데이터 구조

강의 노트 2 알고리즘 분석

김유중, 박사 조교수
컴퓨터 과학 및 정보 공학부
한국 가톨릭 대학교, 대한민국

알고리즘 분석

- **문제(알고리즘 분석에서)** : 개발, 편의성, 지식의 발전 등을 위해 해결해야 할 **문제입니다.**
  - 일반적으로 컴퓨터 프로그래밍을 통해 해결하려고 시도했습니다.
  - 복잡성.

- 예
  - 최단 경로 찾기.
  - 인공지능 진공청소기의 청소 경로.

알고리즘 분석

- **알고리즘**: 일반적으로 일련의 문제를 해결하거나 계산을 수행하기 위한 유한한 명령어 집합입니다.
  - 일반적으로 컴퓨터 프로그래밍을 통해 구현됩니다.

- 알고리즘 기준
  - 입력
  - 출력
  - 명확성
  - 유한성
  - 효과성

알고리즘 분석

- 입력: 외부에서 공급되는 수량이 0개 이상입니다.
- 출력: 최소 한 개 이상의 수량이 생산됩니다.
- 명확성: 각 지침은 명확하고 모호하지 않습니다.
- 유한성: 알고리즘의 명령어를 추적하면 모든 경우에 대해 알고리즘은 유한한 수의 단계 후에 종료됩니다.
- 효과성: 모든 교육은 원칙적으로 연필과 종이만 사용하는 사람이 수행할 수 있을 정도로 기본적인 내용이어야 합니다.

알고리즘 분석

- 프로그램: 컴퓨터에서 특정 작업을 수행하는 방법을 설명하는 일련의 지침입니다.
  - 프로그래밍 언어.
  - C/C++, JAVA, Python, R 등.

- 알고리즘 분석
  - 알고리즘이 주어지고 정답으로 결정되었다고 가정합니다,
  - 필요한 리소스(일반적으로 시간)를 추정합니다.

알고리즘 분석

- 시간 복잡도: 알고리즘을 실행하는 데 걸리는 컴퓨터 시간을 설명하는 계산 복잡도입니다.
  - 일반적으로 알고리즘이 수행하는 기본 연산 횟수를 세어 추정합니다.

- 시간, T (□)
  - T(□) 는 프로그램이 취하는 컴파일 시간과 실행 시간(또는 실행) 시간입니다.
  - *TP*

- 실행 시간
  - 입력 크기 N에 따라 다릅니다(예: 숫자 10개와 100개의 합산).
  - T(□) 는 종종 어떤 크기의 입력에 대한 알고리즘의 실행 시간을 나타냅니다.
    는 *N*
    입
    니
    다.

점근 표기법

- 프로그램의 정확한 걸음 수를 계산하는 것은 매우 어려운 작업이며 심지어 필요하지도 않습니다.
  - 가장 좋은 경우 걸음 수
  - 최악의 경우 걸음 수
  - 평균 걸음 수

- Big O
  - 상한
  - $O(\square(\ ))$

- 오메가
  - 하한선
  - $\Omega(\square(\ ))$

- 세타
  - 상한과 하한이 모두 있습니다.

점근 표기법

– $\Theta(\square(\ ))$

수학 표기법

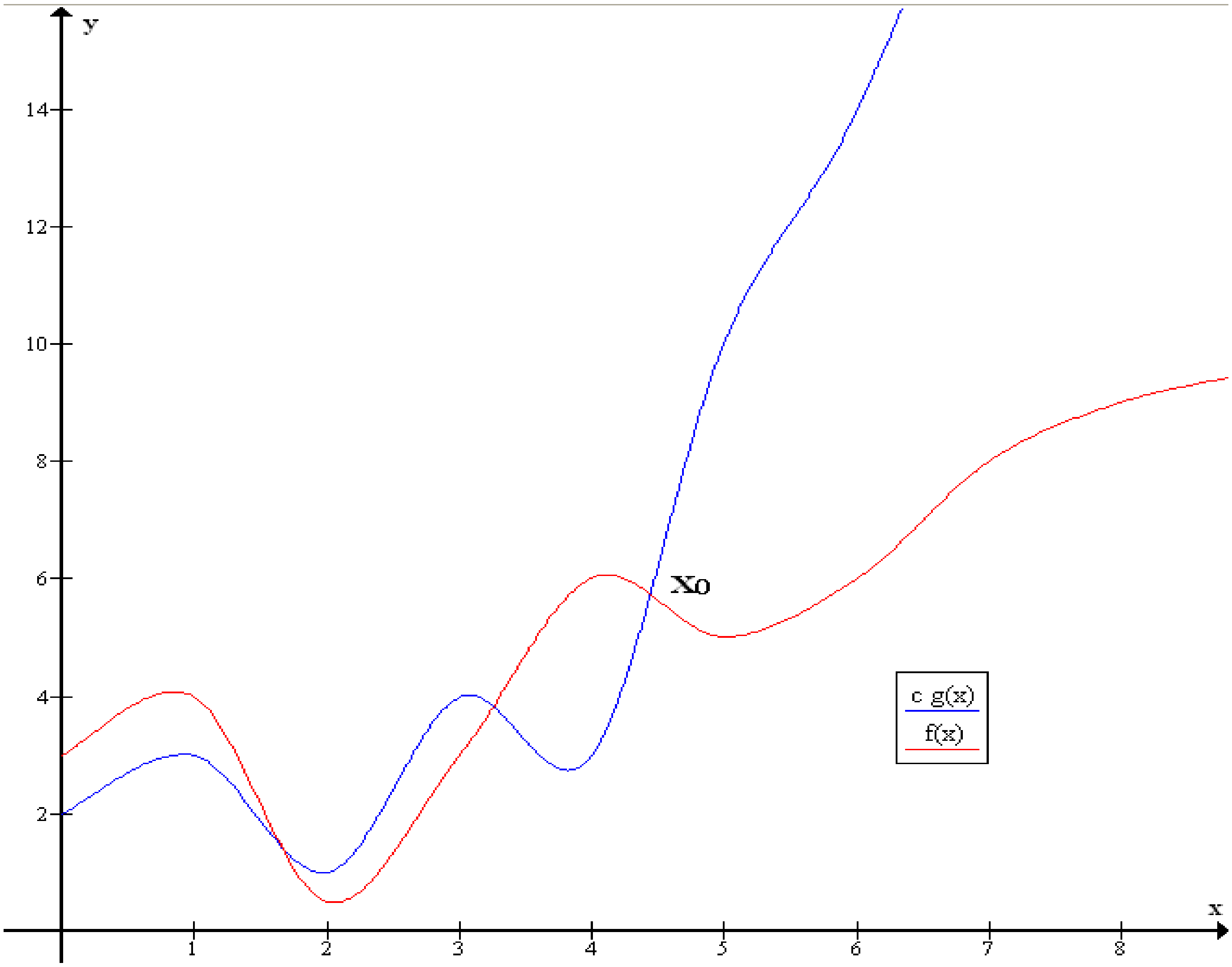
- $T(□) = O(f(□))$, 양수 상수 □와 n_0 가 존재하여 T가 되는 경우 $(□) \leq cf(N)$ $N \geq n_0$.
  - 빅오 함수는 f (□) 는 □(□)의 상한입니다.

- 정리
  - 만약 (□) $= amN^m + am\text{-}1N^{m-1} + \cdots + a1N + a0$, 그러면 T (□) = □ (□□) .

- 이러한 표기는 상대적인 성장률에 대한 표기입니다.
  - 예. (□) T= 1000N의 (□) 경우, (□2) T= O 는 N가 1000 이상인 경우 1000$N \leq N^2$ 이므로 □= □입니다.

수학 표기법

- 예



https://en.wikipedia.org/wiki/Big_O_notation

수학 표기법

- $T(□) = \Omega(f(□))$, 양수 상수 □와 n_0 가 있는 경우 $T(□) \geq cf(N)$ $N \geq n_0$.
  - 오메가 함수는 $f(□)$ 는 □(□)의 하한입니다.

- 정리
  - Ω (□) = $a_m N^m + a_{m-1} N^{m-1} + \cdots + a_1 N +$ □0이고 $a_m > 0$이면 $T(□) =$ (□□).

수학 표기법

- $T(N) = \Theta(f(N))$ 양수 상수 $c1$, $c2$, $n0$이 다음과 같은 경우에만 해당합니다.
  $c1f(□) \le T(□) \le c2f(□)$ 모든 □, $n \ge n0$에 대해.
  - 세타 함수는 빅 O와 오메가 표기법보다 더 정확합니다.

- $T(□) = \Theta(f(□))$ □ (□) 의 상한과 하한은 □ (□).

- 정리
  - Θ (□) = $amN^m + am\text{-}1N^{m-1} + \cdots + a1N +$ □0이고 $am > 0$이면 (□) = (□□).

수학 표기법

- 일반적인 실수
  - 빅 O 표기법에서 '=' 기호는 "같음"을 의미하지 않습니다.
  - 점근 표기법에서는 '='를 "is"로 읽습니다.
  - 대략적이고 모호합니다.
  - 대략적인 설명을 허용합니다.
  - 시간 복잡성뿐만이 아닙니다.

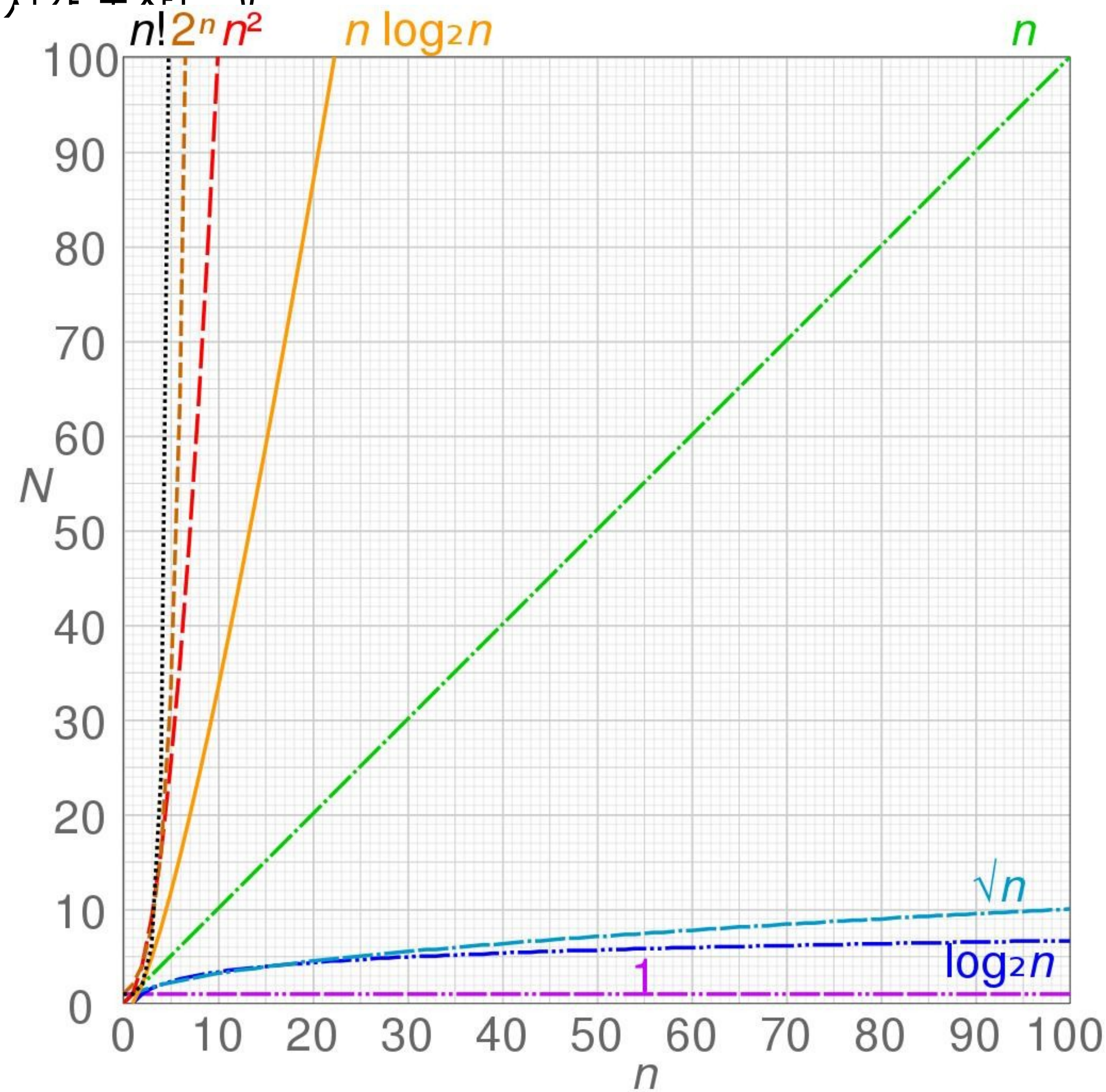
수학 표기법

- 일반적인 시간 복잡도 표

□(□)	**이름**
1	상수
로그 □	로그
로그^{2} □	로그 제곱
□	리니어
□ 로그 □	선형
□2	Quadratic
□□	다항식
□□	지수

수학 표기법

- 일반적인 시간 복잡도 표



https://en.wikipedia.org/wiki/Time_complexity

수학 표기법

- 몇 가지 기본 정리
  - 1 (□) = O (□)) 및 □2 (□) = □O□ (□))를 클릭한 다음
    - (a) □1(□) + □2(□) = 최대(□(□ (□)) (□, □))
    - (b) 1 T(N) × T_2(□) = O(□(□) (□)))
  - T (□) 가 □ 의 다항식 함수라면, T (□) = Θ (N)^k
  - $\log^k N = O(N)$는 상수 □에 대해 로그가 선형 성장보다 느리게 성장한다는 의미입니다.

수학 표기법

- 사람들은 간단한 표기법을 선호합니다.
  - $T = N^2 +$ (3)일 때, 다음이 사실입니다 (□) = □ (2□²) 및 □ (□) = □(² □+) .
    하지만 사람들은 □(□) = □(□²)로 표기하는 것이 올바른 표기법입니다.

- 두 함수를 비교하여 성장률을 결정할 수 있습니다.

  $c = \lim_{N\to\infty} \frac{f(N)}{g(N)}$이면 ...

- f라고 말하는 것 (□) ≤ $O(g)$ (□)은 좋지 않습니다. 왜냐하면 Big-Oh는 이미 '보다 작음'을 의미하기 때문입니다.
  같음".

러닝 시간 계산

- 런타임 계산은 실제 런타임의 상한이기 때문에 종종 Big -Oh 계산을 의미합니다.
  - 실제 실행 시간을 정확히 계산하는 것은 종종 불가능합니다.

- 예

```
int Sum(int N)
{
    int i, PartialSum;

    PartialSum = 0;
    for( i=0 ; i<=N ; i++ )
        PartialSum += i*i*i;

    return PartialSum;
}
```

일반 규칙

- 규칙 1: (루프 실행 시간) = (내부 문의 실행 시간) * (반복 횟수)

- 규칙 2: (중첩된 루프의 실행 시간) = (외부 루프의 반복 횟수) 루프) * (내부 루프 실행)

```
for( i=0 ; i<N ; i++ )
    for( j=0 ; j<N/2 ; j++ )
        k++;
```

- 규칙 3: (연속된 두 문장의 실행 시간) = (연속된 두 명령문의 첫 번째 항목) + (두 번째 항목의 실행 시간)

- 규칙 4: (if -else 문의 실행 시간) = (조건의 실행 시간) + (두 조건문의 실행 시간 중 큰 값)

```
if( condition ) S1();
else S2();
```

재귀 함수

- 계승 함수

```
int factorial(int N)
{
    if( N<=1 ) return 1;
    else return N*factorial(N-1);
}
```

- 피보나치 수열

```
int fib(int N)
{
    if( N<=1 ) return 1;
    else return fib(N-1)+fib(N-2);
}
```

최대 연속 문제

- (음수일 수 있는) 정수 A_1 , A_2 , … , A_N가 주어졌을 때, 다음의 최대값을 구합니다. σi□ □□. 편의상 모든 정수가 음수인 경우 최대 수열 합은 0이 됩니다.
- 예: 입력값 -2, 11, -4, 13, -5, -2의 경우 답은 20(A2 ~ A4)입니다.

- 알고리즘 1

```
int MaxSubsequenceSum_1(const int A[],int N)
{
    int ThisSum, MaxSum, i, j, k;

    MaxSum = 0;
    for( i=0 ; i<N ; i++ ) {
        for( j=i ; j<N ; j++ ) {
            ThisSum = 0;
            for( k=i ; k<=j ; k++ ) ThisSum += A[k];
            if( ThisSum > MaxSum) MaxSum = ThisSum;
        }
    }
    return MaxSum;
}
```

최대 연속 문제

- 알고리즘 22

```
int MaxSubsequenceSum_2(const int A[],int N)
{
	int ThisSum, MaxSum, i, j;

	MaxSum = 0;
	for( i=0 ; i<N ; i++ ) {
		ThisSum = 0;
		for( j=i ; j<N ; j++ ) {
			ThisSum += A[j];
			if( ThisSum > MaxSum ) MaxSum = ThisSum;
		}
	}
	return MaxSum;
}
```

-2, 11, -4, 13, -5, -2

최대 연속 문제

- 알고리즘 23

```
int MaxSubsequenceSum_4(const int A[],int N)
{
    int ThisSum, MaxSum, i;

    ThisSum = MaxSum = 0;
    for( i=0 ; i<N ; i++ ) {
        ThisSum += A[i];
        if( ThisSum > MaxSum ) MaxSum = ThisSum;
        else if( ThisSum < 0 ) ThisSum = 0;
    }
    return MaxSum;
}
```

-2, 11, -4, 13, -5, -2

로그 실행 시간

- 알고리즘은 O (로그) 가 걸리는 경우, 문제 크기를 a만큼 줄이는 데 일정한 시간이 걸리면
  분수(보통 1/2)로 설정합니다.

- 예시: 예: 이진 검색
  - 크기가 N인 미리 정렬된 숫자 목록에서 숫자 찾기.

```
int BinarySearch(const int A[],int N,int X)
{
    int Low, Mid, High;
    Low = 0; High = N-1;
    while( Low <= High ) {
        Mid = (Low+High)/2;
        if( A[Mid]<X ) Low = Mid+1;
        else if( A[Mid]>X ) High = Mid-1;
        else return Mid;
    }
    return -1;
}
```

로그 실행 시간

-5, -4, -2, 11, 13

로그 실행 시간

- 예시: 최대공약수(GCD)에 대한 유클리드의 알고리즘.

```
unsigned int GCD(unsigned int M, unsigned int N)
{
    unsigned int Rem;

    if( M<N ) swap(M,N);
    while( N > 0 ) {
        Rem = M%N;
        M = N;
        N = Rem;
    }
    return M;
}
```

로그 실행 시간

- 예시: 지수화
  - X^N 계산하기

```
int Pow1(int X,unsigned int N)
{
    int i, Mul=1;
    for( i=0; i<N ; i++ ) Mul *= X;
    return Mul;
}

int Pow2(int X,unsigned int N)
{
    if( N == 0 ) return 1;
    if( N == 1 ) return X;
    if( N%2 == 0 ) return Pow2(X*X,N/2);
    else return Pow2(X*X,N/2)*X;
}
```

더 많은 이

- 최악의 실행 시간과 평균 실행 시간 비교
  - 최악의 실행 시간: 실행에 걸리는 가장 긴 시간으로, 일반적으로 예상되는 시간입니다.
  - 평균 실행 시간: 일반적인 입력에 대한 예상 평균 시간으로, 때때로 추정하기 어렵습니다.
  - 이진 검색에서 최상의 경우 실행 시간은 1, 최악의 경우 실행 시간은 시간은 로그N이고 평균은 1에서 로그N 사이입니다.

- 최적화
  - 알고리즘의 실행 시간이 달성 가능한 최적의 시간인지 확인합니다. 는 일반적으로 매우 어렵습니다.

- 실용적인 문제
  - 알고리즘 분석에는 포함되지 않았지만 실제로 실행 중인 프로그램에 영향을 미치는 많은 요소가 있습니다.
  - 데이터 I/O 시간: 디스크와의 읽기/쓰기 시간
  - 메모리 관리: 메모리 할당 및 캐시 미스를 위한 시스템 호출

더 많은 이

– 함수 호출: 서브루틴 함수 호출에는 오버헤드가 있습니다.