

윤성우의 열혈 자료구조

: C언어를 이용한 자료구조 학습서



Chapter 01. 자료구조와 알고리즘의 이해

Introduction To Data Structures Using C

Chapter 01. 자료구조와 알고리즘의 이해



Chapter 01-1:

자료구조에 대한 기본적인 이해

교재에서 C언어와 관련하여 알고 있다고 가정하는 부분

- ▶ 구조체를 정의할 줄 안다.
- ▶ 메모리의 동적 할당과 관련하여 이해한다.
- ▶ 포인터와 관련하여 이해와 활용에 부담이 없다.
- ▶ 헤더파일의 정의하고 활용할 줄 안다.
- ▶ 각종 매크로를 이해하고 `#ifndef ~ #endif`의 의미를 안다.
- ▶ 다수의 소스파일, 헤더파일로 구성된 프로그램 작성 가능!
- ▶ 재귀함수에 어느 정도 익숙하다.

➔ 본 강의에서는 관련 내용이 나올 때마다 보충 설명을 하도록 하겠습니다.

자료구조란 무엇인가?

자료구조

“프로그램이란 데이터를 표현 하고,

표현에는 저장의 의미가 포함된다!

그렇게 표현된 데이터를 처리 하는 것이다.”

알고리즘



자료구조의 분류

교재에서 자료구조를 설명하는 방향

- ▶ 자료구조 학습 방법의 일반적인 구분
 - 자료구조의 모델 자체에 대한 이해 중심 학습
 - 코드 레벨에서의 자료구조 구현 중심 학습
- ▶ 자료구조 학습에 있어서 기억할 것 두 가지
 - 자료구조의 모델을 그림으로 우선 이해해야 한다.
 - 코드 레벨로 구현이 가능해야만 의미가 있는 것은 아니다.

자료구조와 알고리즘

```
int main(void)
```

```
{
```

```
// 배열의 선언
```

자료구조

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

```
...
```

```
// 배열에 저장된 값의 합
```

```
for(idx=0; idx<10; idx++)
```

```
    sum += arr[idx];
```

알고리즘

```
...
```

```
}
```

알고리즘은 자료구조에 의존적이다!

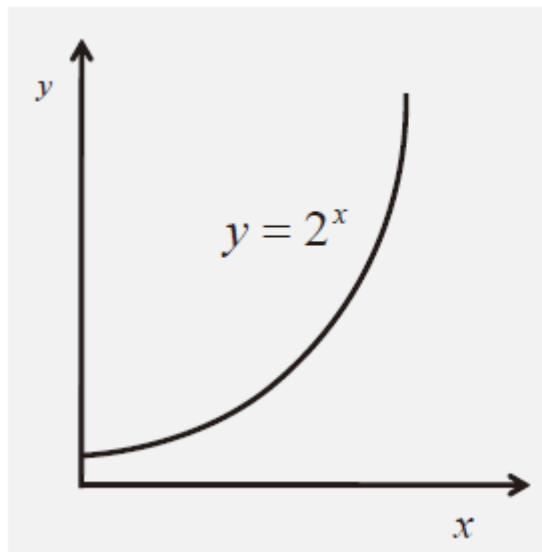
Chapter 01. 자료구조와 알고리즘의 이해



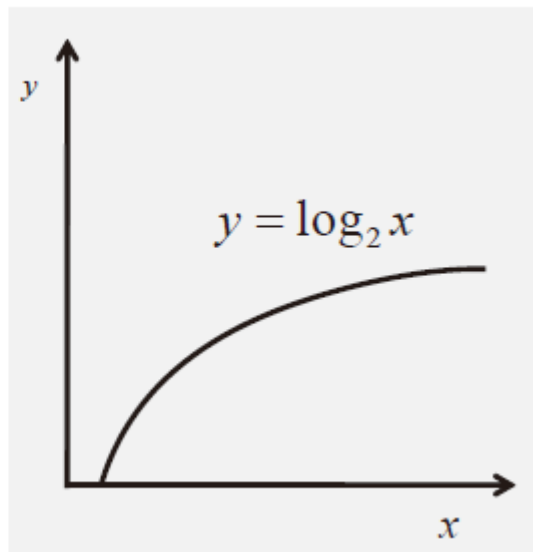
Chapter 01-2:

알고리즘의 성능분석 방법

교재에서 수학과 관련해서 알고 있다고 가정하는 것



지수식



로그식

X 축이 데이터의 수! Y 축이 연산의 횟수를 의미한다면?

시간 복잡도 & 공간 복잡도 (1/2)

▶ 알고리즘을 평가하는 두 가지 요소

- 시간 복잡도(time complexity) → 얼마나 빠른가?
- 공간 복잡도(space complexity) → 얼마나 메모리를 적게 쓰는가?
- 일반적으로 시간 복잡도를 더 중요시 한다.

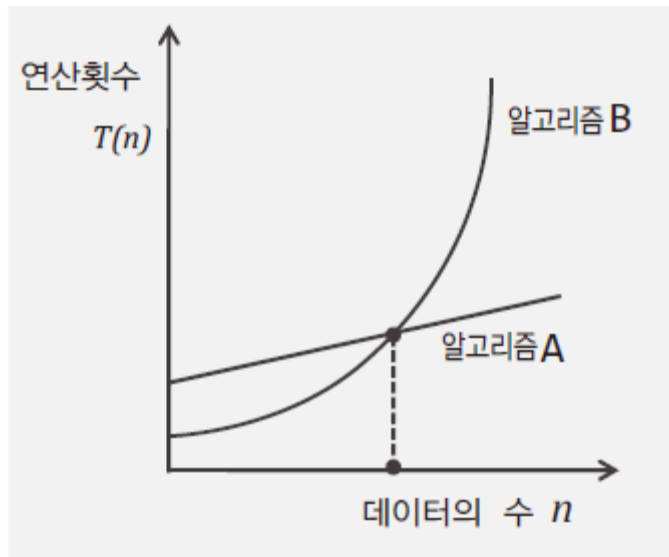
▶ 시간 복잡도의 평가 방법

- **중심이 되는 특정 연산**의 횟수를 세어서 평가를 한다.
- 데이터의 수(n)에 대한 연산횟수의 함수 $T(n)$ 을 구한다.

시간 복잡도 & 공간 복잡도 (2/2)

▶ 알고리즘의 수행 속도 비교 기준

- 데이터의 수(n)가 적은 경우의 수행 속도는 큰 의미가 없다.
- 데이터의 수(n)에 따른 수행 속도의 변화 정도를 비교 기준으로 한다.



두 알고리즘을 비교 평가해보면?

순차 탐색 알고리즘과 시간 복잡도

// 순차 탐색 알고리즘 적용된 함수
int LSearch(int ar[], int len, int target)

{

int i;

for(i=0; i<len; i++)

{

if(ar[i] == target)

return i; // 찾은 대상의 인덱스 값 반환

}

return -1; // 찾지 못했음을 의미하는 값 반환

}

어떠한 연산자의 연산 횟수를
세어야 하겠는가?

최악의 경우와 최상의 경우

▶ 순차 탐색에서 **운이 가장 좋은 경우(최상의 경우)**

- 배열의 맨 앞에서 대상을 찾았을 때
- 만족스러운 상황은 성능평가의 주 관심이 아니다!
- ‘**최상의 경우(best case)**’라 한다.

시간 복잡도와 공간 복잡도 각각에 대해서 최악의 경우와 최상의 경우를 구할 수 있다.

▶ 순차 탐색에서 **운이 가장 좋지 않은 경우(최악의 경우)**

- 배열에 대상이 저장되어 있지 않은 경우
- 만족스럽지 못한 상황으로 성능평가의 주 관심이다!
- ‘**최악의 경우(worst case)**’라 한다.

평균적인 경우(Average Case)

- ▶ 가장 현실적인 경우에 해당한다.
 - 일반적으로 등장하는 상황에 대한 경우의 수이다.
 - 최상의 경우와 달리 알고리즘 평가에 도움이 된다.
 - 하지만 **계산하기가 어렵다. 객관적 평가가 어렵다.** (단점)
- ▶ 평균적인 경우의 복잡도 계산이 어려운 이유
 - ‘평균적인 경우’의 **연출이 어렵다.**
 - ‘평균적인 경우’임을 **증명하기 어렵다.**
 - 따라서 ‘평균적인 경우’는 상황에 따라 달라진다.

반면 최악의 경우는 늘 동일하다.

순차 탐색 최악의 경우 시간 복잡도

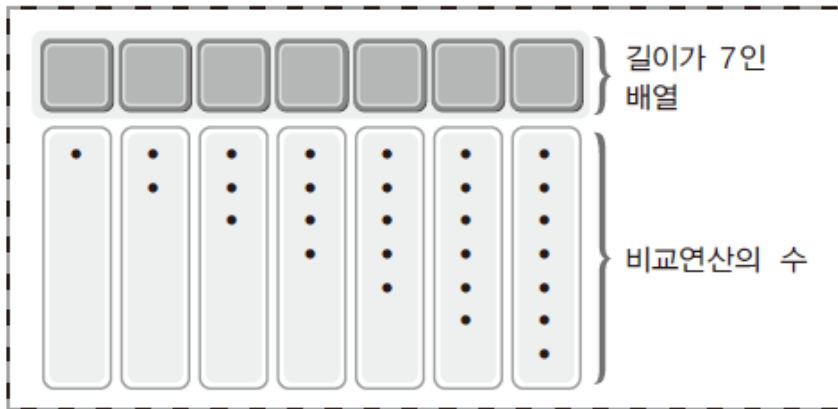
“데이터의 수가 n 개일 때,

최악의 경우에 해당하는 연산횟수는(비교연산의 횟수는) n 이다.”

$T(n) = n$ 최악의 경우를 대상으로 정의한 함수 $T(n)$

순차 탐색 평균적 경우 시간 복잡도

- 가정 1 탐색 대상이 배열에 존재하지 않을 확률 50%
- 가정 2 배열 첫 요소부터 마지막 요소까지 탐색 대상 존재 확률 동일!
- 탐색 대상이 존재하지 않는 경우의 연산횟수는 n
- 가정 2에 의해서 탐색 대상이 존재하는 경우의 연산횟수는 $(n+1)/2 \rightarrow n/2$



$$T(n) = n \times \frac{1}{2} + \frac{n}{2} \times \frac{1}{2} = \frac{3}{4}n$$

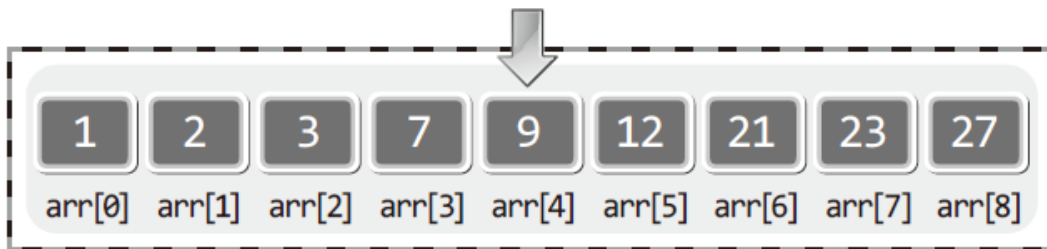
결론!

가정 1과 2는 평균적인 경우라 할 수 있겠는가? 그렇다면 무엇이 평균적인 경우이겠는가?

이진 탐색 알고리즘의 소개 1

👉 이진 탐색 알고리즘의 첫 번째 시도:

1. 배열 인덱스의 시작과 끝은 각각 0과 8이다.
2. 0과 8을 합하여 그 결과를 2로 나눈다.
3. 2로 나눠서 얻은 결과 4를 인덱스 값으로 하여 `arr[4]`에 저장된 값이 3인지 확인!



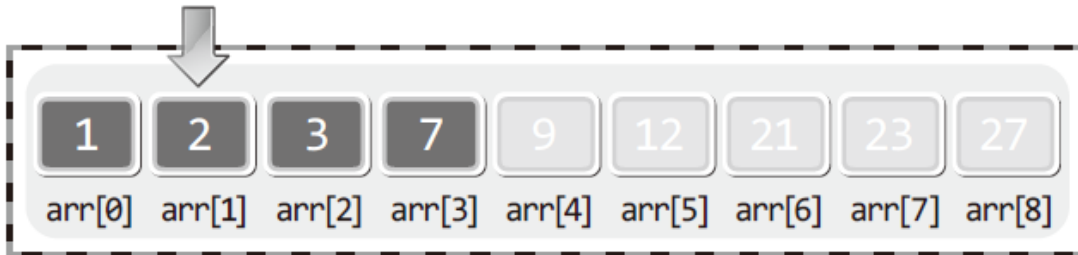
3이 저장되어 있는지를
탐색한다!

이진 탐색은 순차 탐색보다 훨씬 좋은 성능을 보이지만, **배열이 정렬되어 있어야 한다**는 제약이 따른다.

이진 탐색 알고리즘의 소개 2

👉 이진 탐색 알고리즘의 두 번째 시도:

1. `arr[4]`에 저장된 값 9와 탐색 대상인 3의 대소를 비교한다.
2. **대소 비교결과는 `arr[4] > 3`이므로 탐색 범위를 인덱스 기준 0~3으로 제한!**
3. 0과 3을 더하여 그 결과를 2로 나눈다. 이때 나머지는 버린다.
4. 2로 나눠서 얻은 결과가 1이니 `arr[1]`에 저장된 값이 3인지 확인한다.

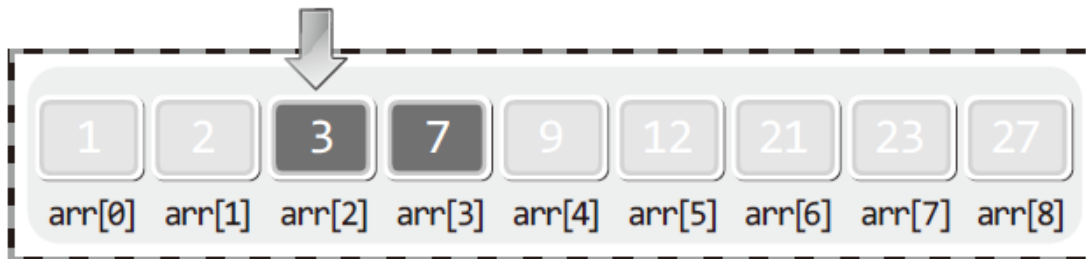


탐색의 대상이 첫 번째 시도 이후 반으로 줄었다!

이진 탐색 알고리즘의 소개 3

👉 이진 탐색 알고리즘의 세 번째 시도:

1. `arr[1]`에 저장된 값 2와 탐색 대상인 3의 대소를 비교한다.
2. **대소 비교결과 `arr[1] < 3`이므로 탐색의 범위를 인덱스 기준 2~3으로 제한!**
3. 2와 3을 더하여 그 결과를 2로 나눈다. 이때 나머지는 버린다.
4. 2로 나눠서 얻은 결과가 2이니 `arr[2]`에 저장된 값이 3인지 확인한다.



탐색의 대상이 다시 반으로 줄었다!

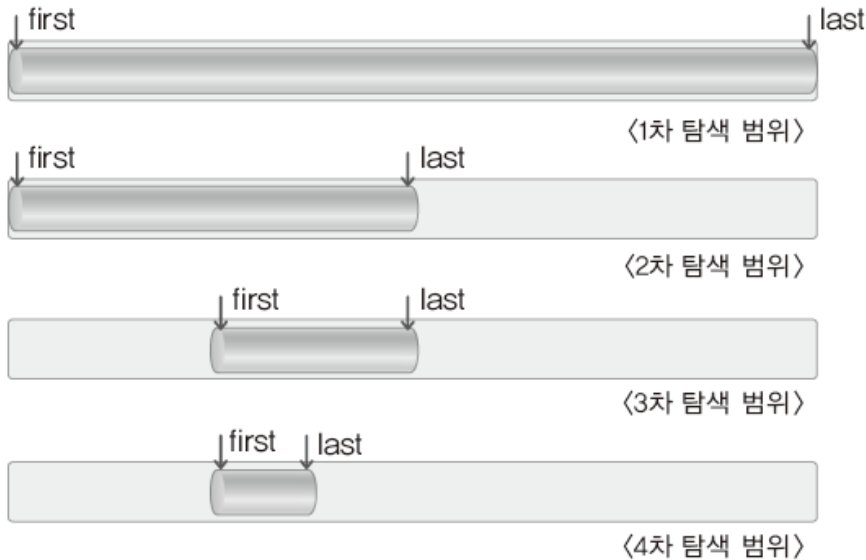
이진 탐색 알고리즘의 소개 4



이진 탐색 알고리즘의 핵심!

이진 탐색의 매 과정마다 탐색의 대상을 절반씩 줄여 나가기 때문에, 순차 탐색보다 좋은 성능을 보인다.

이진 탐색 알고리즘의 구현 1



- **first와 last가 만났다는 것은**
탐색 대상이 하나 남았다는 것을 뜻함!
- 따라서 **first와 last가 역전될때까지**
탐색의 과정을 계속!

이진 탐색의 기본 골격

```
while(first <= last)    // first <= last 가 반복의 조건임에 주의!  
{  
    // 이진 탐색 알고리즘의 진행  
}
```

이진 탐색 알고리즘의 구현 2

```
int BSearch(int ar[], int len, int target)
{
    int first = 0;           // 탐색 대상의 시작 인덱스 값
    int last = len-1;        // 탐색 대상의 마지막 인덱스 값
    int mid;

    while(first <= last)
    {
        mid = (first+last) / 2; // 탐색 대상의 중앙을 찾는다.
        if(target == ar[mid]) // 중앙에 저장된 것이 타겟이라면
        {
            return mid;       // 탐색 완료!
        }
        else // 타겟이 아니라면 탐색 대상을 반으로 줄인다.
        {
            if(target < ar[mid])
                last = mid-1;   // 왜 -1을 하였을까?
            else
                first = mid+1;   // 왜 +1을 하였을까?
        }
    }
    return -1; // 찾지 못했을 때 반환되는 값 -1
}
```

이진 탐색 알고리즘 최악의 경우 시간 복잡도1

```
while(first <= last)
{
    mid = (first+last) / 2;
    if(target == ar[mid])
    {
        return mid;
    }
    else    // 타겟이 아니면
    {
        ....
    }
}
```

시간 복잡도 계산을 위한 핵심 연산은?

== 연산자!

따라서 == 연산자의 연산 횟수를 기준으로
시간 복잡도를 결정할 수 있다.

데이터의 수가 n개일 때, 최악의 경우에 발생하는 비교연산의 횟수는?

이진 탐색 알고리즘 최악의 경우 시간 복잡도2

☞ 데이터의 수가 n 개일 때 비교 연산의 횟수

- 처음에 데이터 수가 n 개일 때의 탐색과정에서 1회의 비교연산 진행
- 데이터 수를 반으로 줄여서 그 수가 $n/2$ 개일 때의 탐색과정에서 1회 비교연산 진행
- 데이터 수를 반으로 줄여서 그 수가 $n/4$ 개일 때의 탐색과정에서 1회 비교연산 진행
- 데이터 수를 반으로 줄여서 그 수가 $n/8$ 개일 때의 탐색과정에서 1회 비교연산 진행

..... n 이 얼마인지 결정되지 않았으니

이 사이에 도대체 몇 번의 비교연산이 진행되는지 알 수가 없다!.....

- 데이터 수를 반으로 줄여서 그 수가 1개일 때의 탐색과정에서 1회의 비교연산 진행

이렇게 말로 표현하는 것은 객관적 성능 비교가 불가능하다!

이진 탐색 알고리즘 최악의 경우 시간 복잡도3

👉 비교 연산의 횟수의 예

- 8이 1이 되기까지 2로 나눈 횟수 3회, 따라서 **비교연산 3회** 진행
- 데이터가 1개 남았을 때, 이때 마지막으로 **비교연산 1회** 진행

👉 비교 연산의 횟수의 일반화

- n 이 1이 되기까지 2로 나눈 횟수 k 회, 따라서 **비교연산 k 회** 진행
- 데이터가 1개 남았을 때, 이때 마지막으로 **비교연산 1회** 진행

👉 비교 연산의 횟수의 1차 결론!

- 최악의 경우에 대한 시간 복잡도 함수 $T(n)=k+1$ 이제 k 만 구하면 된다!

이진 탐색 알고리즘 최악의 경우 시간 복잡도4

$$n \times \left(\frac{1}{2}\right)^k = 1$$

n을 몇번 반으로 나눠야 1이 되는가?
이 때 k가 나눠야 하는 횟수를 말한다.

👉 k에 관한 식으로...

$$n \times \left(\frac{1}{2}\right)^k = 1 \quad \blacktriangleright \quad n \times 2^{-k} = 1 \quad \blacktriangleright \quad n = 2^k$$

$$n = 2^k \quad \blacktriangleright \quad \log_2 n = \log_2 2^k \quad \blacktriangleright \quad \log_2 n = k \log_2 2 \quad \blacktriangleright \quad \log_2 n = k$$

이진 탐색 알고리즘 최악의 경우 시간 복잡도5

👉 이제 결론

- 함수 $T(n) = k+1$ 이므로...
- $T(n) = \log_2 n + 1$

👉 그러나 +1은 생략하여

- $T(n) = \log_2 n$

시간 복잡도의 목적은 n 의 값에 따른 $T(n)$ 의 증가 및 감소의 정도를 판단하는 것이므로 +1은 생략 가능!

그렇다면 +1이 아닌 +200 인 경우도 생략이 가능한가?
빅-오에 대한 이해를 통해서 이에 대한 답을 얻자!

빅-오 표기법(Big-O Notation)

$$T(n) = n^2 + 2n + 1$$

↓ 1차 근사치 식

$$T(n) = n^2 + 2n$$

↓ 2차 근사치 식

$$T(n) = n^2$$

↓ T(n)의 빅-오

$$O(n^2)$$

빅-오의 표기 방법

$T(n)$ 에서 실제로 영향력을 끼치는 부분을 가리켜 빅-오(Big-O)라 한다.

☞ n 의 변화에 따른 $T(n)$ 의 변화 정도를 판단하는 것이 목적이니 $+1$ 은 무시할 수 있다.

n	n^2	$2n$	$T(n)$	n^2 의 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%

☞ $2n$ 도 근사치 식의 구성에서 제외시킬수 있음을 보인 결과!
 n 이 증가함에 따라 $2n$ 이 차지하는 비율은 미미해진다.

단순하게 빅-오 구하기

👉 빅-오는?

- $T(n)$ 이 다항식으로 표현이 된 경우, **최고차항의 차수**가 **빅-오**가 된다.

👉 빅-오 결정의 예

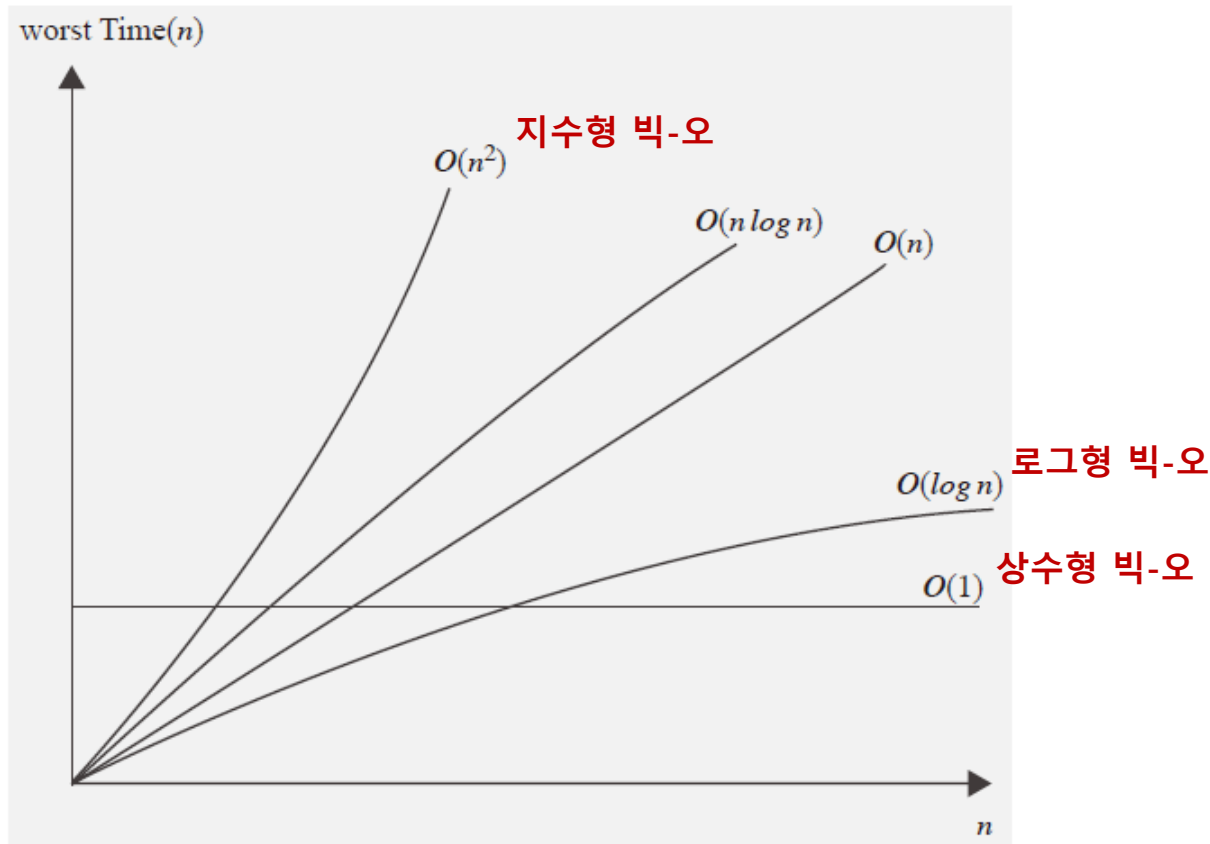
최고차항의 차수를 빅-오로 결정!

- $T(n) = \underline{n^2} + 2n + 9$ ▶ $O(n^2)$
- $T(n) = \underline{n^4} + n^3 + n^2 + 1$ ▶ $O(n^4)$
- $T(n) = \underline{5n^3} + 3n^2 + 2n + 1$ ▶ $O(n^3)$

👉 빅-오 결정의 일반화

- $T(n) = \underline{a_m n^m} + a_{m-1} n^{m-1} + \dots + a_1 n^1 + a_0$ ▶ $O(n^m)$

대표적인 빅-오



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

순차 탐색 알고리즘 vs. 이진 탐색 알고리즘

$$\bullet T(n) = n$$

순차 탐색의 최악의 경우 시간 복잡도



$$O(n)$$

순차 탐색의 빅-오

$$\bullet T(n) = \log_2 n + 1$$

이진 탐색의 최악의 경우 시간 복잡도



$$O(\log n)$$

이진 탐색의 빅-오

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

이진 탐색의 연산횟수는
예제 `BSWorstOpCount.c`에서
확인한 결과

최악의 경우에 진행이 되는 연산의 횟수



연습문제1

- ▶ 알고리즘의 평가에 대한 설명으로 **잘못된** 것은?
 - ① 시간 복잡도는 알고리즘이 얼마나 빠른가를 나타낸다.
 - ② 데이터 수(n)에 대한 연산 횟수의 함수는 $T(n)$ 이다.
 - ③ 시간 복잡도는 모든 연산의 횟수를 세어서 평가한다.
 - ④ 데이터 수(n)이 적은 경우의 수행 속도는 큰 의미가 없다.

연습문제2

- ▶ 탐색 알고리즘에 대한 설명으로 **잘못된** 것은?
 - ① 순차 탐색에서 데이터 수가 n 개일 때, 최악의 경우 시간 복잡도는 n 이다.
 - ② 이진 탐색에서 데이터 수가 n 개일 때, 최악의 경우 시간 복잡도는 $\log_2 n$ 이다.
 - ③ 탐색 알고리즘에서 시간 복잡도를 계산하기 위한 핵심 연산자는 ++연산자이다.
 - ④ 이진 탐색 알고리즘의 반복 조건은 $\text{first} \leq \text{last}$ 이다.

연습문제3

- ▶ 어느 알고리즘의 $T(n)$ 함수가 다음과 같다. 빅-오를 판단하시오.

$$T(n) = 5n^3 + 3n^2 + 2^n + 1$$

- ① $O(5n^3)$
- ② $O(n^3)$
- ③ $O(3n^2)$
- ④ $O(2^n)$
- ⑤ $O(1)$

수고하셨습니다~



Chapter 01에 대한 강의를 마칩니다!