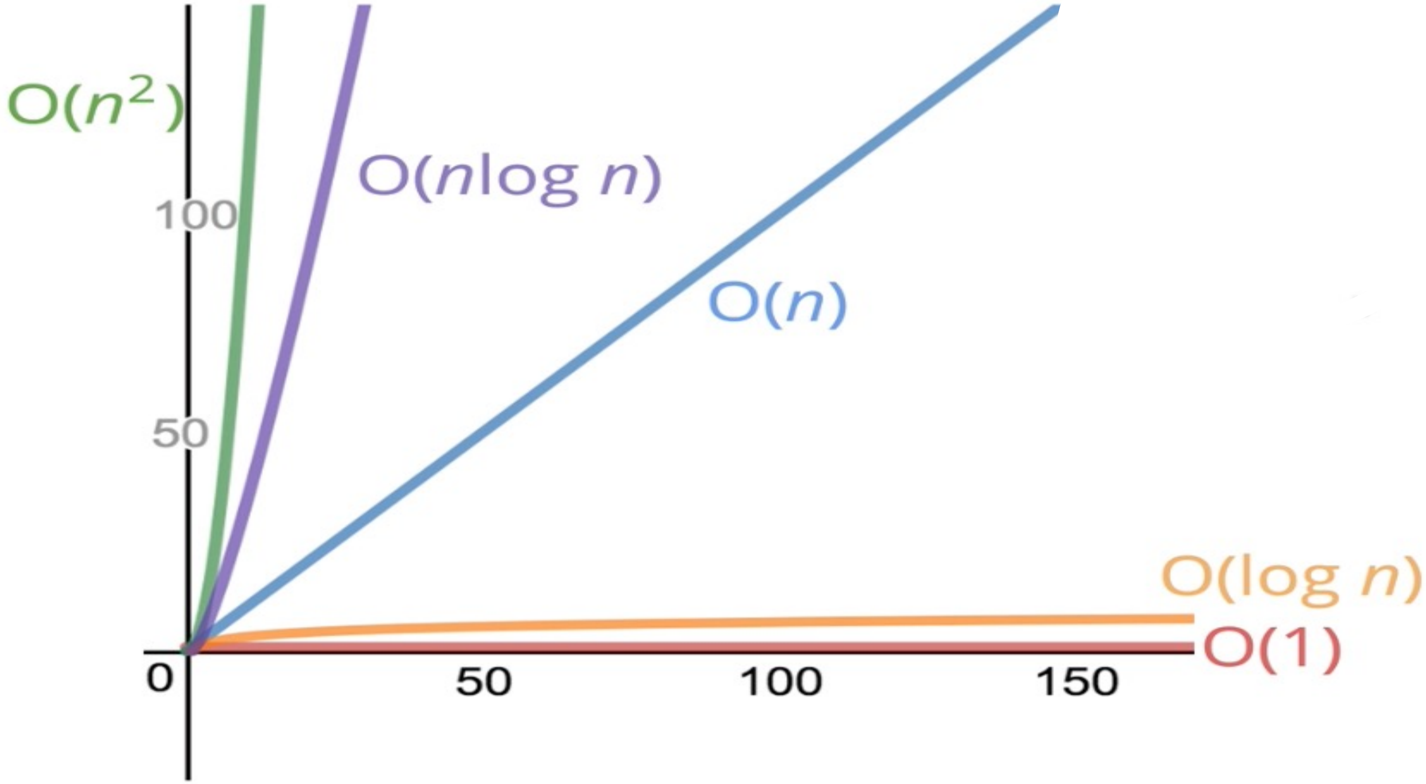
순차탐색의 시간 복잡도

- 평균의 경우 분석
  - target이 배열에 없는 경우 또한 계산 필요
  - target이 배열에 있을 확률이 p라면, target이 k번째에 있을 확률은 p/len, target이 배열에 없을 확률은 1 – p
  - target을 k번째 위치에서 찾을 경우 반복문을 k번 실행하고, target이 없는 경우 반복문을 len번 실행
  - A(n) $= \sum_{k=1}^{n}\left(k \times \frac{p}{n}\right) + n(1-p) = \frac{p}{n} \times \frac{n(n+1)}{2} + n(1-p) = n\left(1-\frac{p}{2}\right) + \frac{p}{2}$

시간 복잡도 사용

- 시간 복잡도를 계산할 때는 주로 최악의 경우를 사용
- 최악의 경우 분석은 어떠한 입력이 들어와도 해당 값을 초과하지 않아 보장된 성능을 보임

점근적 표기법(asymptomatic notation)



Big-Oh notation

- 주어진 복잡도 함수 $f(n)$에 대해서 $O\big(f(n)\big)$은 정수 N 이상의 모든 n에 대해서 다음 부등식이 성립하는 양의 실수 c와 음이 아닌 정수 N이 존재하는 복잡도 함수 $g(n)$의 집합
  - $g(n) \leq c \times f(n)$
- 함수의 궁극적인 상태만 고려하기 때문에 함수의 점근적인 (asymptotic) 상태를 나타내어, big O는 함수의 점근적인 상한(asymptotic upper bound)을 정함

Big-Omega notation

- 주어진 복잡도 함수 $f(n)$에 대해서 $\Omega(f(n))$은 N 이상의 모든 n에 대해서 다음 부등식을 만족하는 양의 실수 c와 음이 아닌 정수 N이 존재하는 복잡도 함수 $g(n)$의 집합
  - $g(n) \geq c \times f(n)$
- 복잡도 함수의 점근적인 하한(asymptotic lower bound)을 나타냄

Big-Theta notation

- 주어진 복잡도 함수 $f(n)$에 대해서 다음과 같이 정의
  - $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$
- $\Theta\big(f(n)\big)$는 N 이상의 모든 정수 n에 대해서 다음 부등식이 만족하는 양의 실수 c, d와 음이 아닌 정수 N이 존재하는 복잡도 함수 $g(n)$의 집합
  - $c \times f(n)) \leq g(n) \leq d \times f(n)$

점근적 표기법 정리

- $O(f(n)) = g(n)$이라면, $g(n)$은 $f(n)$보다 점근적으로 작다.

- $\Omega(f(n)) = g(n)$이라면, $g(n)$은 $f(n)$보다 점근적으로 크다.

- $\Theta(f(n)) = g(n)$이라면, $g(n)$은 $f(n)$과 점근적으로 같다.