자료구조와 알고리즘 이해

1주차-강의

남춘성

SUNG KYUN KWAN UNIVERSITY

자료구조의 정의- I

SUNG KYUN KWAN UNIVERSITY

- C언어에서 프로그래밍
  - 프로그램은 데이터를 표현하고, 표현된 데이터를 처리 하는 작업
    - 프로그램 표현 : 자료구조 -> 데이터의 표현 및 저장방법
    - 데이터 처리 : 알고리즘 -> 문제의 해결 방법

자료구조의 정의 - II

SUNG KYUN KWAN UNIVERSITY

- 자료구조의 예

자료구조

```
// 배열의 선언
int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

- 알고리즘의 예

```
// 배열에 저장된 값의 합
for(idx=0; idx<10; idx++)
    sum += arr[idx];
```

알고리즘

→ 알고리즘은 자료구조에 의존적이다

알고리즘의 성능분석 방법 - 시간 및 공간 복잡도 I

SUNG KYUN KWAN UNIVERSITY

- 알고리즘 평가를 위한 2가지 요소
  - 시간 복잡도(Time complexity) : 속도에 해당하는 수행시간 분석 결과
  - 공간 복잡도(Space complexity) : 메모리 사용량에 대한 분석 결과

  → 시간 복잡도를 더 중요시 함!

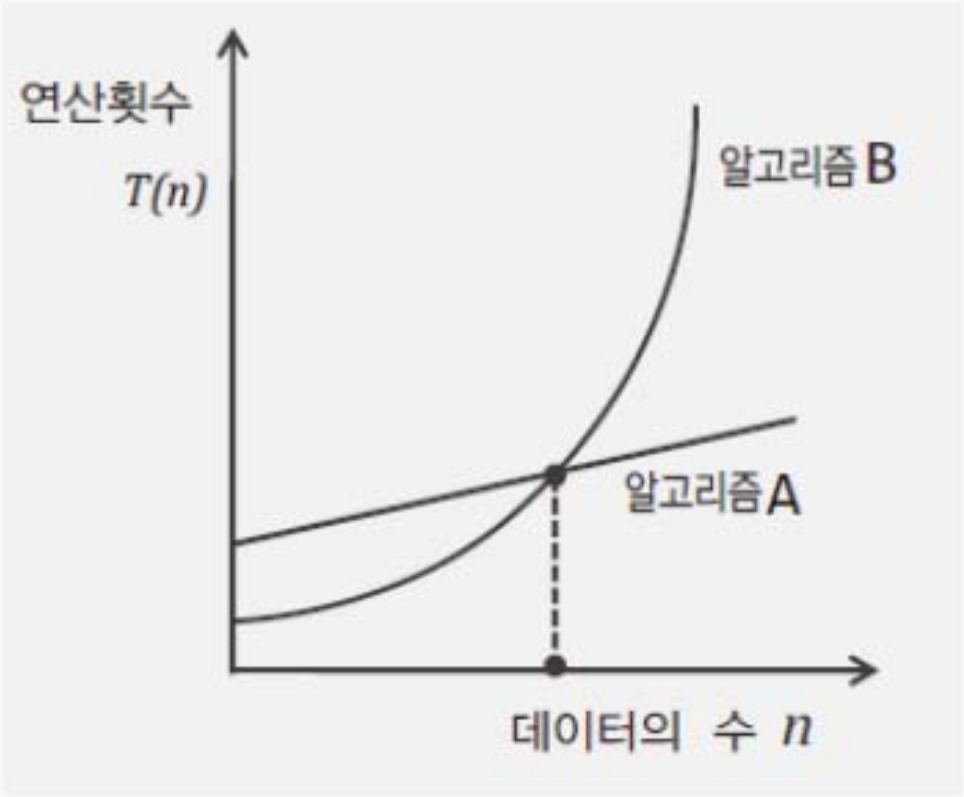
- 알고리즘 수행 속도 평가 : 시간 복잡도
  - 연산의 횟수를 세어 평가
  - 처리할 데이터의 수 n에 대한 연산횟수 함수 T(n)을 구함

  → 식을 구성하면 데이터 수 증가에 따른 연산 횟수 변화 정도를 판단할 수 있음

알고리즘의 성능분석 방법 - 시간 및 공간 복잡도 II

SUNG KYUN KWAN UNIVERSITY

- 알고리즘 수행 속도 비교
  - 데이터 수가 적으면 B를 적용하고, 많으면 A를 적용
  - 즉, 상황에 맞는 답을 내려야 함



알고리즘의 성능분석 방법 - 순차 탐색 알고리즘 I

SUNG KYUN KWAN UNIVERSITY

- 순차탐색 알고리즘 시간 복잡도
  - 시간복잡도: T(n)
    - 데이터의 수 n에 대한 연산횟수 함수 T(n)
  - 코드에서
    - <, ++, == 이 얼마나 연산을 수행했는가?
    - arr = { 3, 5, 2, 4, 9}
    - LSearch(… , …, 4)와 Lsearch(…, …, 7) 인 경우를 통해 비교

```
int LSearch(int arr[], int len, int target) {

        int i;

        for (i = 0; i < len; i++) {
            if (arr[i] == target)
                return i;
        }

        return -1;
}
```

알고리즘의 성능분석 방법 - 순차 탐색 알고리즘 II

SUNG KYUN KWAN UNIVERSITY

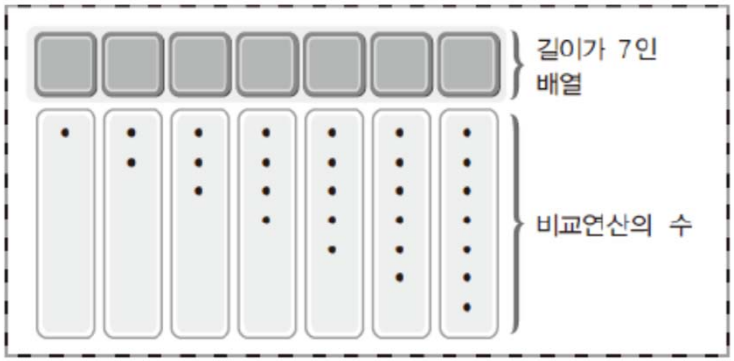
- 순차 탐색 상황
  - 운이 좋은 경우
    - 배열의 맨 앞에서 대상을 찾는 경우 : Best case
  - 운이 좋지 않은 경우
    - 배열의 끝에서 찾거나 저장되어 있지 않은 경우 : Worst case
      - 최악의 경우는 늘 동일하게 발생할 수 있음
  - 평균적인 경우
    - 일반적인 상황에 대한 경우의 수 : Average case
      - '평균적인 경우' 임을 증명하기 어렵고, 상황에 따라 달라짐.

알고리즘의 성능분석 방법 - 순차 탐색 알고리즘 III

SUNG KYUN KWAN UNIVERSITY

- 순차 탐색 시 최악의 경우 시간 복잡도
  - 정의 : 데이터 수가 n개 일 때, 최악의 경우에 해당하는 연산 횟수는 n이다.
  - T(n) = n : 최악의 경우를 정의한 함수로 구현 가능

- 순차 탐색 시 평균적인 경우 시간 복잡도
  - 가정 : 탐색대상이 존재하지 않을 확률 - 50%
  - 가정 : 배열 첫 요소부터 마지막 요소까지 탐색 대상 존재 확률 동일
    - 탐색 대상이 존재하지 않는 경우 연산 횟수 : n
    - 탐색 대상이 존재하는 경우 연산 횟수 : n/2



존재하는 경우 ½, 존재하지 않는 경우 ½

T(n) = n * (1/2) + (n/2)*(1/2) = (3/4)n

알고리즘의 성능분석 방법 - 이진 탐색 알고리즘 I

SUNG KYUN KWAN UNIVERSITY

- 이진 탐색 알고리즘 방법
  - 배열 인덱스 시작과 끝을 합하여(n) 결과를 2로 나눔
  - 얻은 결과 n/2를 인덱스 값으로 하여 arr[n/2]에 저장된 값 확인
  - arr[n/2]에 저장된 값과 탐색 대상 값 대소를 비교
  - 결과에 따라 인덱스 범위를 제한
    - 탐색 대상이 크면 (n/2)+1 ~ n-1, 작으면 0 ~ (n/2)-1.
  - 첫 번째 문장부터 반복.

First와 last가 만나면 탐색 대상이 1개만 남음
First와 last가 역전된다면, 상황은 종료

```
while(first<=last) {
        //이진 탐색 알고리즘 진행
}
```

알고리즘의 성능분석 방법 - 이진 탐색 알고리즘 II

SUNG KYUN KWAN UNIVERSITY

- 이진 탐색 알고리즘 시 최악의 경우 시간 복잡도
  - n이 1이 되기까지 2로 나눈 횟수 k회, 비교연산 k회 진행
  - 데이터가 1개 남았을 때, 이때 마지막 비교연산 1회 진행
  - T(n) = k+1

  - K 구하기
    - n이 1이 되기까지 2로 나눈 횟수가 k : n * (1/2)^k = 1
    - n * (1/2)^k = 1 → n*2^(-k) = 1 → n = 2^k → $\log_2 n = \log_2 2^k$ → $\log_2 n = k \log_2 2$ → $\log_2 n = k$
    - k = $\log_2 n$

  - T(n) 구하기
    - T(n) = $\log_2 n+1$
      - 시간 복잡도의 목적은 n의 값에 따른 T(n)의 증가 및 감소 정도를 판단하는 것이라 +1 생략 가능
    - → T(n) = $\log_2 n$

알고리즘의 성능분석 방법 - 빅-오 표기법 I

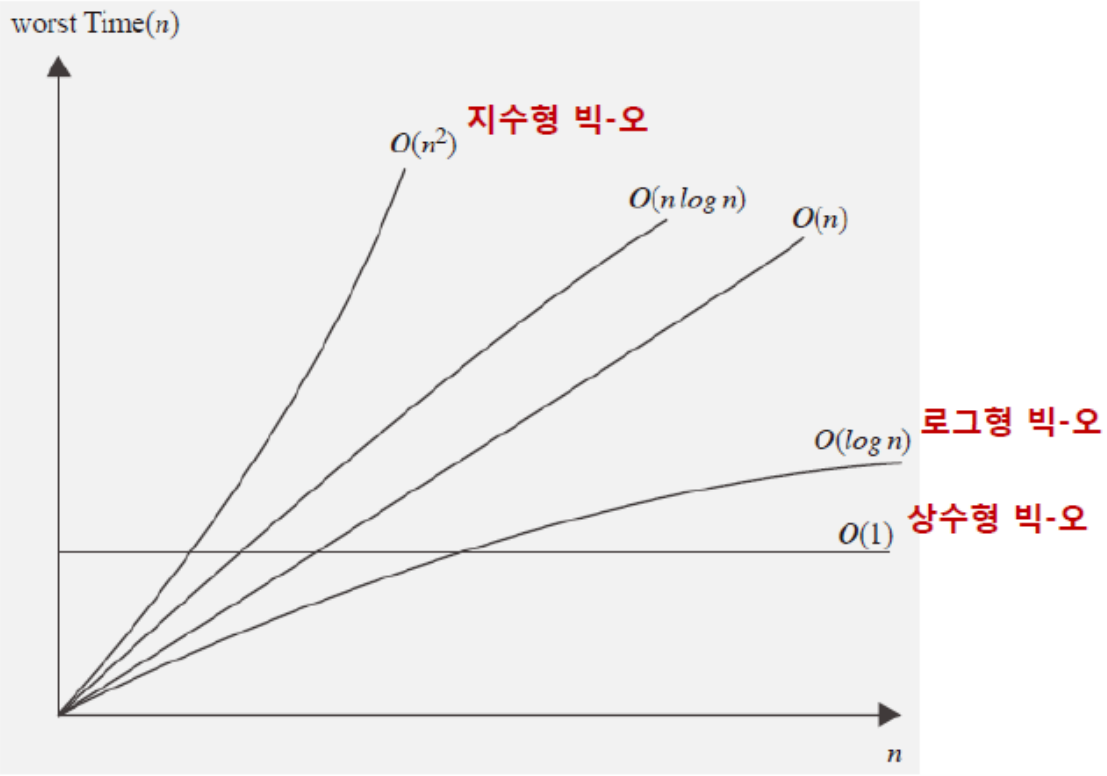
SUNG KYUN KWAN UNIVERSITY

- 빅오(Big-Oh Notation)
  - T(n)에서 실제로 영향력을 끼치는 부분을 가리켜 빅-오(Big-Oh)라 함.
  - N의 증감함에 따라 상수값이 차지하는 비율은 미미해짐.

n	n^2	$2n$	$T(n)$	n^2의 비율
10	100	20	120	83.33%
100	10,000	200	10,200	98.04%
1,000	1,000,000	2,000	1,002,000	99.80%
10,000	100,000,000	20,000	100,020,000	99.98%
100,000	10,000,000,000	200,000	10,000,200,000	99.99%

  - $T(n) = n^2 + 2n + 1 \rightarrow T(n) = n^2 + 2n \rightarrow T(n) = n^2 \rightarrow O(n^2)$
  - 결국 : T(n)이 다항식으로 표현된 경우, 최고차항의 차수가 빅-오가 됨.
  - $\rightarrow T(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n^1 + a_0 \rightarrow O(n^m)$

알고리즘의 성능분석 방법 - 빅-오 표기법 II



$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$$

알고리즘의 성능분석 방법 - 빅-오 표기법 III

SUNG KYUN KWAN UNIVERSITY

- 순차 탐색 알고리즘 vs 이진 탐색 알고리즘
  - 순차 $T(n) = n$ → $O(n)$
  - 이진 $T(n) = \log_2 n+1$ → $O(\log_2 n)$

n	순차 탐색 연산횟수	이진 탐색 연산횟수
500	500	9
5,000	5,000	13
50,000	50,000	16

알고리즘의 성능분석 방법 - 빅-오 표기법 IV

SUNG KYUN KWAN UNIVERSITY

- 빅-오 수학적 정의
  - 두 개의 함수 f(n)과 g(n)이 주어졌을 때, 모든 n>=k에 대하여 f(n)<=Cg(n)을 만족하는 두 개의 상수 C와 K가 존재하면, f(n)의 빅-오는 O(g(n))이다.
  - 결국, f(n)이 연산횟수의 증가율이 크다고 해도 증가율 패턴은 g(n)을 넘지 못함.
- 예)
  - 두 개의 함수f(n)과 g(n)이 주어졌을 때
    - 두 개의 함수 $f(n) = 5n^2+100$, $g(n) = n^2$이 주어졌을 때
  - 모든 n>= k에 대하여
    - 모든 n>= 12에 대하여
    - (즉, n의 값이 점차 증가하여 어느 순간 이후부터)
  - f(n)<=Cg(n)을 만족하는 두 개의 상수 C와 K가 존재하면
    - $5n^2+100 <= 3500n^2$ 을 만족하는 3500(C)와 12(K)가 존재하면,
  - f(n)의 빅오는 O(g(n))이다.
    - $5n^2+100$의 빅-오는 $O(n^2)$이다.

알고리즘의 성능분석 방법 - 빅-오 표기법 IV (점근법)

SUNG KYUN KWAN UNIVERSITY

- *O-표기법 : O(g(n))*
  - 기껏해야 $g(n)$의 비율로 증가하는 함수
  - e.g., $O(n)$, $O(n\log n)$, $O(n^2)$, $O(2^n)$, …

- Formal definition
  - $O(g(n)) = \{ f(n) \mid \exists c > 0, n_0 \geq 0 \text{ s.t.} \forall n \geq n_0, cg(n) \geq f(n) \}$
  - $= \{ f(n) \mid$ 모든 $n \geq n_0$ 에 대하여 or 충분히 큰 모든 n에 대하여
  - $cg(n) \geq f(n)$ 인 양의 상수 c와 n_0 가 존재한다$\}$
  - $f(n) \in O(g(n))$을 관행적으로 $f(n) = O(g(n))$이라고 쓴다.

∃ : 존재한다 (exist)
∈ : 원소이다
∀ : 임의의
s.t : such that

- 직관적 의미
  - $f(n) = O(g(n)) \Rightarrow f$는 g보다 빠르게 증가하지 않는다
  - 상수 비율의 차이는 무시