1-3 수행시간의 점근표기법

- 수행시간은 알고리즘이 수행하는 기본 연산 횟수를 입력 크기에 대한 함수로 표현
- 이러한 함수는 다항식으로 표현되며 이를 입력의 크기에 대한 함수로 표현하기 위해 점근표기법(Asymptotic Notation)이 사용
- O (Big-Oh)-표기법
- Ω (Big-Omega)-표기법
- Θ (Theta)-표기법

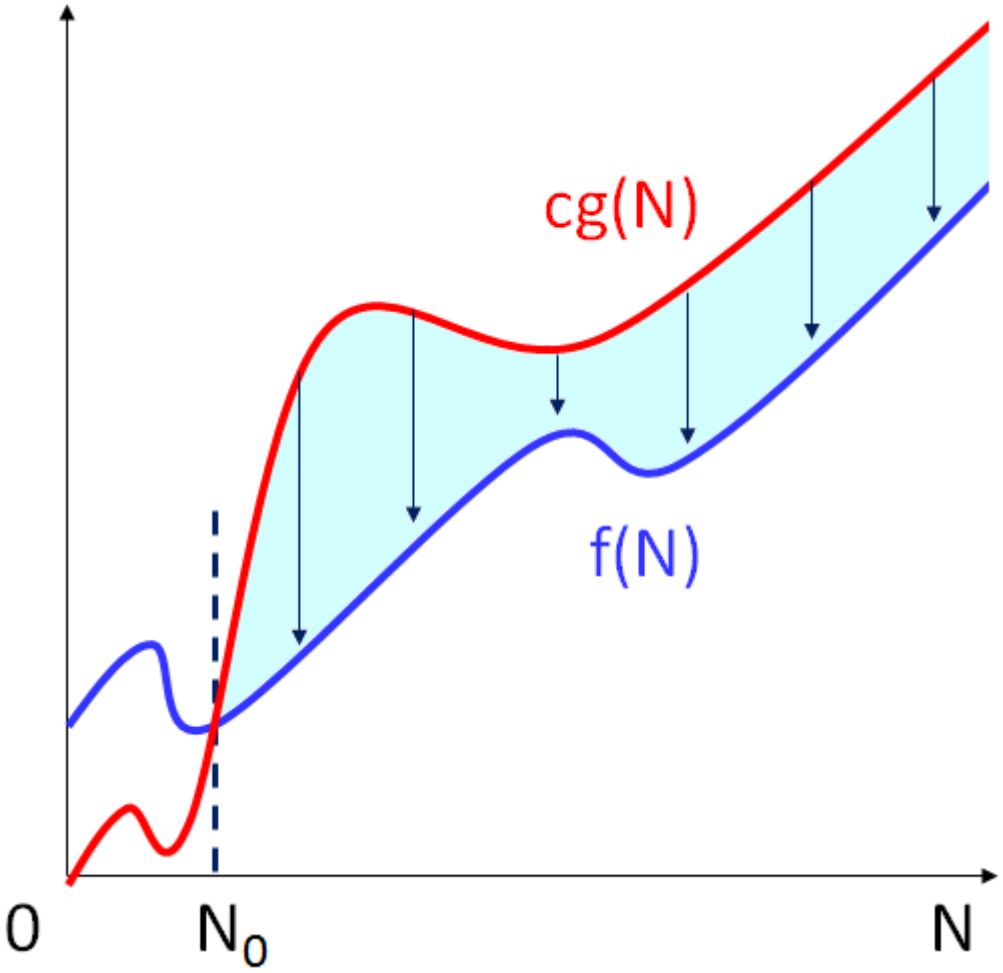
O (Big-Oh)-표기법

- **[O-표기의 정의]**

  모든 N ≥ N0에 대해서 f(N) ≤ cg(N)이 성립하는
  양의 상수 c와 N_0가 존재하면, f(N) = O(g(N))이다.

- **O-표기의 의미: N0과 같거나 큰 모든 N (즉, N_0 이후의 모든 N) 대해서 f(N)이 cg(N)보다 크지 않다는 것**
  - f(N) = O(g(N))은 N_0 보다 큰 모든 N 대해서
    f(N)이 양의 상수를 곱한 g(N)에 미치지 못한다는 뜻
- **g(N)을 f(N)의 상한(Upper Bound)이라고 한다.**

O (Big-Oh)-표기법



$f(N) = O(g(N))$

O (Big-Oh)-표기법

[예제]

$f(N) = 2N^2 + 3N + 5$이면,

양의 상수 c 값을 최고 차항의 계수인 2보다 큰 4를 택하고
$g(N) = N^2$으로 정하면,
3보다 큰 모든 N에 대해 $2N^2 + 3N + 5 < 4N^2$이 성립한다.
즉, $f(N) = O(N^2)$이다.

- 물론 $2N^2 + 3N + 5 = O(N^3)$도 성립하고, $2N^2 + 3N + 5 = O(2^N)$도 성립.
  그러나g(N)을 선택할 때에는 정의를 만족하는
  가장 차수가 낮은 함수를 선택하는 것이 바람직하다.

- $f(N) \leq cg(N)$을 만족하는 가장 작은 c값을 찾지 않아도 됨.
  왜냐하면 $f(N) \leq cg(N)$을 만족하는 양의 상수 c와 N_0가 존재하기만 하면
  되기 때문이다.

O (Big-Oh)-표기법

- 주어진 수행시간의 다항식에 대해 O-표기를 찾기 위해 간단한 방법은 다항식에서 최고 차수 항만을 취한 뒤, 그 항의 계수를 제거하여 g(N)을 정한다.

- 예를 들어, $2N^2 + 3N + 5$에서 최고 차수항은 $2N^2$이고, 여기서 계수인 2를 제거하면 N^2이다. 따라서 $2N^2 + 3N + 5 = O(N^2)$이다.

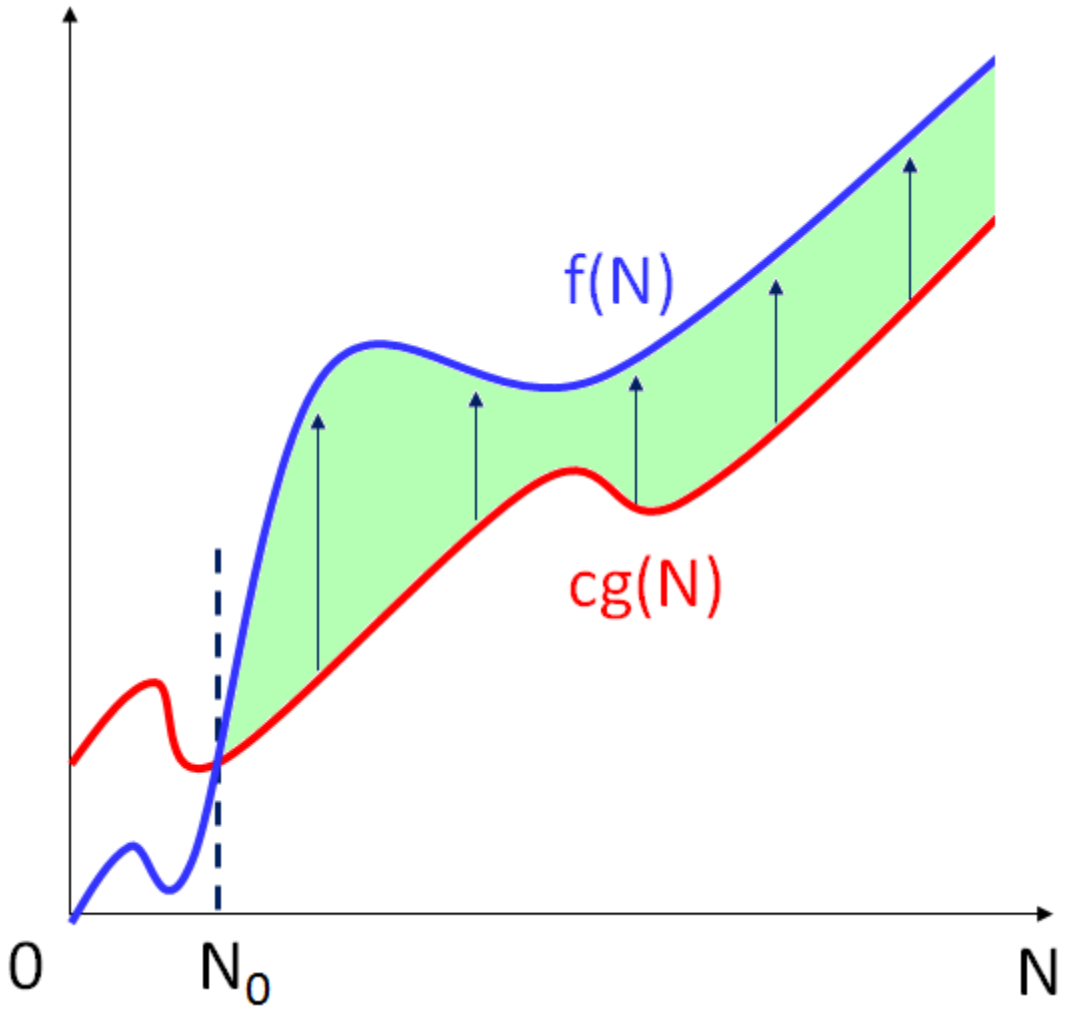
Ω-표기법

[Ω-표기의 정의]

모든 $N \geq N_0$에 대해서 $f(N) \geq cg(N)$이 성립하는
양의 상수 c와N_0가 존재하면, $f(N) = \Omega(g(N))$이다.

- Ω-표기의 의미는 N_0 보다 큰 모든 N 대해서f(N)이 cg(N)보다 작지 않다는 것이다.
- f(N) = Ω(g(N))은 양의 상수를 곱한 g(N)이f(N)에 미치지 못한다는 뜻
- g(N)을 f(N)의 하한(Lower Bound)이라고 한다.

Ω-표기법



$f(N) = \Omega(g(N))$

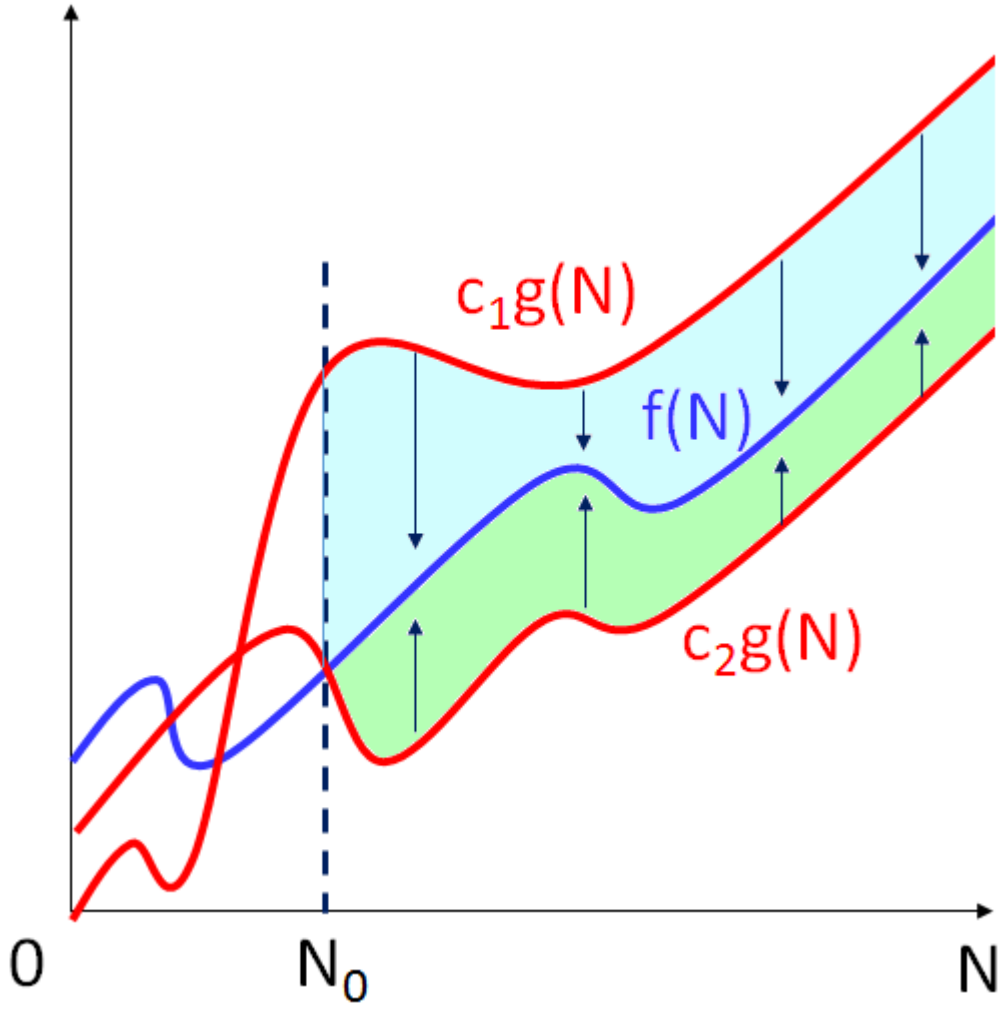
Θ-표기법

[Θ-표기의 정의]

모든 $N \geq N_0$에 대해서 $c_1 g(N) \geq f(N) \geq c_2 g(N)$이 성립하는 양의 상수 c_1, c_2, N_0가 존재하면, $f(N) = \Theta(g(N))$이다.

- Θ-표기는 수행시간의 O-표기와 Ω-표기가 동일한 경우에 사용한다.
- $2N^2 + 3N + 5 = O(N^2)$과 동시에$2N^2 + 3N + 5 = \Omega(N^2)$ 이므로, $2N^2 + 3N + 5 = \Theta(N^2)$이다.
- $\Theta(N^2)$은N^2과 $(2N^2 + 3N + 5)$이 유사한 증가율을 가지고 있다는 뜻이다. 따라서 $2N^2 + 3N + 5 \neq \Theta(N^3)$ 이고, $2N^2 + 3N + 5 \neq \Theta(N)$이다.

Θ-표기법



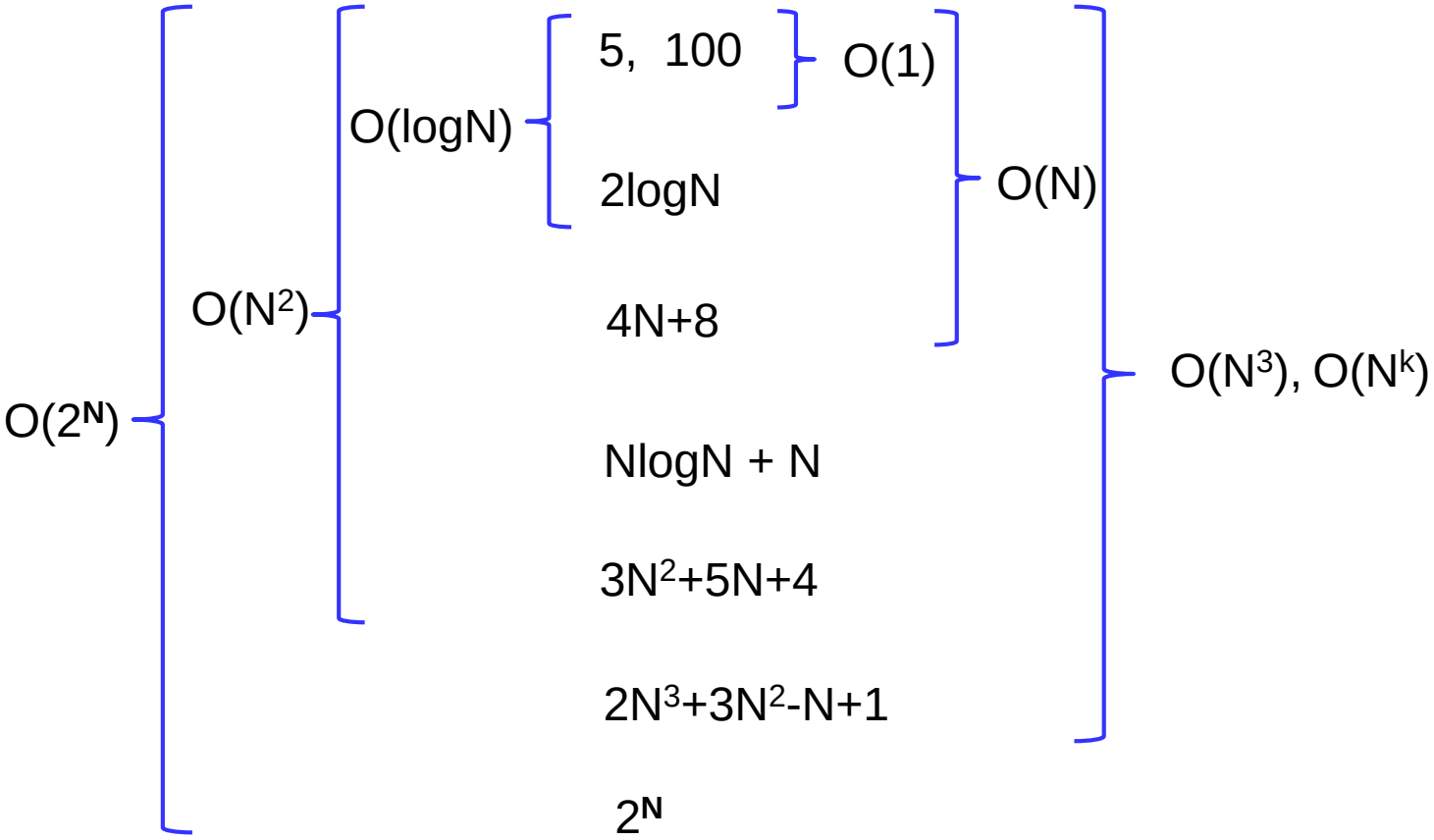
$f(N) = \Theta(g(N))$

자주 사용되는 함수의 O-표기와 이름

- **알고리즘의 수행시간은 주로 O-표기를 사용**
  - 보다 정확히 표현하기 위해 Θ-표기를 사용하기도 함

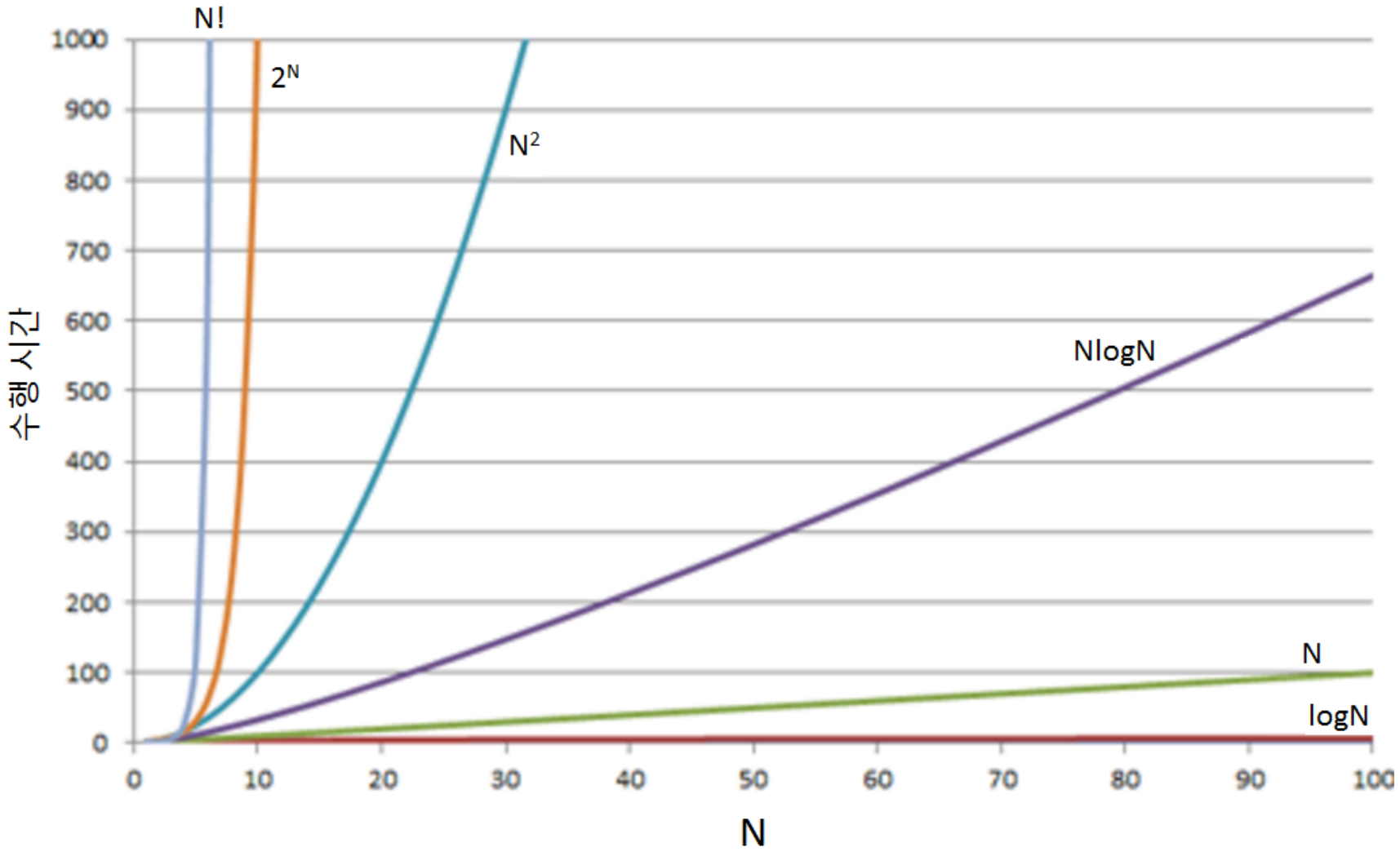
- **자주 사용하는 O표기와 그 이름들**
  - O(1) 상수시간(Constant Time)
  - O(logN) 로그(대수)시간(Logarithmic Time)
  - O(N) 선형시간(Linear Time)
  - O(NlogN) 로그선형시간(Log-linear Time)
  - $O(N^2)$ 제곱시간(Quadratic Time)
  - $O(N^3)$ 세제곱시간(Cubic Time)
  - $O(2^N)$ 지수시간(Exponential Time)

자주 사용되는 함수의 O-표기와 이름



O-표기의 포함 관계

O 함수의 증가율 비교


N!
2N
N2
NlogN
N
logN
수행 시간
N
1000
900
800
700
600
500
400
300
200
100
0
0
10
20
30
40
50
60
70
80
90
100

O 함수의 증가율 비교

- 10^9 instructions/sec의 속도를 가지는 컴퓨터에서의 실행시간 추정량

n	O(n)	O(nlogn)	$O(n^2)$	$O(n^3)$	$O(n^4)$	$O(2^n)$
10	.01μs	.03μs	.1μs	1μs	10μs	1μs
20	.02μs	.09μs	.4μs	8μs	160μs	1ms
30	.03μs	.15μs	.9μs	27μs	810μs	1sec
40	.04μs	.21μs	1.6μs	64μs	2.56ms	18.3min
50	.05μs	.28μs	2.5μs	125μs	6.25ms	13day
100	.10μs	.66μs	10.0μs	1ms	100ms	$4*10^{13}$yr
1,000	1.00μs	9.96μs	1.0ms	1sec	16.67min	$32*10^{283}$yr
10,000	10.00μs	130.03μs	100.0ms	16.67min	115.7day	
100,000	100.00μs	1.66ms	10.0sec	11.57day	317yr	
1,000,000	1.00ms	19.92ms	16.67min	31.71yr	$3.17*10^7$yr	

1-4 자바 언어에 대한 기본적인 지식

- **자바 언어는 객체지향 프로그래밍 언어로서 클래스를 선언하여 데이터를 저장하고 메소드(Method)를 선언하여 객체들에 대한 연산을 구현.**

```
public class 클래스이름 {
    인스턴스 변수; //멤버 변수라고도 함
    객체 생성자;
    생성된 객체에 대한 연산을 위한 메소드;
}
```

- 인스턴스 변수: 객체에 정보를 저장
- 객체 생성자(Constructor): 객체를 생성
- 메소드: 객체 혹은 객체 내부 인스턴스 변수에 대한 연산 수행

자바 언어에 대한 기본적인 지식

- **[예제] 학생을 객체로 표현하기 위해 Student 클래스**
  - 각 Student 객체는 이름과 학번을 각각 저장하는 인스턴스 변수 name과 id를 가짐

```java
public class Student {
    private String name;
    private int id;
    public Student(String n
        name = newName;
        id   = newId;
    }
    public String getName()
    public int    getId()
}
```

```cpp
#include    <string>

class Student
{
private :
    string name ;
    int id ;
public :
    Student(string newName, int newID)
    {
        name = newName ;
        id = newID ;
    }
    string getName() { return name ;}
    int getID() { return id ;}
}
```

자바 언어에 대한 기본적인 지식

```java
import java.util.Scanner;

public class Rectangle {
    int width;
    int height;

    public int getArea() {
        return width*height;
    }
}

public class RectApp{
    public static void main(String[] args) {
        Rectangle rect = new Rectangle(); // 객체 생성

        Scanner scanner = new Scanner(System.in);
        System.out.print(">> ");
        rect.width = scanner.nextInt();
        rect.height = scanner.nextInt();
        System.out.println("사각형의 면적은 " + rect.getArea());

        scanner.close();
    }
}
```

자바 언어에 대한 기본적인 지식

- 배열(Array): 동일한 타입의 원소들이 연속적인 메모리 공간에 할당되어 있는 기초적인 자료구조
- 각 항목이 하나의 원소에 저장
- C++에서는 정적배열과 동적 배열이 가능하지만, Java에서는 항상 new를 이용하 동적 배열만 사용할 수 있음

```
데이터타입[] 배열이름 = new 데이터타입[배열크기];
```

```
int[] a = new int[10];
String[] s = new String[10];
Student[] st = new Student[100];
```

자바 언어에 대한 기본적인 지식

- **if-문: 조건문으로서 하나 또는 여러가지 조건을 검사하도록 선언**
  - 조건식이 true이면 해당 명령문을 수행하고, false이면 else의 명령문을 수행

```
if (조건식) {
    명령문;
} else{
    명령문;
}
```

```
if (조건식) {
  명령문;
} else if (조건식) {
  명령문;
…
} else {
  명령문;
}
```

자바 언어에 대한 기본적인 지식

- **반복문:**
  for-문은 두 가지 방식으로 사용가능한데, 초기화식, 조건식, 증감식을 통해 반복문을 제어하는 방식과 배열의 모든 원소들을 차례로 읽으며 명령문을 처리하는 방식
- while -문은 조건식이 만족된 동안에만 반복 수행

```
for (초기화식; 조건식; 증감식){
    명령문;
}
```

```
for (데이터타입 변수; 배열){
    명령문;
}
```

```
while (조건식){
    명령문;
}
```

자바 언어에 대한 기본적인 지식

- Comparable 은 java.lang에 선언된 인터페이스
- 객체의 하나의 멤버만을 기준으로 객체들을 정렬할 때 사용
- compareTo 메소드

```
public interface Comparable {
    public int compareTo(T other);
}
```

x.compareTo(y)는
- x < y이면 음수
- x = y이면 0
- x > y이면 양수 리턴

자바 언어에 대한 기본적인 지식

- import 문은 이미 정의되어 있는 클래스를 사용하기 위한 선언문

```
import 메인패키지이름.서브패키지이름. ••• . 클래스이름;
```

- Java.lang은 기본적으로 자바에서 제공하는 패키지로서 import할 필요가 없음
- 유용한 유틸리티를 제공하는 패키지인 java.util의 클래스들

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;
import java.util.PriorityQueue;
import java.util.NoSuchElementException;
import java.util.EmptyStackException;
```

1-5 순환(Recursion)

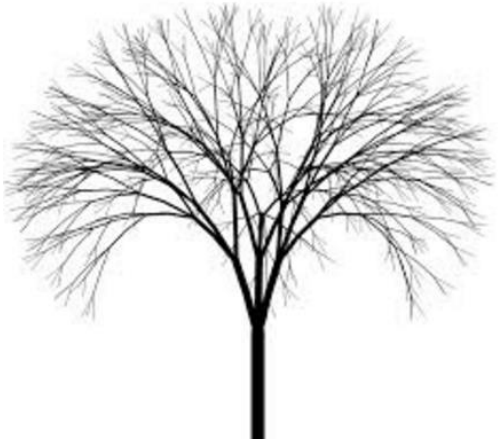
- 메소드의 실행 과정 중 스스로를 호출하는 것
- 팩토리얼, 조합을 계산하기 위한 식의 표현, 무한한 길이의 숫자 스트림을 만들기, 분기하여 자라나는 트리 자료구조, 프랙털(Fractal) 등에서 기본 개념으로 사용

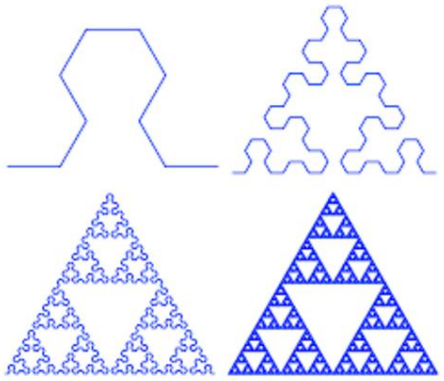
$$n! = n \cdot (n-1)!$$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

$$F_n = F_{n-1} + F_{n-2},\ F_0 = 0,\ F_1 = 1$$

$$X_n = (aX_{n-1} + c)\ \%\ m,\ X_0 = seed$$





순환(Recursion)

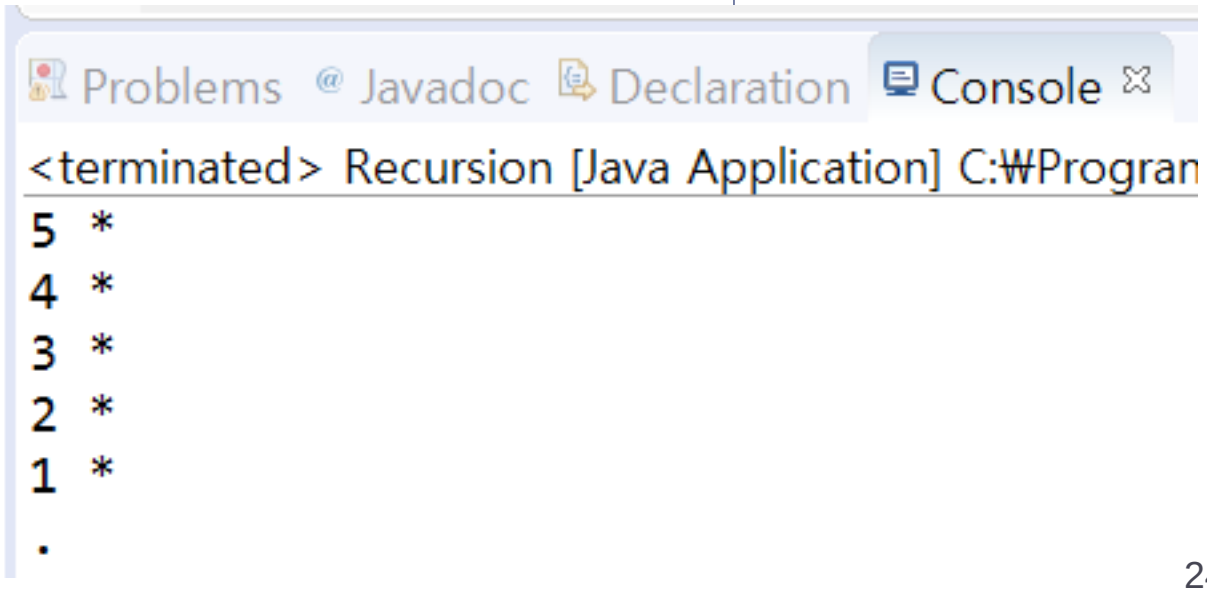
- **메소드가 자기 자신을 호출할 때 무한 호출을 방지해야 함**
  - 다음의 자바 프로그램을 실행시키면 StackOverflowError 발생
  - 이는 스스로의 호출을 중단시킬 수 있는 조건문이 없기 때문

```
01 public class Recursion {
02     public void recurse(){
03         System.out.println("*");;
04         recurse();
05     }
06     public static void main(String[] args){
07         Recursion r = new Recursion();
08         r.recurse();
09     }
10 }
```

Problems Javadoc Declaration Console

```
<terminated> Recursion [Java Application] C:₩Program Files₩Java₩jdk1.8.0_121₩bin₩javaw.exe
*
*
*
Exception in thread "main" java.lang.StackOverflowError
        at sun.nio.cs.ext.DoubleByte$Encoder.encodeLoop(DoubleByte.java:617)
        at java.nio.charset.CharsetEncoder.encode(CharsetEncoder.java:579)
        at sun.nio.cs.StreamEncoder.implWrite(StreamEncoder.java:271)
        at sun.nio.cs.StreamEncoder.write(StreamEncoder.java:125)
```

```java
public class Recursion {
    public void recurse(int count){
        if (count <= 0 )
            System.out.println(".");
        else {
            System.out.println(count+" *");
            recurse(count-1);
        }
    }
    public static void main(String[] args){
        Recursion r = new Recursion();
        r.recurse(5);
    }
}
```



순환(Recursion)

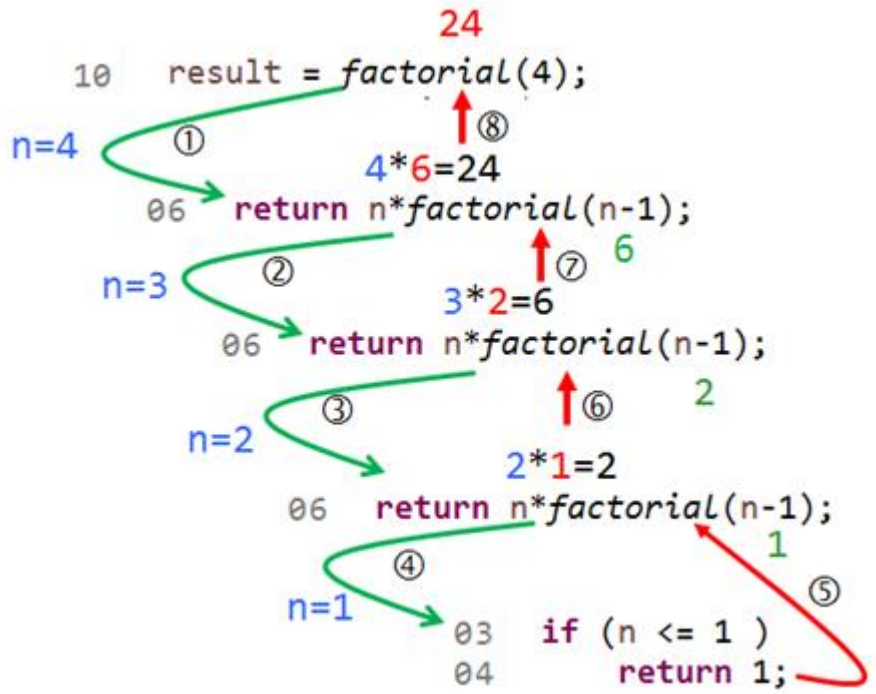
- **순환으로 구현된 메소드는 두 부분으로 구성 하나는**
  - 기본(Base) case: 스스로를 더 이상 호출하지 않는 부분
  - 순환 case: 스스로를 호출하는 부분

  - 무한 호출을 방지하기 위해 선언한 변수 또는 수식의 값이
    호출이 일어날 때마다 순환 case에서 감소되어
    최종적으로 if-문의 조건식에서 기본 case를 실행하도록 제어해야 함

팩토리얼 계산 자바 프로그램

- Line 10에서 factorial(4)로 메소드를 호출
- [그림]의 번호 순서대로 수행되어 24를 출력

```
01 public class Factorial {
02     public static int factorial(int n){
03         if (n <= 1 )
04             return 1;
05         else
06             return n*factorial(n-1);
07     }
08     public static void main(String[] args)
09         int result;
10         result = factorial(4);
11         System.out.println(result);
12     }
```



반복문으로 팩토리얼을 계산하는 프로그램

- 반복문을 이용한 계산은 메소드 호출로 인해 시스템 스택을 사용 하지 않으므로 순환을 이용한 계산보다 매우 간단하며 메모리도 적게 사용

```
01 public class Factorial {
02     public static void main(String[] args){
03         int n = 4;
04         int result = 1;
05         for (int i = 1; i<=n; i++)
06             result = result * i;
07         System.out.println(result);
08     }
09 }
```

반복문으로 팩토리얼을 계산하는 프로그램

- 일반적으로 순환은
  프로그램(알고리즘)의 가독성을 높일 수 있는 장점을 갖지만
  시스템 스택을 사용하기 때문에 메모리 사용 측면에서 비효율적
- 반복문으로 변환하기 어려운 순환도 존재하며,
  억지로 반복문으로 변환하는 경우
  프로그래머가 시스템 스택의 수행을 처리해야 하므로
  프로그램이 매우 복잡해질 수 밖에 없음
  - Ex. Honoi Tower Algorithm
- 반복문으로 변환된 프로그램의 수행 속도가
  순환으로 구현된 프로그램보다 항상 빠르다는 보장 없음

요약 (1)

- **자료구조는 일련의 동일한 타입의 데이터를 정돈하여 저장한 구성체.**
- **추상데이터타입은 데이터와 그 데이터에 관련된 추상적인 연산들로서 구성.**
  - 추상적이란 연산을 구체적으로 어떻게 구현하여야 한다는 상세를 포함하고 있지 않다는 뜻. 자료구조는 추상데이터타입을 구체적(실제 프로그램)으로 구현한 것.
- **수행시간은 알고리즘이 수행하는 기본적인 연산 횟수를 입력 크기에 대한 함수로 표현.**

요약 (2)

- **알고리즘의 수행시간 분석 방법**
  - 최악경우, 평균경우, 최선경우 분석, 상각분석
- **점근표기법**
  - 입력 크기가 증가함에 따른 수행시간의 간단한 표기법
  - O(Big-Oh)-표기: 점근적 상한
  - Ω(Big-Omega)-표기: 점근적 하한
  - Θ(Theta)-표기: 동일한 증가율