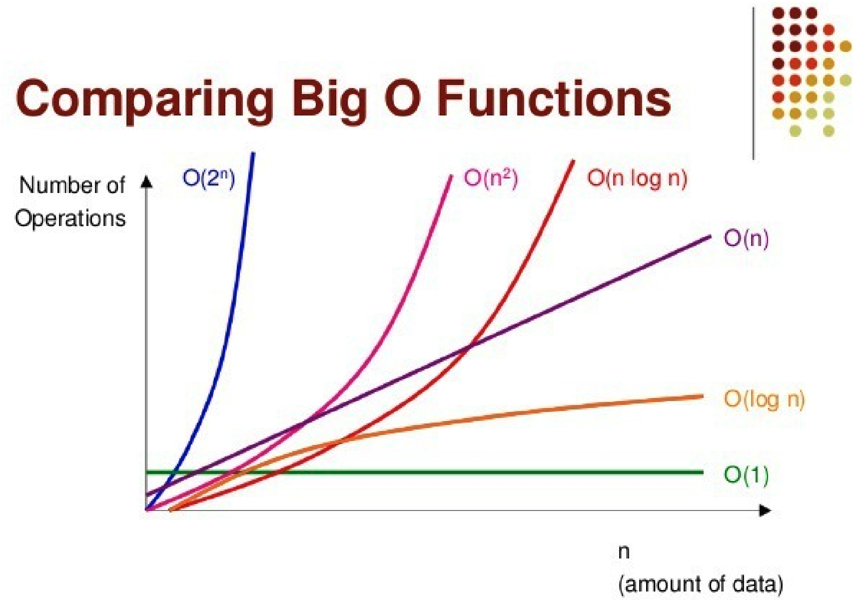
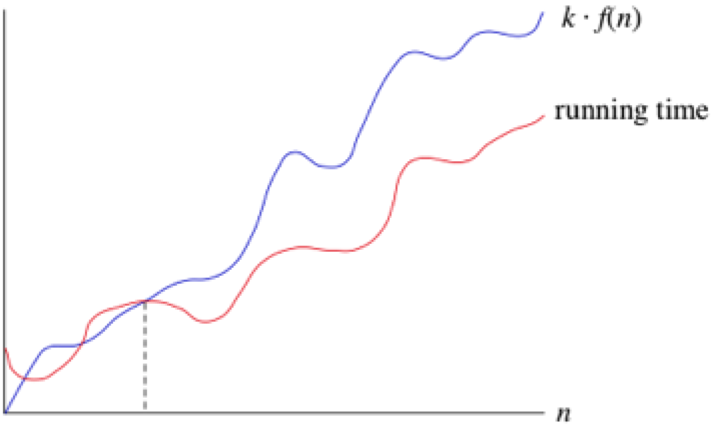
3 / 22

- 알고리즘 성능 분석 기초
  - 점화식의 점근적 표기법
    - 알고리즘이 얼마나 오래 걸리는지, 알기위한 것.
    - O(g(n)) → 상수항 무시, 영향력이 없는 항을 무시한다. → 점근적 표기법
    - 빅오 표기법에서의 성능 비교 - 상한 표기법
      - 점근적 상한만 알고 있을 때 사용하는 표기법
        - 최악의 경우에도 이 기준을 넘지 않는다.



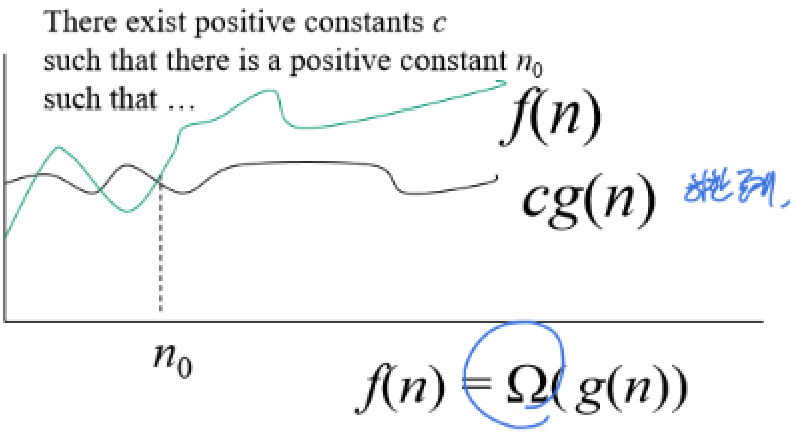
- O(1) < O(log n) < O(n) < O(n log n) < O(n^2) < O(2^n)
- 실행시간이 O(f(n))일때, 값 N과 상수 K에 대해 시행 시간은 최대 K * f(n) 이 된다. 실행 시간이 O(f(n))인 경우 다음과 같은 그림이 나온다.



- 위 그래프로 보면, 최악의 경우 케이스의 경우 모두 맞지만, 이전 케이스는 그렇지 않음을 볼 수 있다.
- 이분탐색을 예를 들어보면, 시간 복잡도는 O(log n)으로 생각할 수 있다.
- 과연 모든 경우에서 log n의 시간 복잡도를 가질까? 답은 아니다.
- 운이 좋아 1번에 찾는 케이스도 있을 수 있다. 허나 케이스의 복잡도가 높으면 높을 수록 log n에 수렴한다는 뜻이다.

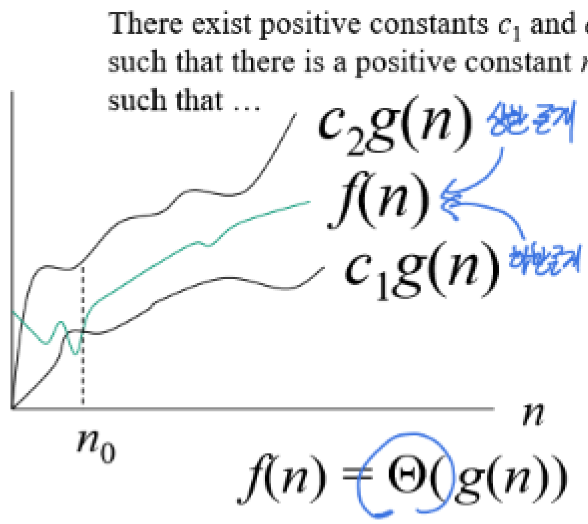
▪ 오메가 표기법

- 점근적 하한만 알고 있을 때, 사용하는 표기법
    - 아무리 빨라도 이 기준보다 빠를 수 없다.
    - 모든 n ≥ n0에 대해 0 ≤ c*g(n) ≤ f(n)인 양의 상수 c and n0가 존재한다.



▪ 세타 표기법

- 상한과 하한을 둘다 알고 있을 때 사용하는 표기법(상한 표기법, 하한 표기법 포함관계)
    - 모든 n ≥ n0에 대해 0 ≤ c1 * g(n) ≤ f(n) ≤ c2 * g(n) 인 양의 상수 c1, c2, n0이 존재한다.



- 세타 표기법이 시간 복잡도를 계산하는 가장 강력한 표기 방법이다. 역이 성립하기 때문

우선 바운드 관련 증명을 하라 할때 f(n)이 표기법(g(n))임을 증명하라는 문제로 출제된다.
이때, c*g(n) >(= f(n) 형태로 만든뒤, c >(= f(n) / g(n) 형태로 바꿔준다.
여기서, n부분에 1부터 넣어주며 c를 구하고, 이 값이 성립하는 no과 c값인지 하나씩 확인해주면된다.
* 세타 표기법 증명은 상한 증명, 하한 증명으로 둘다 증명하면 자연스럽게 증명됨.

- 반복 대치
    - 점화식을 반복하여 대치해가면서, 점화식을 전체적으로 나열시킨 후에 점근적 시간복잡도를 구하는 방법
    - 팩토리얼값을 구하는 알고리즘의 시간복잡도 분석과정

```
Factorial(n){
        if (n = 1) return 1;              // 1
        else return (n * Factorial(n-1)); // 2
}
```

- 결과값을 구하는 데 소요되는 시간을 T(n)이라 하면, T(n) = T(n - 1) + c으로 표현이 된다.
- n의 계승을 구하는 시간은 (n - 1)의 계승을 구하는 시간에 상수시간을 더한 형태이다.
- 여기서 상수시간 c는 1번문장과 2번문장의 곱셈 연산을 수행하는데, 걸리는 시간
- 이제 반복 대치를 통해 T(n)을 전개해보면

  $$\begin{aligned} T(n) &= T(n-1) + c \\ &= T(n-2) + c + c = T(n-2) + 2c \\ &= T(n-3) + c + 2c = T(n-3) + 3c \\ &\cdots \\ &= T(1) + (n-1)c \\ &\leq c + (n-1)c = cn \end{aligned}$$

  ($\because T(1)$: 1!을 계산하는데 소요되는 시간이고, 이는 상수시간 c보다 작은 수치일수 밖에 없음)

  $\therefore T(n) \leq cn$ 이므로 $T(n) = O(n)$ 으로 표현가능하다.

  https://www.youtube.com/watch?v=kg-bcK1ygIA&ab_channel=권오흠
- 부록
  - 재귀 알고리즘과 점화식의 관계를 이해한다.
  - 점화식의 점근적 분석을 이해한다.
  - 점화식 - recurrence → 재귀
    - 어떤 함수를 자신보다 더 작은 변수에 대한 함수와의 관계로 표현한 것
    - 재귀적 함수의 복잡도를 구하는데 유용함
    - 반복대치
      - 더 작은 문제에 대한 함수로 반복해서 대치해 나가는 해법
      - 팩토리얼
        - T(n) = T(n - 1) + c
        - 왜 앞에 n이 곱해지지 않는가 → 반복대치는 재귀의 패턴을 보는 것.
        - 하나의 함수에서 재귀는 1번만 이루어지는데, n - 1씩 재귀하기 때문

          - 추정후 증명
            - 결론을 추정하고 수학적 귀납법을 이용해 증명하는 방법
          - 마스터 정리
            - 형식에 맞는 점화식의 복잡도를 바로 알 수 있다.

- Recursion 이해와 연습
  - Hanoi Tower
    - 실제로 하노이 타워를 해보면, 재귀의 모습을 뛰고있음을 볼 수 있다.
    - 점화식 도출, 하노이는 총 2의 n승 - 1만큼 이동하며..
    - 두번의 재귀식을 실행 시킨다.
      - Base Condition (n == 1) 원반이 1개 밖에 없을 땐, A에서 C로 이동이 가능
      - if(n == 2) 일 때
      - A에서 C로 1을 옮긴다.
      - A에서 B로 2를 옮긴다.
      - C에서 B로 1을 옮긴다
      - https://www.youtube.com/watch?v=aPYE0anPZqI&ab_channel=얄팍한코딩사전