

DM 13-2. Recursion & Algorithm

Dr. Minsu Cho

School of Information Convergence

mcho@kw.ac.kr



Announcement

- Homework #2
 - Due. : 12/6
- 기말고사
 - 일시 : 12/12(월) 18:00 ~, 새빛관 101호
 - 범위 : 9.2. Counting ~
 - 문제유형 : 중간고사와 유사

목차

- 수열
- 점화 관계
- 점화 관계의 응용 : 하노이 탑 문제
- 알고리즘의 개념과 표현
- 알고리즘의 복잡도
- 유클리드 알고리즘
- 재귀 알고리즘
- 탐색 알고리즘

수열

- **수열(Sequence)**

- 수 또는 다른 대상들이 한 줄로 배열된 순서 있는 나열로 $a_1, a_2, a_3, \dots, a_n$ 의 형태로 표시
- 수열의 예 : 양의 홀수를 크기 순으로 나열한 1, 3, 5, 7, ...
- 수열에 나타나는 원소 : 항(term)
 - a_1 : 초항 또는 첫째 항,
 - a_2 : 2항 또는 둘째 항,
 - a_n : n 항 또는 일반항

- **수열(Sequence)**
 - 반드시 일정한 규칙을 갖고 수가 나열되어야 할 필요는 없음
 - 예를 들어, $-2, 1.2, 0.4, 55, -30$ 과 같은 것도 수열
 - 그러나, 일반적으로 일정한 규칙을 가진 수열을 많이 다룸
 - 규칙에 따라 등차수열, 등비수열, 조화수열 등으로 구분
 - 수열의 분류
 - 항의 수가 유한 : 유한수열
 - 항의 수가 무한 : 무한수열

수열의 표시 방법

- 수열(Sequence)의 표시 방법
 - 원소를 나열하여 표시하는 방법
 - 수열의 원소를 직접 나열하여 표현하는 방법 : 예, 2, 4, 6, ...
 - 일반항으로 표현하는 방법
 - 수열의 n 번째 항인 a_n 을 수식 등의 형태로 명시하는 방법
 - (예) $a_n = 2n$ 일 때 20번째 항의 값은 $a_{20} = 2 \times 20 = 40$
 - 귀납적 정의로 표시하는 방법
 - 수열의 n 번째 항이 앞의 항들 a_1, a_2, \dots, a_{n-1} 로부터 귀납적으로 결정될 때 표시하는 방식으로, 이 관계를 점화 관계라고 함

수열의 표시 방법

- 수열(Sequence)의 표시 방법
 - 귀납적 정의로 표시하는 방법
 - 점화 관계가 $a_n = a_{n-1} + 3$ 일 경우,
 - ✓ a_5 를 알기 위해서는 $a_4 + 3$ 을 구해야 하므로, a_4 를 알아야 함
 - ✓ 또한, a_4 를 알려면 a_3 을 알아야 함
 - 이처럼, 앞의 항을 알면 다음 항을 구할 수 있는 경우 첫 번째 항을 포함한 몇 개의 항은 따로 주는 것이 일반적임
 - 점화식으로 표현되는 가장 유명한 수열 : 피보나치 수열

수열의 표시 방법

- 수열(Sequence)의 표시 방법
 - 등차수열 : 인접한 항 사이의 간격이 일정한 수열
 - 일정한 간격 : 공차
 - 공차가 d 인 수열은 귀납적으로 $a_n = a_{n-1} + d$ 라는 점화식으로 정의
 - 첫 번째 항이 a_1 인 등차수열의 일반항은 $a_n = a_1 + (n - 1)d$
 - 다음 수열에서 x 에 들어갈 값은? $1, 3, 5, 7, 9, x$
 - 일반항을 활용하면?

수열의 표시 방법

- 수열(Sequence)의 표시 방법

- 등차수열 : 인접한 항 사이의 간격이 일정한 수열

- 일정한 간격 : 공차
 - 공차가 d 인 수열은 귀납적으로 $a_n = a_{n-1} + d$ 라는 점화식으로 정의
 - 첫 번째 항이 a_1 인 등차수열의 일반항은 $a_n = a_1 + (n - 1)d$
 - 다음 수열에서 x 에 들어갈 값은? $1, 3, 5, 7, 9, x$
 - 일반항 : $a_n = 1 + (n - 1)2 = 2n - 1$
 - $x = a_6 = 2 \times 6 - 1 = 11$

수열의 표시 방법

- 수열(Sequence)의 표시 방법
 - 등비수열 : 앞의 항에 일정한 수를 곱해서 다음 항이 얻어지는 수열
 - 일정한 수 : 공비
 - 공비가 r 인 수열은 귀납적으로 $a_n = r \cdot a_{n-1}$ 이라는 점화식으로 정의
 - 첫 번째 항이 a_1 인 등비수열의 일반항은 $a_n = a_1 \cdot r^{n-1}$
 - 다음 수열에서 x 에 들어갈 값은? 2, 6, 18, 54, 162, x
 - 일반항을 활용하면?

수열의 표시 방법

- 수열(Sequence)의 표시 방법
 - 등비수열 : 앞의 항에 일정한 수를 곱해서 다음 항이 얻어지는 수열
 - 일정한 수 : 공비
 - 공비가 r 인 수열은 귀납적으로 $a_n = r \cdot a_{n-1}$ 이라는 점화식으로 정의
 - 첫 번째 항이 a_1 인 등비수열의 일반항은 $a_n = a_1 \cdot r^{n-1}$
 - 다음 수열에서 x 에 들어갈 값은? $2, 6, 18, 54, 162, x$
 - 일반항 : $a_n = 2 \times 3^{n-1}$
 - $x = a_6 = 2 \times 3^5 = 486$

수열의 표시 방법

- 수열(Sequence)의 표시 방법
 - 계차수열 : 그 수열의 인접하는 두 항의 차로 이루어진 수열
 - 계차(Difference) : 수열에서 항과 그 바로 앞의 항의 차
 - 이 계차들로 이루어진 수열을 그 수열의 계차수열이라 함
 - 예시 : 2, 4, 7, 11, 16, ...
 - 점화 수열 : 점화 관계를 가진 수열
 - 점화 수열의 대표적인 문제인 피보나치(G. Fibonacci)의 토끼 문제
 - 이 문제는 1202년 르네상스 이전의 유럽 수학자로는 당대 최고의 학자인 이탈리아 수학자인 피보나치가 “산반서”라는 책에서 처음 제기한 피보나치의 토끼 문제로 생물학적인 문제라기보다는 정수론의 연습문제로 제기된 것

수열의 표시 방법

- 수열(Sequence)의 표시 방법
 - 피보나치(G. Fibonacci)의 토끼 문제

1년 뒤 토끼는 모두 몇 쌍?

첫 달에는 새로 태어난 토끼 한 쌍만이 존재한다.

두 달 이상이 된 토끼는 번식 가능하다.

번식 가능한 토끼 한 쌍은 매달 암수 토끼 한 쌍을 낳는다.

토끼는 죽지 않는다.

일 년 뒤 토끼는 모두 몇 쌍이 될까?

-<산반서> 3부 중

수열의 표시 방법

- 수열(Sequence)의 표시 방법
 - 피보나치(G. Fibonacci)의 토끼 문제

1년 뒤 토끼는 모두 몇 쌍?

첫 달에는 새로 태어난 토끼 한 쌍만이 존재한다.

두 달 이상이 된 토끼는 번식 가능하다.

번식 가능한 토끼 한 쌍은 매달 암수 토끼 한 쌍을 낳는다.

토끼는 죽지 않는다.

일 년 뒤 토끼는 모두 몇 쌍이 될까?

개월	처음	1개월 후	2개월 후	3개월 후	4개월 후	5개월 후	...
토끼							...
							
							
쌍	1쌍	1쌍	2쌍	3쌍	5쌍	8쌍	...



점화 관계

- **점화 관계**

[정의] 점화 관계

점화 관계 recurrence relation 혹은 점화식 recurrence formula은 수열 a_0, a_1, \dots, a_n 에서 a_n 과 그 앞의 항들인 $a_0, a_1, a_2, \dots, a_{n-1}$ 과의 종속 관계를 말한다. 즉, 수열에서 n 번째 원소와 그 앞의 원소와의 관계를 나타낸다.

- 점화 관계에서 수열을 정의하기 위해서는 초기값이 주어져야 함
 - 초기값 : 초기 조건(Initial condition)

점화 관계

- 점화 관계 (예제)

5, 8, 11, 14, 17, ...

- 해당 수열의 일반항 표현
 - $a_1 = 5$
 - $a_n = a_{n-1} + 3, n \geq 2$
- 즉, 방정식 $a_n = a_{n-1} + 3$ 은 점화 관계가 됨
 - n 번째 항의 값이 $(n - 1)$ 번째 항의 값에 의해 결정되는 것으로 a_n 의 값은 a_{n-1} 에 종속
- 이와 같이 a_n 이 바로 전 항인 a_{n-1} 에 의존하면 1차(First-order)라고 하고,
방정식 $a_n = a_{n-1} + 3$ 을 1차 선형 점화 관계라고 함

점화 관계

예제

점화 수열의 대표적인 문제로 알려진 피보나치의 토끼 번식 문제를 생각해보자. 한 쌍의 토끼가 있다. 암컷 토끼는 두 달이 지날 때부터 시작하여 매달 한 쌍의 토끼를 낳는다고 한다. 또 새로 태어난 암컷 토끼도 두 달이 지난 후부터 시작하여 매달 한 쌍의 토끼를 낳는다고 가정할 때, 1년이 지난 후 전체 토끼의 쌍의 수를 구하라. 단, 모든 토끼들은 계속 살아 있다고 가정한다.



점화 관계

예제

점화 수열의 대표적인 문제로 알려진 피보나치의 토끼 번식 문제를 생각해보자. 한 쌍의 토끼가 있다. 암컷 토끼는 두 달이 지날 때부터 시작하여 매달 한 쌍의 토끼를 낳는다고 한다. 또 새로 태어난 암컷 토끼도 두 달이 지난 후부터 시작하여 매달 한 쌍의 토끼를 낳는다고 가정할 때, 1년이 지난 후 전체 토끼의 쌍의 수를 구하라. 단, 모든 토끼들은 계속 살아 있다고 가정한다.

개월	처음	1개월	2개월	3개월	4개월	5개월	...
어미 토끼(쌍)	1	1	1	2	3	5	...
새끼 토끼(쌍)	-	-	1	1	2	3	...
전체 쌍의 수	1	1	2	3	5	8	...

$R(n)$ 을 n 달이 지난 후의 전체 토끼의 쌍의 수를 나타낸다고 하자. 그러면

$$R(12) = R(11) + R(10)$$

$$R(n) = R(n-1) + R(n-2)$$

$$\begin{aligned} &= 144 + 89 \\ &= 233(\text{쌍}) \end{aligned}$$

점화 관계

- **피보나치 수열**($R(n) = R(n - 1) + R(n - 2)$)
 - n 번째($n \geq 2$) 피보나치 숫자를 계산하기 위해 이전의 두 피보나치 숫자인 ($n - 1$)번째 피보나치 숫자와 ($n - 2$)번째 피보나치 숫자를 더해서 계산
 - 이러한 종류의 정의 : 재귀적 정의(recursive definition)
 - 원하는 값이 이전에 계산된 값을 통해 계산
 - 피보나치 수열은 자연에서 예상치 못하게 나타나는데, 이는 자연계의 법칙을 담고 있으며, 이는 인간이 눈으로 보기 가장 좋은 비율인 황금 비율(미의 기준)과 연관이 있음
 - 황금비율은 $1 : 1.618$ 으로써 피보나치 수열에서 n 항과 $n + 1$ 항의 비율로 $1 : 1.618$ 이 됨

✓ (전체 길이) : 긴 선분 길이 = (긴 선분 길이) : (짧은 선분 길이)

$$\Rightarrow (1 + x) : x = x : 1$$

$$\Rightarrow x^2 - x - 1 = 0$$

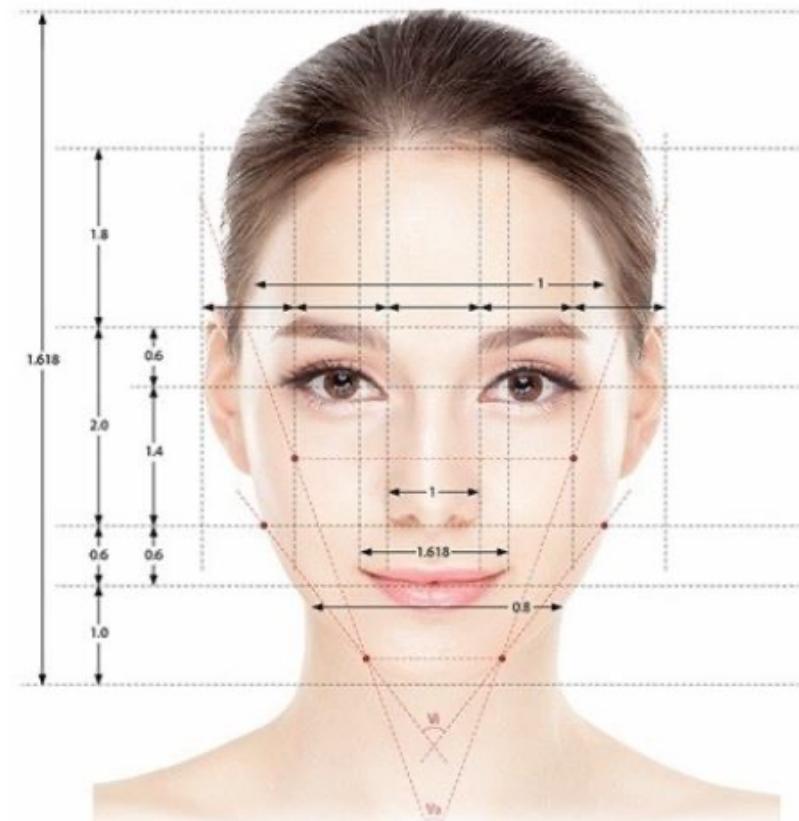
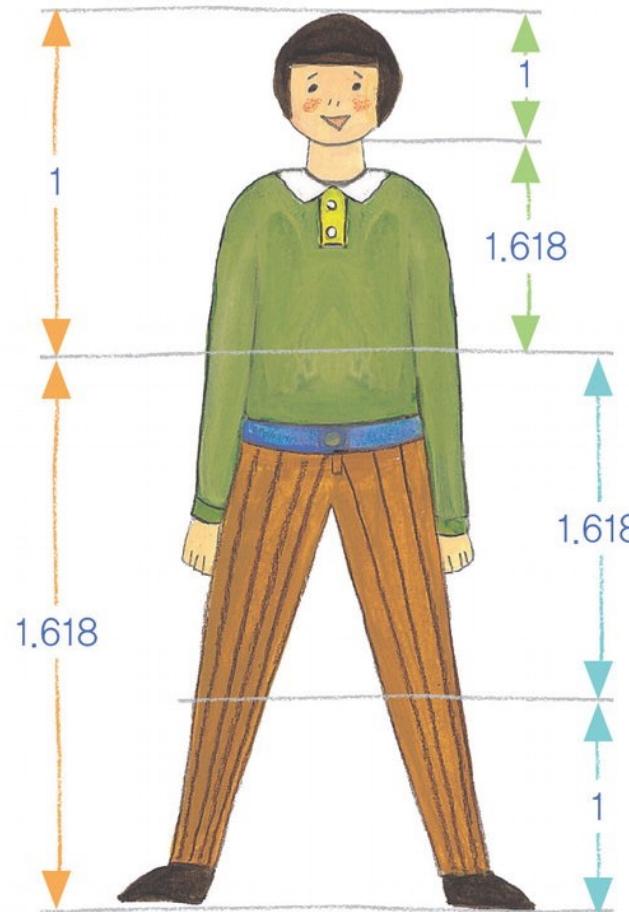
$$\Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{-1 \pm \sqrt{1+4}}{2} \Rightarrow \frac{-1+\sqrt{5}}{2} \approx 1.618$$

점화 관계

- 미인들의 얼굴 : 성형외과 의사(스티븐 마쿠어드)가 황금 마스크 만듦
- 영화 속 : 다빈치 코드 (암호로 숫자들이 나타나는데 정리하면 피보나치 수열)
- 그림 : 보티첼리의 “비너스의 탄생”
 - 레오나르도 다빈치는
대부분 황금비를 이용해서 작품을 만듦
(배꼽기준으로 상체, 하체 길이비, 허벅지,
종아리 등 이상적인 신체 비율)
“인체의 비례론”
- 식물 : 해바라기 씨앗의 나선모양, 솔잎 - 비바람에 잘 견딤
- 건축 : 파르테논 신전, 피라미드, 영주 부석사 무량수전
- 생활 : 신용카드, 명함, TV의 가로 세로 길이가 황금비율로 만들어짐



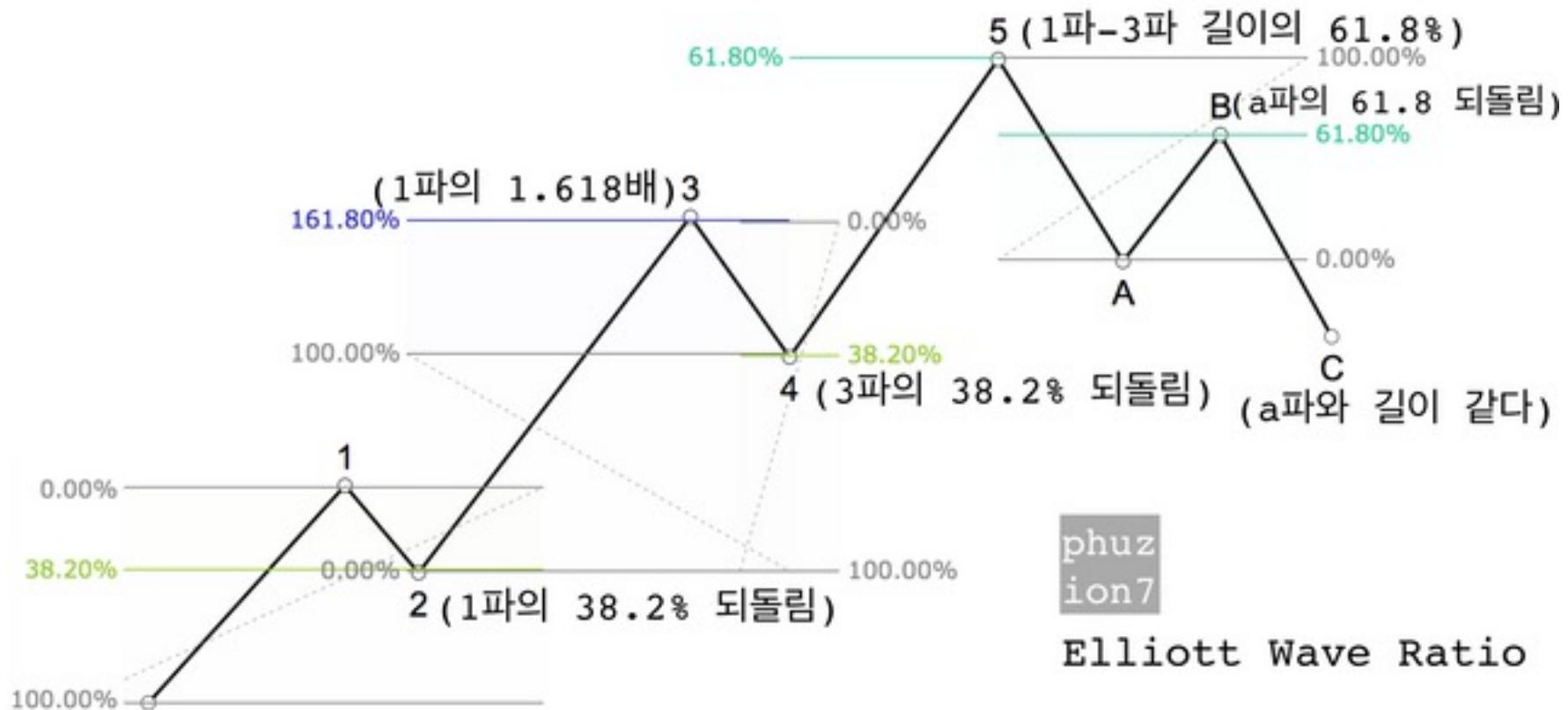
점화 관계



점화 관계



엘리어트 파동이론의 조정비율



점화 관계의 응용 : 하노이 탑 문제

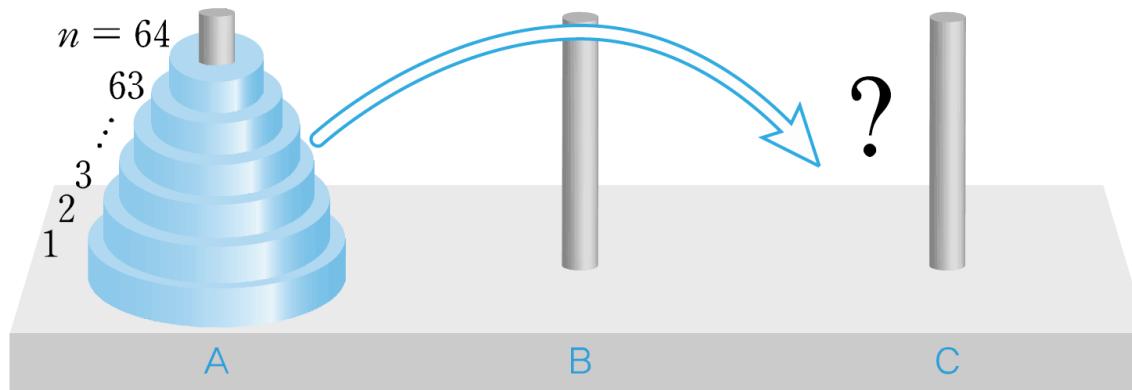
- 점화 관계의 응용 : 하노이 탑 문제
 - 이산수학의 전형적인 문제
 - 1883년 프랑스 수학자 루카스(E. Lucas)가 제시한 하노이 탑 문제(Hanoi Tower Problem)
 - 재귀 호출을 이용해서 풀 수 있는 가장 유명한 예제
 - 베트남(Vietnam)의 하노이(Hanoi)시 외곽에 있는 베나레스(Benares) 사원의 한가운데 있는 돔(Dome)에 동판이 있다. 동판에 다이아몬드 막대가 세 개 있고, 크기가 서로 다른 64개의 황금 원판이 한 막대에 꽂혀 있음
 - 이러한 황금 원판을 규칙에 따라 다른 막대로 모두 옮기는 문제

점화 관계의 응용 : 하노이 탑 문제

예제

다음 그림과 같이 하노이 탑에 황금 원판이 동판에 꽂혀 있다. 황금 원판을 옮기는 규칙이 다음과 같을 때 원판을 모두 다른 막대로 옮기는 데 필요한 이동^{move} 횟수를 구하라.

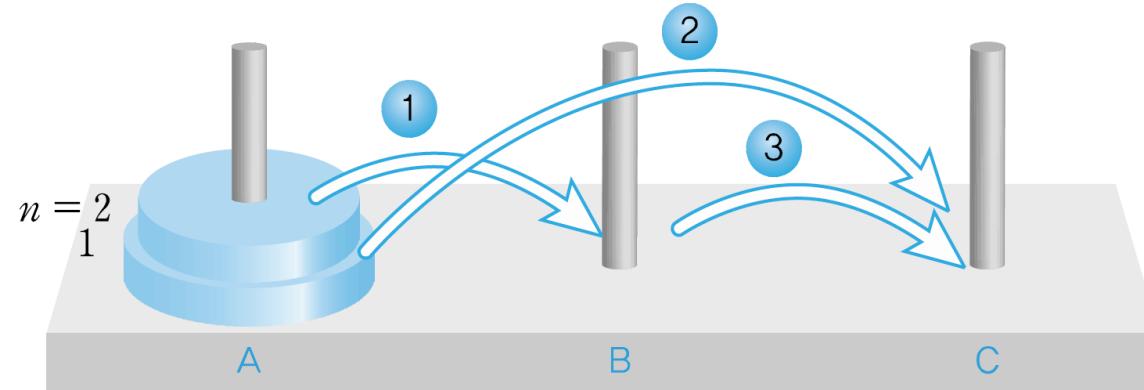
- 한 번에 1개의 황금 원판만 옮긴다.
- 크기가 큰 황금 원판은 반드시 크기가 작은 황금 원판 아래쪽에 있어야 한다.



점화 관계의 응용 : 하노이 탑 문제

예제

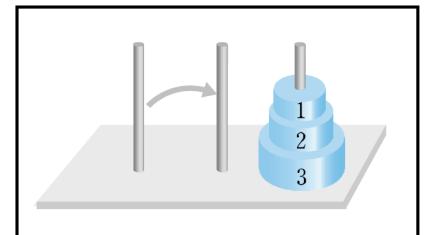
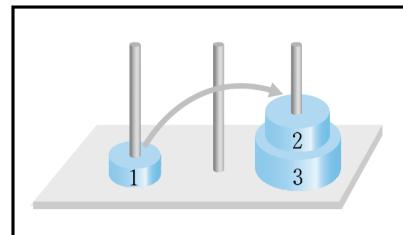
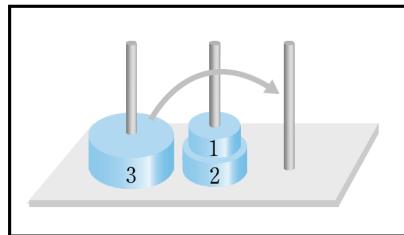
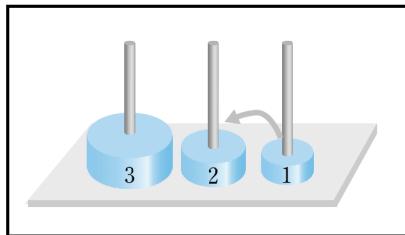
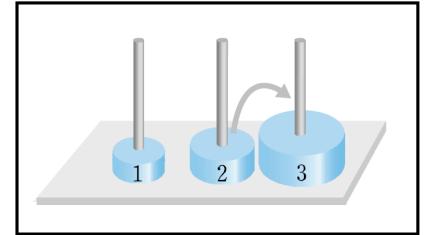
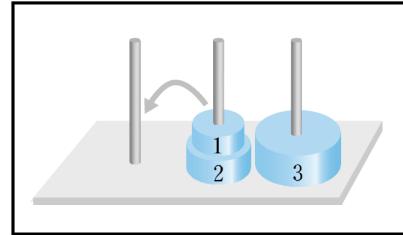
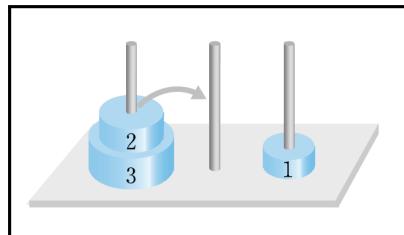
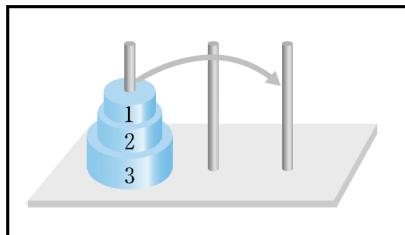
다음 그림과 같이 하노이 탑에 황금 원판이 동판에 꽂혀 있다. 황금 원판을 옮기는 규칙이 다음과 같을 때 원판을 모두 다른 막대로 옮기는 데 필요한 이동^{move} 횟수를 구하라.



점화 관계의 응용 : 하노이 탑 문제

예제

다음 그림과 같이 하노이 탑에 황금 원판이 동판에 꽂혀 있다. 황금 원판을 옮기는 규칙이 다음과 같을 때 원판을 모두 다른 막대로 옮기는 데 필요한 이동^{move} 횟수를 구하라.



점화 관계의 응용 : 하노이 탑 문제

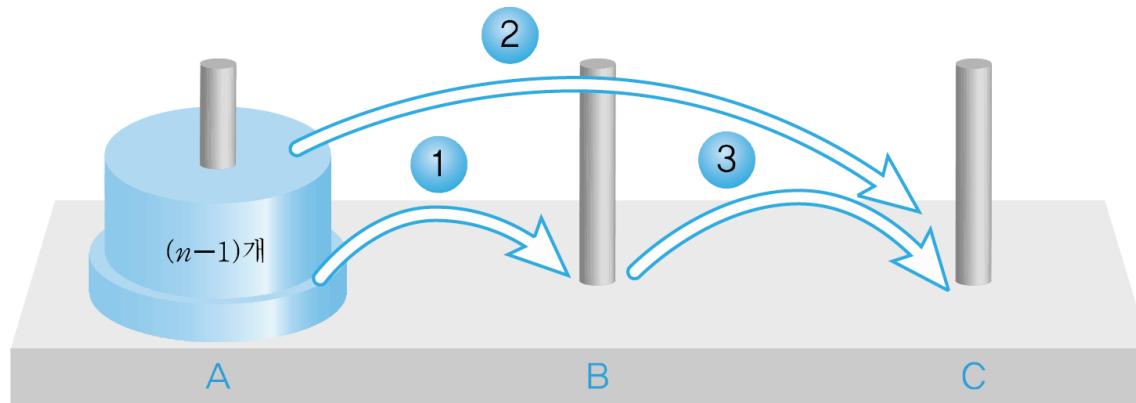
이제 일반적으로 n 개의 원판이 있을 때 필요한 원판의 이동 횟수를 알아보도록 하자. 이 경우의 풀이 전략은 다음과 같다. 먼저 n 개의 원판인 있을 때 모두 옮기는 데 필요한 원판의 이동 횟수를 $P(n)$ 이라고 하자. 그러면

$$P(1) = 1$$

$$P(2) = 3$$

$$P(3) = 7$$

임을 알 수 있다. 전체적으로 다음 그림과 같이 분할 정복 방법 *divide and conquer*을 이용한다.

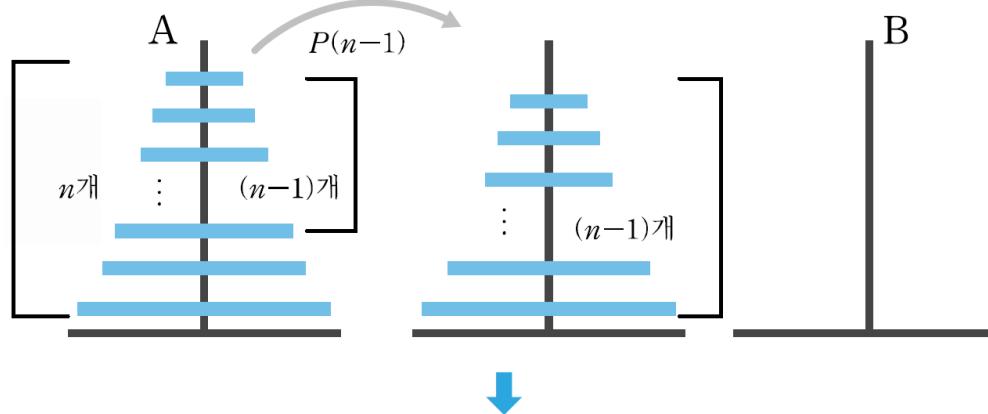


점화 관계의 응용 : 하노이 탑 문제

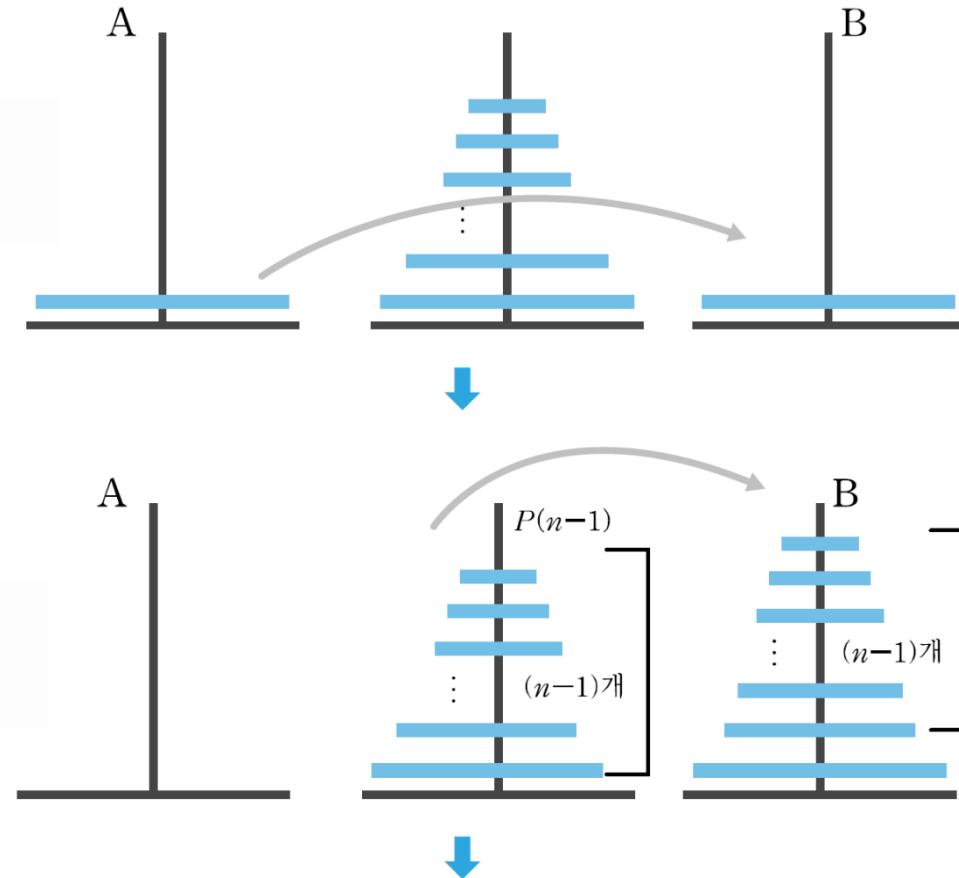
즉, n 개의 원반을 다음과 같이 크게 두 부분으로 나눈다.

- 한 부분 : 제일 아래쪽에 가장 큰 원반(n 번) 하나만으로 이루어진 부분
- 나머지 부분 : 위 부분을 제외한 1번부터 $(n-1)$ 번 원반으로 구성된 부분

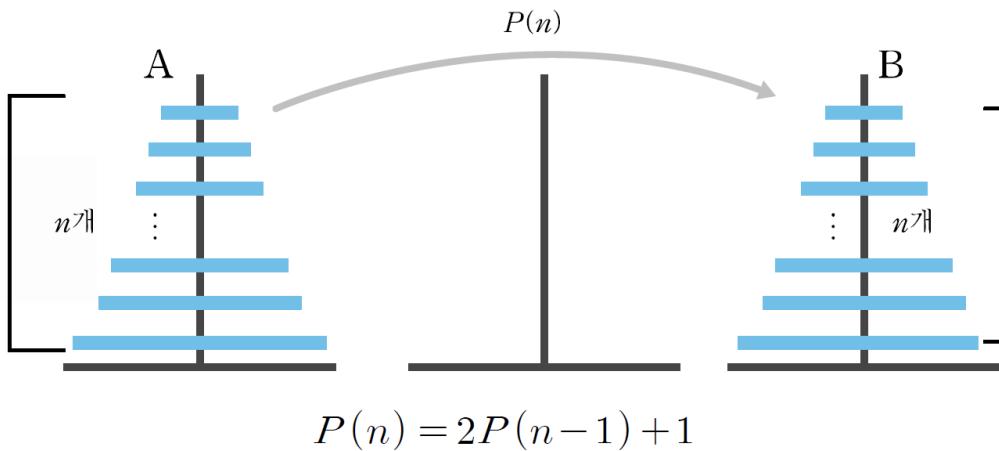
그리고 이 두 부분을 마치 원반이 2개인 것처럼 다음 그림과 같은 순서로 해결한다.



점화 관계의 응용 : 하노이 탑 문제



점화 관계의 응용 : 하노이 탑 문제



일반적으로 $P(n)$ 은 $P(n-1)$ 로부터 유도할 수 있다. 그림을 보면, 모든 자연수 n 에 대해서 관계식 $P(n) = 2P(n-1) + 1$ 을 추측할 수 있다, 그러면 앞의 점화 관계에 차례로 대입하여 알아보면

$$P(1) = 1$$

$$P(2) = 2P(1) + 1 = 2 + 1 = 3$$

$$P(3) = 2P(2) + 1 = 2 \times 3 + 1 = 4 + 2 + 1 = 7$$

$$P(4) = 2P(3) + 1 = 2 \times 7 + 1 = 8 + 4 + 2 + 1 = 15$$

⋮

점화 관계의 응용 : 하노이 탑 문제

이 된다. 등비수열의 합 공식을 이용하면

$$P(n) = 1 + 2 + 2^2 + \cdots + 2^{n-1} = \frac{2^n - 1}{2 - 1} = 2^n - 1 \quad (1 \text{ 장에서 } \text{다룬 } \text{메르센 } \text{소수})$$

이다. 따라서 원래의 하노이 탑 문제의 경우는 $n = 64$ 인 경우이므로 필요한 원판의 이동 총수(이것은 가장 최소의 이동 횟수를 의미하며, 만약 이동 순서를 잘 못하면 그만큼 이동 횟수는 늘어난다)는 다음과 같다.

$$P(64) = 2^{64} - 1 = 18,446,744,073,709,551,615$$

알고리즘의 개념과 표현

- 알고리즘
 - 문제 해결을 위한 체계적인 명령의 나열

[정의] 알고리즘

알고리즘 algorithm이란 어떤 업무 및 계산을 하는 데 필요한 전 과정을 순차적으로 나열한 것이다. 즉, 주어진 문제를 풀기 위한 일련의 조작법으로서 다음과 같은 다섯 가지의 특징이 있다.

- (1) 입력 *input* : 문제를 풀기 위한 입력이 있어야 한다.
- (2) 출력 *output* : 문제를 해결했을 때 그 답이 있어야 한다.
- (3) 유한성 *finiteness* : 유한 번 조작 후에 반드시 끝내야 한다.
- (4) 명확성 *definiteness* : 일의적一義的으로 규정되어 있어야 한다.
- (5) 효율성 *effectiveness* : 실제적이고 정확해야 한다.

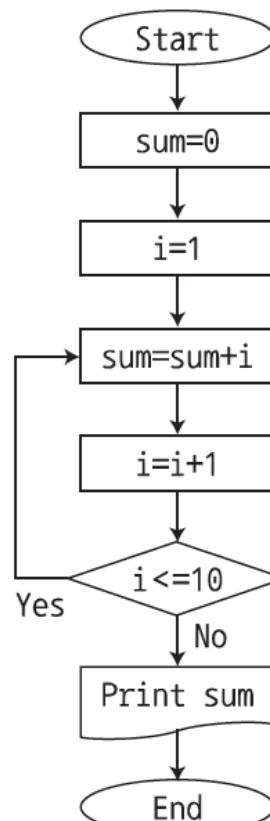
알고리즘의 개념과 표현

- 알고리즘 표현 방법
 - 순서적인 도표를 사용해서 표현하는 순서도(Flow chart)
 - 실제적인 언어를 닮은 의사 코드(Pseudo code)

기호	설명	기호	설명
작업의 시작/종료	순서도의 시작과 끝	입/출력	Data 의 입력 / 출력
준비	기억장소, 초기값 등 준비과정	출력	Data 의 출력
처리	Data 의 이동, 연산, 변환	수동입력	Data 의 수동 입력
미리 정의된 처리	미리 정의된 처리과정	→	작업의 흐름과 연결 표시
판단	비교 판단에 의한 분기	연결자	흐름이 다른 곳으로 연결되는 출입구

알고리즘의 개념과 표현

- 알고리즘 표현 방법
 - 순서도/의사코드

순서도	의사코드
	<pre>1 algorithm sum1to10(){ 2 sum=0; 3 i=1; 4 while i<=10 5 sum = sum + i; 6 i=i+1; 7 endwhile 8 print sum; 9 }</pre>

알고리즘의 개념과 표현

예제

다음 알고리즘의 내용을 설명하고 결과를 알아내라.

(a)

```
1 algorithm A( )
2   i = 0;
3   while i <= 20
4     x = 1;
5     x = x + 1;
6     if i < 20 then
7       x = 10;
8     else
9       x = 0;
10    endif
11    i = i+2;
12  endwhile
13  print i, x;
14 }
```

(b)

```
1 algorithm B( )
2   i = 1;
3   x = 0;
4   while i <= n
5     x = x + 1;
6     i = i + 1;
7   endwhile
8   print i, x;
9 }
```

(c)

```
1 algorithm C(a, b, c){
2   x = a;
3   if b > x then
4     x = b;
5   endif
6   if c > x then
7     x = c;
8   endif
9   print x;
10 }
```



알고리즘의 개념과 표현

예제

다음 알고리즘의 내용을 설명하고 결과를 알아내라.

(a)

```
1 algorithm A( )
2   i = 0;
3   while i <= 20
4     x = 1;
5     x = x + 1;
6     if i < 20 then
7       x = 10;
8     else
9       x = 0;
10    endif
11    i = i+2;
12  endwhile
13  print i, x;
14 }
```

(b)

```
1 algorithm B( )
2   i = 1;
3   x = 0;
4   while i <= n
5     x = x + 1;
6     i = i + 1;
7   endwhile
8   print i, x;
9 }
```

(c)

```
1 algorithm C(a, b, c){
2   x = a;
3   if b > x then
4     x = b;
5   endif
6   if c > x then
7     x = c;
8   endif
9   print x;
10 }
```

$i = 22, x = 0$

논리적 명령 실행 불가

a, b, c 중 가장 큰 값 출력

알고리즘의 복잡도

- **복잡도**

- 알고리즘 수행 시 필요한 시간 또는 공간에 대한 비용

- 시간 복잡도(Time complexity) : 프로그램이 수행되는 시간
 - 공간 복잡도(Space complexity) : 프로그램이 차지하는 기억 공간
 - 최상 : 1회 실행으로 결과 출력
 - 최악 : 최대 횟수 실행으로 결과 출력
 - 평균 : 평균 횟수 실행으로 결과 출력

- **Big-O 표기법** : $f(n) = O(g(n))$

- 함수 $f(n)$ 과 $g(n)$ 이 $n > k(k \geq 0)$ 일 때 항상 $|f(n)| \leq c \cdot |g(n)|(c > 0)$ 을 만족하는 c 와 k 가 존재할 때의 표기

알고리즘의 복잡도

- **시간 복잡도**

- $O(1)$: 상수 복잡도
- $O(\log n)$: 로그 복잡도
- $O(n)$: 선형 복잡도
- $O(n \log n)$: $n \log n$ 복잡도
- $O(n^2)$: 2차 복잡도
- $O(n^3)$: 3차 복잡도
- $O(2^n)$: 지수 복잡도

알고리즘의 복잡도

- **Big-O 표기법** : $f(n) = O(g(n))$
 - Input size n 에 따른 알고리즘의 시간 복잡도

알고리즘의 복잡도

- **Big-O 표기법** : $f(n) = O(g(n))$
- **Input size n** 에 따른 알고리즘의 시간 복잡도

```
list_input = list(range(0, 10))

def print_first(list_input):
    print(list_input[0])
```

알고리즘의 복잡도

- **Big-O 표기법** : $f(n) = O(g(n))$
- **Input size n** 에 따른 알고리즘의 시간 복잡도

```
list_input = list(range(0, 1000))

def print_first(list_input):
    print(list_input[0])
```

알고리즘의 복잡도

- **Big-O 표기법** : $f(n) = O(g(n))$
- **Input size** n 에 따른 알고리즘의 시간 복잡도

```
list_input = list(range(0, 1000))

def print_first(list_input):
    print(list_input[0])
    print(list_input[1])
```

- **Big-O 표기법은 input size**에 따라서 대략적으로 어떻게 함수가 작동하는지에 관심이 있음
 - Input size에 따른 시간 그래프 형태의 변화 여부
 - Input size에 따라 시간이 어떻게 변화하는지를 나타냄

알고리즘의 복잡도

- **Big-O 표기법** : $f(n) = O(g(n))$
- **Input size n** 에 따른 알고리즘의 시간 복잡도

```
list_input = list(range(0, 1000))

def print_all(list_input):
    for n in list_input:
        print n
```

알고리즘의 복잡도

- **Big-O 표기법 :** $f(n) = O(g(n))$
- **Input size n 에 따른 알고리즘의 시간 복잡도**

```
list_input = list(range(0, 1000))

def print_all(list_input):
    for n in list_input:
        print n

    for n in list_input:
        print n+1
```

알고리즘의 복잡도

- **Big-O 표기법 :** $f(n) = O(g(n))$
- **Input size n 에 따른 알고리즘의 시간 복잡도**

```
list_input = list(range(0, 1000))

def print_all(list_input):
    for n in list_input:
        print n

    for n in list_input:
        print n+1

    print(list_input[0])
```

알고리즘의 복잡도

- **Big-O 표기법** : $f(n) = O(g(n))$
- Big-O 표기법은 input size에 따라서 대략적으로 어떻게 함수가 작동하는지에 관심이 있음
 - 항의 계수는 무시하고 표기
 - 영향력 없는 항은 무시하고 표기
- 예, $O(2n^2 + 3n - 5) \rightarrow O(n^2)$

알고리즘의 복잡도

예제

다음 함수의 복잡도를 구하라.

- (a) $f(n) = n$ (b) $f(n) = 10n^2 - 3n + 2$ (c) $f(n) = 10^{10}\log_{10}n$ (d) $f(n) = n!$

알고리즘의 복잡도

예제

다음 함수의 복잡도를 구하라.

(a) $f(n) = n$ (b) $f(n) = 10n^2 - 3n + 2$ (c) $f(n) = 10^{10}\log_{10}n$ (d) $f(n) = n!$

(a) $n \geq 0$ 일 때 $f(n) = n \leq n^0$ 성립하므로 $c = 1, k = 0$ 이 존재한다.

$\therefore f(n)$ 의 복잡도는 $O(n)$ 이다.

(b) $n \geq 1$ 일 때 $f(n) = 10n^2 - 3n + 2 \leq 10n^2 + 3n^2 + 2n^2 = 15n^2$ 이 성립하므로 $c = 15, k = 1$ 이 존재한다.

$\therefore f(n)$ 의 볍잡도는 $O(n^2)$ 이다.

(c) $n \geq 1$ 일 때 $f(n) = 10^{10}\log_{10}n \leq 10^{10}\log_{10}n$ 이 성립하므로 $c = 10^{10}, k = 1$ 이 존재한다.

$\therefore f(n)$ 의 볍잡도는 $O(\log_{10}n)$ 이다.

(d) $n \geq 1$ 일 때 $f(n) = n! = 1 \cdot 2 \cdot \dots \cdot (n-1) \cdot n \leq n \cdot n \cdot \dots \cdot n = n^n$ 이 성립하므로 $c = 1, k = 1$ 이 존재한다.

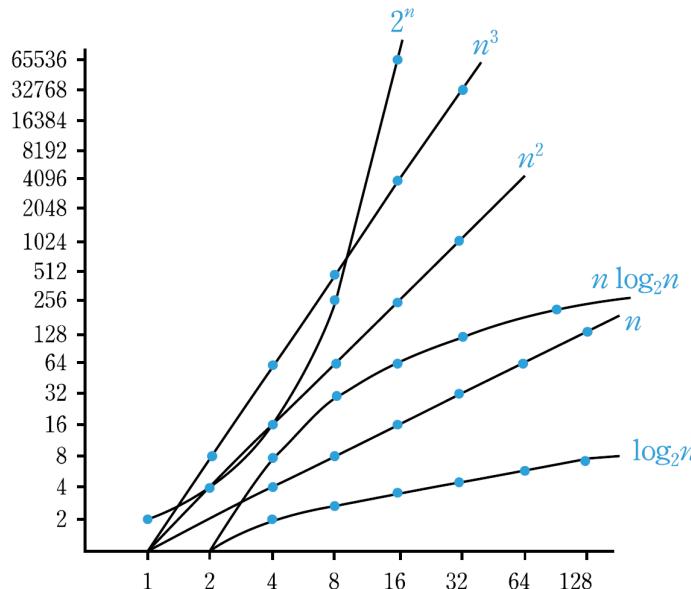
$\therefore f(n)$ 의 볍잡도는 $O(n^n)$ 이다.

알고리즘의 복잡도

- 복잡도 간 관계

$$O(1) < O(\log_2 n) < O(n) < O(n \log_2 n) < O(n^2) < O(n^3) < O(2^n)$$

- 실제로 $O(n^2)$ 이상 되는 알고리즘은 n 값이 아주 커짐에 따라 수행 시간이 너무 증가하므로 거의 수행 불가능하다고 할 수 있으며, 특히 $O(2^n)$ 알고리즘은 n 값이 조금만 커져도 컴퓨터 수행이 거의 불가능해짐



$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	4294967296

알고리즘의 복잡도

- 알고리즘 복잡도 예시

```
1 algorithm(int n){  
2     x = 0;  
3     for i=0 to n  
4         x += 1;  
5     next i  
6 }
```

- $n \geq 1$ 일 때, $f(n) = n + 1 \leq n + n = 2n$
- ∴ $c = 2, k = 1$ 일 때, 알고리즘의 복잡도 $O(n)$

알고리즘의 복잡도

예제

다음과 같은 알고리즘이 주어졌을 때, 알고리즘의 복잡도를 구하라.

(a)

```
1 algorithm A( ){  
2     i = 0;  
3     print i;  
4 }
```

(b)

```
1 algorithm B(int n){  
2     x = 0;  
3     for i=0 to n  
4         x = (x + (3 * i)) / 10;  
5     next i  
6 }
```

(c)

```
1 algorithm C(int n){  
2     x = 0;  
3     for i=0 to n  
4         for j=0 to n  
5             x += 1;  
6         next j  
7     next i  
8 }
```



알고리즘의 복잡도

- (a) 실행 횟수를 고려할 수 있는 실행문은 3 번 라인의 `print i;` 뿐이므로 1회 실행된다. 그러므로 실행 횟수를 함수로 표현하면 $f(n) = 1$ 로 표현된다.
 $\therefore f(n)$ 의 복잡도는 $O(1)$ 이다.
- (b) 실행 횟수를 고려할 수 있는 실행문은 4 번 라인의 `x = (x + (3 * i)) / 10;`으로, 이 연산식 안에는 `+`, `*`, `/`의 3 가지 연산이 포함되므로 3회의 실행을 한다. 그런데 4 번 라인은 0부터 n 까지의 반복문 안에 포함되므로 각 연산은 모두 $n+1$ 회씩 실행된다. 그러므로 이 알고리즘의 실행 횟수를 함수로 표현하면 $f(n) = 3(n+1)$ 이다. 이 함수는 $n \geq 1$ 일 때 $f(n) = 3n + 3 \leq 3n + 3n = 6n$ 으로 $c = 6$, $k = 1$ 이 존재한다.
 $\therefore f(n)$ 의 복잡도는 $O(n)$ 이다.
- (c) 실행 횟수를 고려할 수 있는 실행문은 5 번 라인의 `x += 1;`인데 이 실행문은 중첩 루프 안에 포함된 실행문이다. 변수 j 에 대한 루프는 0부터 n 까지 $n+1$ 회 반복되는데, 이 루프는 변수 i 에 대한 루프 안에 포함되는 루프로 변수 i 에 대한 루프가 1회 실행될 때 변수 j 에 대한 루프가 $n+1$ 회 실행된다. 그러므로 이 루프들 안에 포함되는 5 번 라인은 $(n+1)^2$ 만큼 실행될 것이다. 이 실행 횟수를 함수로 표현하면 $f(n) = (n+1)^2$ 이다. 이 함수는 $n \geq 1$ 일 때 $f(n) = (n+1)^2 = n^2 + 2n + 1 \leq n^2 + 2n^2 + n^2 = 4n^2$ 으로 $c = 4$, $k = 1$ 이 존재한다.
 $\therefore f(n)$ 의 복잡도는 $O(n^2)$ 이다.

유클리드 알고리즘

- 유클리드 알고리즘(Euclidean algorithm, $GCD(a, b)$)
 - 두 양의 정수 a, b 의 공약수 중 최대 공약수를 찾는 알고리즘
 - 최대공약수에 대한 정리
 - 양의 정수 $a, b (a > b)$ 에 대해 $r = a \bmod b (0 \leq r < b)$ 가 성립한다면 다음이 성립함

$$GCD(a, b) = GCD(b, r)$$

유클리드 알고리즘

예제

다음 두 수의 최대공약수를 구하라.

(a) 1020, 816

(b) 927, 1125

(c) 2592, 3870

유클리드 알고리즘

$$\begin{aligned}(a) \quad GCD(1020, 816) &= GCD(816, 204) && (\because 1020 \bmod 816 = 204) \\ &= GCD(204, 0) && (\because 816 \bmod 204 = 0)\end{aligned}$$

\therefore 1020과 816의 최대공약수는 204이다.

$$\begin{aligned}(b) \quad GCD(927, 1125) &= GCD(1125, 927) \\ &= GCD(927, 198) && (\because 1125 \bmod 927 = 198) \\ &= GCD(198, 135) && (\because 927 \bmod 198 = 135) \\ &= GCD(135, 63) && (\because 198 \bmod 135 = 63) \\ &= GCD(63, 9) && (\because 135 \bmod 63 = 9) \\ &= GCD(9, 0) && (\because 63 \bmod 9 = 0)\end{aligned}$$

\therefore 927과 1125의 최대공약수는 9이다.

$$\begin{aligned}(c) \quad GCD(2592, 3870) &= GCD(3870, 2592) \\ &= GCD(2592, 1278) && (\because 3870 \bmod 2592 = 1278) \\ &= GCD(1278, 36) && (\because 2592 \bmod 1278 = 36) \\ &= GCD(36, 18) && (\because 1278 \bmod 36 = 18) \\ &= GCD(18, 0) && (\because 36 \bmod 18 = 0)\end{aligned}$$

\therefore 2592와 3870의 최대공약수는 18이다.

유클리드 알고리즘

- 유클리드 알고리즘

```
1 algorithm gcd(a, b){  
2     if a < b then  
3         tmp = a;  
4         a = b;  
5         b = tmp;  
6     endif  
7     while b ≠ 0  
8         r = a mod b;  
9         a = b;  
10        b = r;  
11    endwhile  
12    return a;  
13 }
```

$$f(n) = 5(\log_{10} b + 1) + 4 = 5\log_{10} b + 9$$

∴ 알고리즘 복잡도 $O(\log_{10} b)$

재귀 알고리즘

- 재귀 알고리즘
 - 어떤 함수가 자기 자신을 다시 호출해 사용하는 알고리즘

```
1 algorithm factorial(n){  
2     if n <= 1 then  
3         return 1;  
4     else  
5         return n * factorial(n-1);  
6     end if  
7 }
```

$$f(n) = n$$

\therefore 알고리즘 복잡도 $O(n)$

재귀 알고리즘

예제

다음 함수들의 재귀 알고리즘을 작성하고 복잡도를 구하라.

$$(a) \begin{cases} f(0) = 1 \\ f(n) = 1 + f(n-1), \quad n \geq 1 \end{cases}$$

$$(b) \begin{cases} f(1) = 1 \\ f(2) = 1 \\ f(n) = f(n-1) + f(n-2), \quad n \geq 3 \end{cases}$$

재귀 알고리즘

(a) 함수를 전개하면 다음과 같다.

$$f(0) = 1$$

$$f(1) = 1 + f(0) = 1 + 1 = 2$$

$$f(2) = 1 + f(1) = 1 + \{1 + f(0)\} = 1 + 1 + 1 = 3$$

...

즉, 1을 $n+1$ 회 더하는 함수이다. 이를 재귀 알고리즘으로 작성하면 다음과 같다.

```
1 algorithm sum(n){  
2     if n = 0 then  
3         return 1;  
4     else  
5         return 1 + sum(n-1);  
6     endif  
7 }
```

- 3번 라인 : $n = 0$ 일 때, 1회 실행된다.
- 5번 라인 : $n > 0$ 일 동안 $\text{sum}(n)$ 이 n 회 호출되어 1과의 합을 실행한다. 즉, 총 $n+1$ 회 실행된다.

∴ 실행 횟수 함수는 $f(n) = n + 2$ 이고, 복잡도는 $O(n)$ 이다.

재귀 알고리즘

(b) 이 함수는 피보나치수열을 이용해 합을 구하는 함수이다. 이를 재귀 알고리즘으로 작성하면 다음과 같다.

```
1 algorithm fibonacci(n){  
2     if n = 1 or n = 2 then  
3         return 1;  
4     else  
5         return fibonacci(n-1) + fibonacci(n-2);  
6     endif  
7 }
```

- 3번 라인 : $n = 1, n = 2$ 일 때 각각 1회씩 호출된다.
- 5번 라인 : $n > 2$ 인 동안 $\text{fibonacci}(n)$ 이 $n-1$ 과 $n-2$ 에 대해 1회씩 호출되어 합을 실행한다.

이 피보나치 수열에 대한 재귀 알고리즘은 하나의 함수 결과를 얻기 위해 두 개의 재귀 함수를 호출한다.
 \therefore 실행 횟수 함수는 $f(n) = 2^n$ 이고, 복잡도는 $O(2^n)$ 이다.

탐색 알고리즘

- 탐색 알고리즘
 - 순차탐색 알고리즘
 - 이진탐색 알고리즘
 - 버블 정렬
 - 삽입 정렬
 - 퀵 정렬
 - 합병 정렬

탐색 알고리즘

- 순차탐색 알고리즘(Sequential search algorithm)
 - 원소 집합의 처음부터 하나씩 비교하며 탐색하는 알고리즘

```
1  algorithm seq_search(int arr[], int key, int n){  
2      int i;  
3      for i = 0 to n-1  
4          if arr[i] == key then  
5              return i;  
6          endif  
7      next i  
8      return -1;  
9  }
```

$$f(n) = n$$

∴ 알고리즘 복잡도 $O(n)$

탐색 알고리즘

- **순차탐색 알고리즘(Sequential search algorithm)**
 - 원소 집합의 처음부터 하나씩 비교하며 탐색하는 알고리즘
 - 예시, 원소 J의 인덱스 검색

인덱스	0	1	2	3	4	5	6	7	8	9
원소	H	E	A	I	G	J	B	F	C	D

- (1) 인덱스 0의 H와 J 비교 : $H \neq J$ \therefore 다음 인덱스 탐색
- (2) 인덱스 1의 E와 J 비교 : $E \neq J$ \therefore 다음 인덱스 탐색
- (3) 인덱스 2의 A와 J 비교 : $A \neq J$ \therefore 다음 인덱스 탐색
- (4) 인덱스 3의 I와 J 비교 : $I \neq J$ \therefore 다음 인덱스 탐색
- (5) 인덱스 4의 G와 J 비교 : $G \neq J$ \therefore 다음 인덱스 탐색
- (6) 인덱스 5의 J와 J 비교 : $J = J$ \therefore 탐색 종료

탐색 알고리즘

- 이진탐색 알고리즘(Binary search algorithm)

- 원소 집합을 반으로 나누어 기준을 정하고 그 기준과 특정 원소를 비교하는 방법을 반복하여 탐색 범위를 좁혀가며 원소를 탐색하는 알고리즘
- 기준을 정하는 규칙 : $\left\lfloor \frac{l+r}{2} \right\rfloor$ (l : 왼쪽 인덱스, r : 오른쪽 인덱스)

```
1  algorithm binary_search(int arr[], int key, int n){  
2      int left = 0;  
3      int right = n-1;  
4      int mid;  
5      while (left <= right)  
6          mid = (left + right) / 2;  
7          if arr[mid] < key then  
8              left = mid + 1;  
9          else if arr[mid] > key then  
10              right = mid - 1;  
11          else  
12              return mid;  
13          end if  
14      endwhile  
15      return -1;  
16  }
```

탐색 알고리즘

- 이진탐색 알고리즘(Binary search algorithm)

- 예시, 원소 J의 인덱스 검색

인덱스	0	1	2	3	4	5	6	7	8	9
원소	H	E	A	I	G	J	B	F	C	D

(1) 왼쪽 인덱스 l 은 0, 오른쪽 인덱스 r 은 9이므로 기준은 $\left\lfloor \frac{0+9}{2} \right\rfloor = 4$ 이다.

인덱스 4의 E는 J보다 사전적으로 앞에 오는 문자이다.

∴ 인덱스 5 이후의 원소들을 탐색한다.

(2) 왼쪽 인덱스 l 은 5, 오른쪽 인덱스 r 은 9이므로 기준은 $\left\lfloor \frac{5+9}{2} \right\rfloor = 7$ 이다.

인덱스 7의 H는 J보다 사전적으로 앞에 오는 문자이다.

∴ 인덱스 8 이후의 원소들을 탐색한다.

(3) 왼쪽 인덱스 l 은 8, 오른쪽 인덱스 r 은 9이므로 기준은 $\left\lfloor \frac{8+9}{2} \right\rfloor = 8$ 이다.

인덱스 8의 I는 J보다 사전적으로 앞에 오는 문자이다.

∴ 인덱스 9 이후의 원소들을 탐색한다.

(4) 왼쪽 인덱스 l 은 9, 오른쪽 인덱스 r 은 9이므로 기준은 $\left\lfloor \frac{9+9}{2} \right\rfloor = 9$ 이다.

인덱스 9의 J가 목표하던 원소이다.

∴ 탐색 종료

탐색 알고리즘

- 이진탐색 알고리즘(Binary search algorithm)
 - 이진탐색트리의 성질
 - 이진탐색트리의 루트부터 한 레벨씩 내려갈 때마다 탐색할 노드의 수는 반으로 줄어듬
 - 이진탐색트리를 구성하는 노드의 수 n 은 최대 $2^{h+1} - 1$ 개임 (h : 트리의 높이)

$$\begin{aligned} n \leq 2^H - 1 &\Rightarrow n + 1 \leq 2^H \Rightarrow \log_2(n + 1) \leq H \\ \therefore \log_2(n + 1) \leq H &\leq \log_2(n + 1) + 1 \end{aligned}$$

$$f(n) = \log_2 n + 1 \quad \therefore \text{알고리즘 복잡도 } O(\log_2 n)$$

탐색 알고리즘

- **버블 정렬(Bubble sort)**
 - 인접하는 2개의 원소를 비교해 기준에 따라 순서를 바꾸는 방식
 - 다음 원소들의 오름차순 정렬

인덱스	0	1	2	3	4
원소	16	3	7	4	10

정렬 알고리즘

• 1단계

(1) 원소 $A[0](=16) > A[1](=3)$ \therefore 인덱스 0의 원소와 1의 원소를 바꾼다.

인덱스	0	1	2	3	4
원소	3	16	7	4	10

(2) 원소 $A[1](=16) > A[2](=7)$ \therefore 인덱스 1의 원소와 2의 원소를 바꾼다.

인덱스	0	1	2	3	4
원소	3	7	16	4	10

(3) 원소 $A[2](=16) > A[3](=4)$ \therefore 인덱스 2의 원소와 3의 원소를 바꾼다.

인덱스	0	1	2	3	4
원소	3	7	4	16	10

(4) 원소 $A[3](=16) > A[4](=10)$ \therefore 인덱스 3의 원소와 4의 원소를 바꾼다.

인덱스	0	1	2	3	4
원소	3	7	4	10	16

정렬 알고리즘

• 2단계

(1) 원소 $A[0](=3) < A[1](=7)$ \therefore 인덱스 0의 원소와 1의 원소는 그대로 유지한다.

인덱스	0	1	2	3	4
원소	3	7	4	10	16

(2) 원소 $A[1](=7) > A[2](=4)$ \therefore 인덱스 1의 원소와 2의 원소를 바꾼다.

인덱스	0	1	2	3	4
원소	3	4	7	10	16

(3) 원소 $A[2](=7) < A[3](=10)$ \therefore 인덱스 2의 원소와 3의 원소는 그대로 유지한다.

인덱스	0	1	2	3	4
원소	3	4	7	10	16

정렬 알고리즘

• 3단계

(1) 원소 $A[0](=3) < A[1](=4)$ \therefore 인덱스 0의 원소와 1의 원소는 그대로 유지한다.

인덱스	0	1	2	3	4
원소	3	4	7	10	16

(2) 원소 $A[1](=4) < A[2](=7)$ \therefore 인덱스 1의 원소와 2의 원소는 그대로 유지한다.

인덱스	0	1	2	3	4
원소	3	4	7	10	16

• 4단계

(1) 원소 $A[0](=3) < A[1](=4)$ \therefore 인덱스 0의 원소와 1의 원소는 그대로 유지한다.

인덱스	0	1	2	3	4
원소	3	4	7	10	16

정렬 알고리즘

- 버블 정렬(Bubble sort)

```
1  algorithm bubble(int arr[], int n){  
2      int i, j;  
3      int tmp;  
4      for i = 1 to n-1  
5          for j = 1 to n-i  
6              if arr[j-1] > arr[j] then  
7                  tmp = arr[j-1];  
8                  arr[j-1] = arr[j];  
9                  arr[j] = tmp;  
10             endif  
11             next j  
12         next i  
13     }
```

$$f(n) = 3\{(n-1) + (n-2) + \dots + 2 + 1\} = \frac{3n(n-1)}{2}$$

\therefore 알고리즘 복잡도 $O(n^2)$

정렬 알고리즘

- **삽입정렬(Insertion sort)**

- 원소 집합 중 가장 첫 번째 원소를 이미 정렬된 원소라고 가정하여 그 다음 원소부터 정렬된 원소를 기준으로 적절한 위치에 삽입하는 방식
- 다음 원소들의 오름차순 정렬

인덱스	0	1	2	3	4	
원소	16	3	7	4	10	
정렬 여부	sorted	unsorted				

정렬 알고리즘

- 1단계 : 정렬할 원소는 $A[1](=3)$

(1) $A[0](=16) >$ 정렬할 원소 3 \therefore 인덱스 1에 16을 삽입한다.

인덱스	0	1	2	3	4
원소	16	16	7	4	10

(2) 더 비교할 정렬된 원소가 없으므로 인덱스 0에 3을 삽입한다.

인덱스	0	1	2	3	4
원소	3	16	7	4	10
정렬 여부	sorted		unsorted		

정렬 알고리즘

- 2단계 : 정렬할 원소는 $A[2](=7)$

(1) $A[1](=16) >$ 정렬할 원소 7 \therefore 인덱스 2에 16을 삽입한다.

인덱스	0	1	2	3	4
원소	3	16	16	4	10

(2) $A[0](=3) <$ 정렬할 원소 7 \therefore 그대로 유지한다.

(3) 더 비교할 정렬된 원소가 없으므로 인덱스 1에 7을 삽입한다.

인덱스	0	1	2	3	4
원소	3	7	16	4	10
정렬 여부	sorted			unsorted	

정렬 알고리즘

- 3단계 : 정렬할 원소는 $A[3](=4)$

(1) $A[2](=16) >$ 정렬할 원소 4 \therefore 인덱스 3에 16을 삽입한다.

인덱스	0	1	2	3	4
원소	3	7	16	16	10

(2) $A[1](=7) >$ 정렬할 원소 4 \therefore 인덱스 2에 7을 삽입한다.

인덱스	0	1	2	3	4
원소	3	7	7	16	10

(3) $A[0](=3) <$ 정렬할 원소 4 \therefore 그대로 유지한다.

(4) 더 비교할 정렬된 원소가 없으므로 인덱스 1에 4를 삽입한다.

인덱스	0	1	2	3	4
원소	3	4	7	16	10
정렬 여부	sorted				unsorted

정렬 알고리즘

- 4단계 : 정렬할 원소는 $A[4](=10)$

(1) $A[3](=16) >$ 정렬할 원소 10 \therefore 인덱스 4에 16을 삽입한다.

인덱스	0	1	2	3	4
원소	3	4	7	16	16

(2) $A[2](=7) <$ 정렬할 원소 10 \therefore 그대로 유지한다.

(3) $A[1](=4) <$ 정렬할 원소 10 \therefore 그대로 유지한다.

(4) $A[0](=3) <$ 정렬할 원소 10 \therefore 그대로 유지한다.

(5) 더 비교할 정렬된 원소가 없으므로 인덱스 3에 10을 삽입한다.

인덱스	0	1	2	3	4
원소	3	4	7	10	16
정렬 여부	sorted				

정렬 알고리즘

```
1 algorithm insertion(int arr[], int n){  
2     int i, j;  
3     int t, tmp;  
4     for i = 1 to n-1  
5         tmp = arr[i];  
6         j = i - 1;  
7         t = i;  
8         while (j >= 0)  
9             if arr[j] > tmp then  
10                 arr[j+1]=arr[j];  
11                 t = j;  
12             endif  
13             j = j - 1;  
14         endwhile  
15         arr[t] = tmp;  
16     next i  
17 }
```

$$f(n) = 4(n-1) + \{1 + 2 + \dots + (n-2) + (n-1)\} = 4(n-1) + \frac{n(n-1)}{2} = \frac{(n-1)(n+8)}{2}$$

∴ 알고리즘 복잡도 $O(n^2)$

정렬 알고리즘

- 퀵 정렬(Quick sort)
 - 기준값을 두고 그 기준값보다 큰 값과 작은 값으로 나누어 각 집합을 정렬하는 방식
 - 퀵 정렬의 과정

- ① 주어진 원소들의 가장 앞에서부터 탐색하여 기준값보다 큰 원소의 인덱스를 찾는다.
- ② 주어진 원소들의 가장 뒤에서부터 역으로 탐색하여 기준값보다 작은 원소의 인덱스를 찾는다.
- ③ ①에서 찾은 원소의 인덱스가 ②에서 찾은 원소의 인덱스보다 작으면 서로 교환한다.
- ④ ①에서 원소를 찾을 수 없거나 ①에서 찾은 원소의 인덱스가 ②에서 찾은 원소의 인덱스보다 크고 ②에서 찾은 원소의 인덱스가 기준값의 인덱스보다 크면, ②에서 탐색된 인덱스의 원소와 기준값을 교환한다.
- ⑤ ②에서 원소를 찾을 수 없거나 ②에서 찾은 원소의 인덱스가 기준값보다 작은 경우 다른 기준값으로 변경한다.

정렬 알고리즘

• 퀵 정렬(Quick sort)

인덱스	0	1	2	3	4
원소	7	3	16	4	10

- 1단계 : 기준값은 $A[0](=7)$

(1) 원소들의 앞에서부터 순서대로 탐색하여 기준값보다 큰 원소를 찾는다.

$$A[1](=3) < A[0](=7), \quad A[2](=16) > A[0](=7)$$

기준값보다 큰 원소(16)의 인덱스는 2이다.

(2) 원소들의 뒤에서부터 탐색하여 기준값보다 작은 원소를 찾는다.

$$A[4](=10) > A[0](=7), \quad A[3](=4) < A[0](=7)$$

기준값보다 작은 원소(4)의 인덱스는 3이다.

\therefore 기준값보다 큰 원소(16)의 인덱스(2)가 작은 원소(4)의 인덱스(3)보다 작으므로 두 원소를 교환한다.

기준값	P				
인덱스	0	1	2	3	4
원소	7	3	4	16	10

정렬 알고리즘

- 2단계 : 기준값은 $A[0](=7)$

(1) 원소들의 앞에서부터 순서대로 탐색하여 기준값보다 큰 원소를 찾는다.

$$A[1](=3) < A[0](=7), \quad A[2](=4) < A[0](=7)
A[3](=16) > A[0](=7)$$

기준값보다 큰 원소(16)의 인덱스는 3이다.

(2) 원소들의 뒤에서부터 탐색하여 기준값보다 작은 원소를 찾는다.

$$A[4](=10) > A[0](=7), \quad A[3](=16) > A[0](=7)
A[2](=4) < A[0](=7)$$

기준값보다 작은 원소(4)의 인덱스는 2이다.

\therefore 기준값보다 큰 원소(16)의 인덱스(3)가 작은 원소(4)의 인덱스(2)보다 크고, 작은 원소(4)의 인덱스(2)가 기준값(7)의 인덱스(0)보다 크므로 기준값보다 작은 원소(4)와 기준값(7)을 교환한다.

기준값	P				
인덱스	0	1	2	3	4
원소	4	3	7	16	10

정렬 알고리즘

- 3단계 : 기준값은 $A[0](=4)$

(1) 원소들의 앞에서부터 순서대로 탐색하여 기준값보다 큰 원소를 찾는다.

$$A[1](=3) < A[0](=4), \quad A[2](=7) > A[0](=4)$$

기준값보다 큰 원소(7)의 인덱스는 2이다.

(2) 원소들의 뒤에서부터 탐색하여 기준값보다 작은 원소를 찾는다.

$$A[4](=10) > A[0](=4), \quad A[3](=16) > A[0](=4)$$

$$A[2](=7) > A[0](=4), \quad A[1](=3) < A[0](=4)$$

기준값보다 작은 원소(3)의 인덱스는 1이다.

∴ 기준값보다 큰 원소(7)의 인덱스(2)가 작은 원소(3)의 인덱스(1)보다 크고, 작은 원소(3)의 인덱스

(1)가 기준값(4)의 인덱스(0)보다 크므로 기준값보다 작은 원소(3)와 기준값(4)을 교환한다.

기준값	P				
인덱스	0	1	2	3	4
원소	3	4	7	16	10

정렬 알고리즘

- 4단계 : 기준값은 $A[0](=3)$

(1) 원소들의 앞에서부터 순서대로 탐색하여 기준값보다 큰 원소를 찾는다.

$$A[1](=4) > A[0](=3)$$

기준값보다 큰 원소(4)의 인덱스는 1이다.

(2) 원소들의 뒤에서부터 탐색하여 기준값보다 작은 원소를 찾는다.

$$A[4](=10) > A[0](=3), \quad A[3](=16) > A[0](=3)$$

$$A[2](=7) > A[0](=3), \quad A[1](=4) > A[0](=3)$$

기준값보다 작은 원소를 찾을 수 없다.

∴ 기준값보다 작은 원소를 찾을 수 없으므로 기준값을 $A[1]$ 로 변경한다.

기준값		P			
인덱스	0	1	2	3	4
원소	3	4	7	16	10

정렬 알고리즘

- 5단계 : 기준값은 $A[1](=4)$

(1) 정렬되지 않은 남은 원소들의 앞에서부터 순서대로 탐색하여 기준값보다 큰 원소를 찾는다.

$$A[2](=7) > A[1](=4)$$

기준값보다 큰 원소(7)의 인덱스는 2이다.

(2) 원소들의 뒤에서부터 탐색하여 기준값보다 작은 원소를 찾는다.

$$A[4](=10) > A[1](=4), \quad A[3](=16) > A[1](=4)$$

$$A[2](=7) > A[1](=4)$$

기준값보다 작은 원소를 찾을 수 없다.

∴ 기준값보다 작은 원소를 찾을 수 없으므로 기준값을 $A[2]$ 로 변경한다.

기준값			P		
인덱스	0	1	2	3	4
원소	3	4	7	16	10

정렬 알고리즘

- 6단계 : 기준값은 $A[2](=7)$

(1) 남은 원소들의 앞에서부터 순서대로 탐색하여 기준값보다 큰 원소를 찾는다.

$$A[3](=16) > A[2](=7)$$

기준값보다 큰 원소(16)의 인덱스는 3이다.

(2) 원소들의 뒤에서부터 탐색하여 기준값보다 작은 원소를 찾는다.

$$A[4](=10) > A[2](=7), \quad A[3](=16) > A[2](=7)$$

기준값보다 작은 원소를 찾을 수 없다.

\therefore 기준값보다 작은 원소를 찾을 수 없으므로 기준값을 $A[3]$ 으로 변경한다.

기준값			P	
인덱스	0	1	2	3
원소	3	4	7	16

정렬 알고리즘

- 7단계 : 기준값은 $A[3](=16)$

(1) 남은 원소들의 앞에서부터 순서대로 탐색하여 기준값보다 큰 원소를 찾는다.

$$A[4](=10) < A[3](=16)$$

기준값보다 큰 원소를 찾을 수 없다.

(2) 원소들의 뒤에서부터 탐색하여 기준값보다 작은 원소를 찾는다.

$$A[4](=10) < A[3](=16)$$

기준값보다 작은 원소(10)의 인덱스는 4이다.

\therefore 기준값보다 큰 원소를 찾을 수 없으므로 기준값보다 작은 원소(10)와 기준값(16)을 교환한다.

기준값	sorted			P	
인덱스	0	1	2	3	4
원소	3	4	7	10	16

정렬 알고리즘

- 8단계 : 기준값은 $A[3](=10)$

(1) 남은 원소들의 앞에서부터 순서대로 탐색하여 기준값보다 큰 원소를 찾는다.

$$A[4](=16) > A[3](=10)$$

기준값보다 큰 원소(16)의 인덱스는 4이다.

(2) 원소들의 뒤에서부터 탐색하여 기준값보다 작은 원소를 찾는다.

$$A[4](=16) > A[3](=10), \quad A[2](=7) < A[3](=10)$$

기준값보다 작은 원소(7)의 인덱스는 2이다.

\therefore 기준값보다 큰 원소(16)의 인덱스(4)가 기준값의 인덱스(3)보다 크고 작은 원소(7)의 인덱스(2)가 기준값의 인덱스(3)보다 작으므로 기준값을 $A[4]$ 로 변경한다.

기준값	sorted				P
인덱스	0	1	2	3	4
원소	3	4	7	10	16

정렬 알고리즘

```
1  algorithm quick(int arr[], int left, int right){  
2      int p;  
3      int tmp;  
4      if left < right then  
5          p = arr[left];  
6          i = left;  
7          j = right + 1;  
8          do  
9              do  
10                 i = i + 1;  
11                 while(arr[i] < p);  
12                 do  
13                     j = j - 1 ;  
14                     while(arr[j] > p);  
15                     if i < j then  
16                         tmp = arr[i];  
17                         arr[i] = arr[j];  
18                         arr[j] = tmp;  
19                     endif  
20                     while(i < j);  
21                     tmp = arr[left];  
22                     arr[left] = arr[j];  
23                     arr[j] = tmp;  
24                     quick(arr, left, j-1);  
25                     quick(arr, j+1, right);  
26                 endif  
27 }
```

정렬 알고리즘

- **합병정렬(Merge sort)**
 - 원소의 집합을 비슷한 크기로 반복해서 나눈 뒤, 나누어진 집합을 정렬해가며 합병하는 방식
 - 다음 원소들의 오름차순 정렬

인덱스	0	1	2	3	4
원소	16	3	7	4	10

정렬 알고리즘

- 1단계 : 주어진 5개의 원소를 3개 또는 2개로 분할하는 것을 시작으로 각 원소가 리프노드가 될 때까지 분할하여 이진트리를 만든다.



정렬 알고리즘

- 2단계 : 이진트리의 리프노드부터 오름차순으로 정렬하여 하나의 집합으로 만든다.

(1) 레벨 3 : $16 > 3$ $\therefore 3, 16$ 의 순으로 나열한다.



정렬 알고리즘

(2) 레벨 2 : $16 > 7 > 3$

$\therefore 3, 7, 16$ 의 순으로 나열한다.

$10 > 4$

\therefore 그대로 유지한다.

레벨 0	인덱스	0	1	2	3	4
	원소	16	3	7	4	10

레벨 1	인덱스	0	1	2	3	4
	원소	3	7	16	4	10

레벨 2	인덱스	0	1	2	3	4
	원소	3	16	7	4	10

정렬 알고리즘

(3) 레벨 1 : $16 > 10 > 7 > 4 > 3 \quad \therefore 3, 4, 7, 10, 16$ 의 순으로 나열한다.

레벨 0	인덱스	0	1	2	3	4	
	원소	3	4	7	10	16	
레벨 1	인덱스	0	1	2	인덱스	3	4
	원소	3	7	16	원소	4	10

정렬 알고리즘

```
1  algorithm merge(int arr[], int left, int middle, int right){
2      int tmp[];
3      int i = left;
4      int j = middle + 1;
5      int k = left;
6      int x;
7      while (i <= middle && j <= right)
8          if arr[i] <= arr[j] then
9              tmp[k] = arr[i];
10             i = i + 1;
11         else
12             tmp[k] = arr[j];
13             j = j + 1;
14         endif
15         k = k + 1;
16     endwhile
17     if i > middle then
18         for x = j to right
19             tmp[k] = arr[x];
20             k = k + 1;
21         next x
22     else
23         for x = i to middle
24             tmp[k] = arr[x];
25         k = k + 1;
26     next x
27 endif
28 for x = left to right
29     arr[x] = tmp[x];
30 next x
31 }
32
33 algorithm mergeSort(int arr[], int left, int right){
34     int middle;
35     if left < right then
36         middle = (left + right) / 2
37         mergeSort(arr, left, middle);
38         mergeSort(arr, middle+1, right);
39         merge(arr, left, middle, right);
40     endif
41 }
```

Next Session

- Formal languages

Any questions?

