# 1 알고리즘과 문제 해결

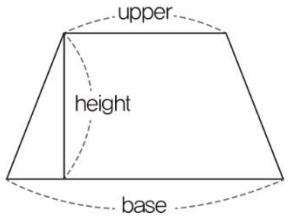
# 1 알고리즘과 문제 해결

## 1 문제해결 전략

- 주어진 조건에 맞추어 다각도로 문제에 접근
- 문제를 철저히 분석
- 그림을 그리거나, 식을 만들거나, 또는 규칙 찾기
- 조건에 따라 거꾸로 생각해보기
- 단계적으로 생각하기(알고리즘)

## 1 문제해결 전략

◆ 예) 사다리꼴의 면적을 구하는 알고리즘




- 사다리꼴의 면적을 구하는 공식
  면적 = (밑변 + 윗변) × 높이 / 2
- 단계별 의사코드로 나타내면 바로 알고리즘

① base = 20 ② upper = 10 ③ height=8
④ area = base + upper
⑤ area = area × height ⑥ area = area / 2

## 1 문제해결 전략

◆ 예) 유클리드 알고리즘으로 최대공약수를 구하는 법

풀이)
① 두 수 a, b에서 큰 수를 a라 하고 작은 수를 b라 함
② a가 b보다 클 때 a를 b로 나눔
③ 만약 나머지 r = 0이면 b가 최대공약수로 결정되고 완료 나머지 r ≠ 0이면 b는 a가 되고, 나머지 r이 b가 됨
④ ②로 가서 반복함

## 1 문제해결 전략

◆ 예) 유클리드 알고리즘으로 최대공약수를 구하는 법

- 유클리드 알고리즘의 다른 풀이 방법
  : 78과 36의 최대공약수는
  $2 \times 3 = 6$

$$
\begin{array}{r|ll}
2 & 78 & 36 \\
\hline
3 & 39 & 18 \\
\hline
 & 13 & 6
\end{array}
$$

## 1 문제해결 전략

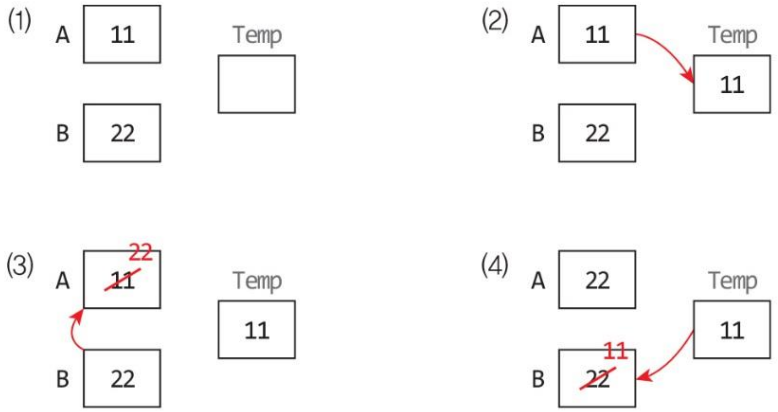
◆ 예) 수 교환하기
A = 11, B = 22 일 때 두 수를 교환하는 방법

풀이)
Temp = A;
A = B;
B = Temp;

- A의 값을 임시 저장장소인 Temp에 이동
- B의 값을 A의 값에다 이동
- Temp에 있는 값을 B로 이동
  → 그 결과 A와 B의 값이 교환됨

## 1 문제해결 전략

- A = B이면 A의 내용이 B의 내용으로 덮어씌워지게 되어 원래 A 값을 잃게 됨
  - 따라서 임시 저장장소인 Temp를 사용해야 함



# 2 점화식

# 2 점화식

## 1 점화식(Recurrence Relation)

- 어떤 함수를 자신보다 더 작은 변수에 대한 함수와의 관계로 표현한 것
- 자기호출을 사용하는 함수(재귀)의 복잡도를 구하는데 유용함
- 수열은 원소들을 일정한 순서로 나열한 형태로 표현하는데 이러한 원소들 사이에는 일정한 규칙이 있을 수 있고 원소들의 나열 대신 이 규칙으로 표현하는 것도 점화식임

# 2 점화식

## 1 점화식(Recurrence Relation)

◆ 예)

- $a_n = a_{n-1} + 2$
- $f(n) = nf(n-1)$
- $f(n) = f(n-1) + f(n-2)$
- $f(n) = f(n/2) + n$

# 2 점화식

## 1 점화식(Recurrence Relation)

- 예) 1, 2, 4, 8, 16, … 와 같은 수열에서 이 수열은 앞의 항에 2를 곱하여 다음 항이 만들어진다.
  따라서 n번째 수열 $a_n$은 앞의 항인 $a_{n-1}$ 에 2를 곱한 형식으로 표현하여 $a_n = 2a_{n-1}$ 이다.

이와 같이 수열에서 n번째 항 $a_n$을
그 앞의 항인 $a_{n-1}$, $a_{n-2}$,… 등에 의해 표현하는 형식을
**점화식**이라 하며 적절한 처음 몇 항이 주어지면
수열의 모든 항을 구할 수 있음

# 2 점화식

## 1 점화식(Recurrence Relation)

### 예) 피보나치 수열을 이용한 점화식

갓 태어난 1쌍의 새끼토끼가 있다.
새끼토끼 1쌍은 2달 후부터는 매달 암수 1쌍의 토끼를 낳는다고 가정하면 1년 후에는 토끼가 모두 몇 쌍이나 되는지 생각해보고 n번째 달의 토끼 쌍의 수를 점화식으로 나타내시오.
(단, 모든 토끼는 죽지 않는다고 가정)

## 1 점화식(Recurrence Relation)

풀이)
맨 처음에는 새끼토끼 1쌍이 있었으므로 $f_0=1$

1달 후에는 새끼토끼가 자라지만 새끼토끼는 2달 후부터 새끼를 낳을 수 있다고 했으므로 여전히 $f_1=1$

다시 1달 후는 비로소 새끼를 낳을 수 있으므로 1쌍을 낳아 $f_2=2$

같은 방법으로 다시 1달 후에는 어미토끼가 1쌍의 새끼토끼를 낳지만 이전 달에 낳은 새끼토끼는 아직 새끼를 낳을 수 없으므로 $f_3=3$

# 2 점화식

## 1 점화식(Recurrence Relation)

◆ 풀이)

| 개월 | 토끼 쌍(새끼토끼(○), 어른토끼(●)) | 토끼 쌍의 수 |
|---|---|---|
| 시작 | ○ | 1 |
| 1개월 | ● | 1 |
| 2개월 | ● ○ | 2 |
| 3개월 | ● ○ ● | 3 |
| 4개월 | ● ○ ● ● ○ | 5 |
| 5개월 | ● ○ ● ● ○ ● ○ ● | 8 |
| 6개월 | ● ○ ● ● ○ ● ○ ● ● ○ ● ● ○ | 13 |

## 1 점화식(Recurrence Relation)

풀이)
토끼 쌍의 수는 다음과 같은 피보나치 수열로 증가

1, 1, 2, 3, 5, 8, 13, …
n달 후의 토끼 쌍의 수를 $f_n$ 이라 할 때

점화식을 구하면
$f_n = f_{n-1} + f_{n-2}$ 이며 초깃값 $f_0$=1, $f_1$=1

## 2 병합 정렬을 이용한 점화식의 수행 시간

- 병합 정렬도 자신보다 더 작은 변수에 대한 함수와의 관계로 표현 가능
- 입력의 크기가 n인 배열을 병합 정렬하려면 일단 배열을 이등분한 다음 각각을 재귀적으로 병합해 이들을 다시 병합함으로써 정렬이 끝남

## 2 병합 정렬을 이용한 점화식의 수행 시간

```
mergeSort(A[ ], p, r)      ▷ A[p ... r]을 정렬
{
    if (p < r) then {
        q ← ⌊(p+r)/2⌋;  -------- ①  ▷ p, r의 중간 지점 계산
        mergeSort(A, p, q); ---------- ②  ▷ 전반부 정렬
        mergeSort(A, q+1, r); ---------- ③  ▷ 후반부 정렬
        merge(A, p, q, r);  ---------- ④  ▷ 병합
    }
}
merge(A[ ], p, q, r)
{
    정렬된 두 배열 A[p ... q]와 A[q+1 ... r]을 합하여
    정렬된 하나의 배열 A[p ... r]을 만듦
}
```

## 2 병합 정렬을 이용한 점화식의 수행 시간

- 입력의 크기가 n인 병합 정렬 시간은 크기가 n/2인 병합 정렬을 두 번 하는 시간과 나머지 오버헤드를 더한 시간임
- 수행 시간의 점화식 : $T(n) = 2T(n/2) + 오버헤드$

- 예) 입력의 크기가 n인 문제를 해결하는 어떤 알고리즘이 자신보다 크기가 1/4인 문제를 한번 풀고 후처리에 상수시간이 든다면 수행시간은 점화식으로 어떻게 표현할까요?
- 수행 시간의 점화식 : $T(n) = T(n/4) + 오버헤드$

# 3 점화식의 점근적 분석 방법

# 3 점화식의 점근적 분석 방법

## 1 점화식의 점근적 분석 방법

| | |
|---|---|
| **반복 대치** | 더 작은 문제에 대한 함수로 반복해서 대치해 나가는 해법 |
| **추정 후 증명** | 결론을 추정하고 수학적 귀납법을 이용하여 증명하는 방법 |
| **마스터 정리** | 형식에 맞는 점화식의 복잡도는 복잡한 계산이나 증명 없이 바로 알 수 있음 |

# 3 점화식의 점근적 분석 방법

## 2 반복 대치

T(n)을 T(n−1)로 대치하고, T(n-1)을 T(n-2)로 대치하고, 계속해서 T(n-2), T(n-3), T(n-4)… T(1)까지 반복해서 대치해가는 방식을 사용해 점근적 복잡도를 구함

# 3 점화식의 점근적 분석 방법

## 2 반복 대치

$$T(n) = T(n-1) + c$$
$$T(1) \leq c$$

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

따라서 $T(n) \leq cn$이므로 $T(n)=O(n)$ 임

# 3 점화식의 점근적 분석 방법

## 2 반복 대치

### (예) 병합 정렬

- 입력의 크기가 n인 배열에 대한 병합 정렬에서 **대소비교**의 총 횟수를 T(n)이라 함
- 즉, 두 수의 대소를 비교하는 횟수를 수행시간의 기준으로 삼는데 원래 문제와 이등분된 문제 간의 관계를 다음과 같이 표현할 수 있음

# 3 점화식의 점근적 분석 방법

## 2 반복 대치

### (예) 병합 정렬

- 수행 시간의 점화식: $T(n) = 2T(n/2) + 오버헤드$

$$T(n) \leq 2T(n/2) + n$$

  - 전체 n개를 n/2로 나눈 2개가 각각 정렬 완료된 후 이 2개의 결과를 각각 비교하여 하나로 병합할 때 비교횟수는 최대 n이므로 오버헤드는 n이 됨

# 3 점화식의 점근적 분석 방법

## 2 반복 대치

### (예) 병합 정렬

$$T(n) \leq 2T(n/2) + n$$

- T(n) :
  입력크기가 n인 문제를 병합 정렬하는데
  필요한 총 비교 횟수

- T(n/2) :
  입력크기가 n/2인 문제를 병합 정렬하는데
  필요한 총 비교 횟수

# 3 점화식의 점근적 분석 방법

## 2 반복 대치

### (예) 병합 정렬

$$T(n) \leq 2T(n/2) + n$$

- n :
  병합정렬에서 merge(A,p,q,r)에 필요한
  최대 비교 회수에 대한 오버헤드

  ↳ 즉, 크기가 n인 문제를 풀기 위해서 크기가 n/2인 문제를 두 개 풀고, 나머지 최대 n의 오버헤드가 필요한 상황을 표현한 것

# 3 점화식의 점근적 분석 방법

## 2 반복 대치

### (예) 병합 정렬

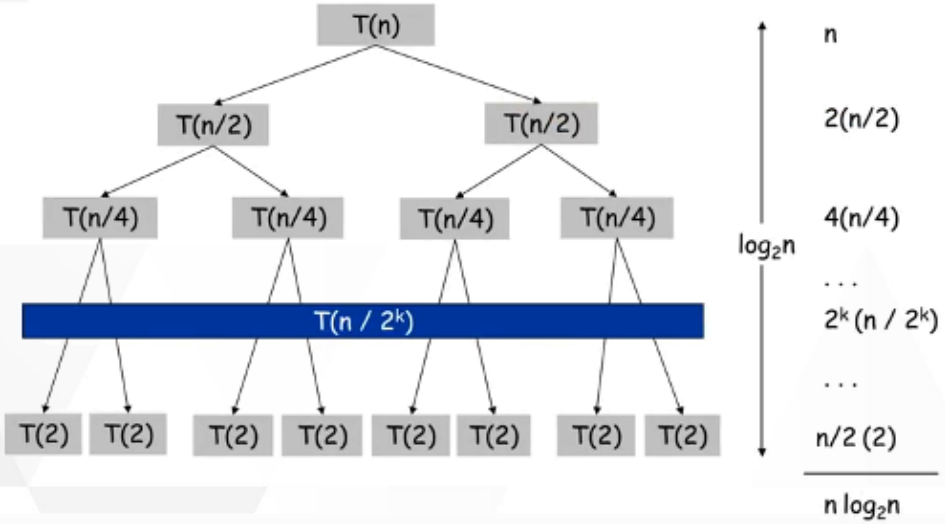
$T(n) = 2T(n/2) + n$
$T(1) = 1$

$T(n) \le 2T(n/2) + n$
$\le 2(2T(n/2^2) + n/2) + n = 2^2T(n/2^2) + 2n$
$\le 2^2(2T(n/2^3) + n/2^2) + 2n = 2^3T(n/2^3) + 3n$
...
$\le 2^kT(n/2^k) + kn$
$= nT(1)+kn$ ← $n=2^k$라고 가정할 수 있음
$= n + n \log n$ ← $T(1) = 1$이고 , $k=\log_2 n$이므로
$= O(n \log n)$

# 3 점화식의 점근적 분석 방법

## 2 반복 대치

### (예) 병합 정렬



# 3 점화식의 점근적 분석 방법

## 3 추정 후 증명

- 식의 모양을 보고 점근적 복잡도를 추정한 다음 그것이 옳음을 귀납적으로 증명하는 방법

# 3 점화식의 점근적 분석 방법

## 3 추정 후 증명

◆ 예) T(n) = 2T(n/2) + n의 점근적 복잡도는 T(n) = O(n log n)임
즉, 충분히 큰 n에 대하여 T(n) ≤ cn log n 인 양의 상수 c가 존재함
(추정 후 증명법을 이용해 확인)

〈증명〉 경계조건 : T(2) ≤ c2 log2 를 만족하는 c가 존재

귀납적 가정: n/2에 대해 T(n/2) ≤ c(n/2) log n/2을 만족

$$
\begin{aligned}
T(n) &= 2T(n/2) + n \\
&\le 2c(n/2)\log(n/2) + n \\
&= cn\log(n/2) + n \\
&= cn\log n - cn\log 2 + n \\
&= cn\log n + (-c\log 2 + 1)n \quad \leftarrow \text{이를 만족하는 c가 존재함} \\
&\le cn\log n
\end{aligned}
$$

# 3 점화식의 점근적 분석 방법

## 3 추정 후 증명

$$
\begin{aligned}
T(n) &= 2T(n/2) + n \\
&\le 2c(n/2)\log(n/2) + n \quad (1) \\
&= cn\log(n/2) + n \\
&= cn\log n - cn\log 2 + n \\
&= cn\log n + (-c\log 2 + 1)n \quad (2) \\
&\le cn\log n \quad (3)
\end{aligned}
$$

- ①은 귀납적 가정을 이용한 부분임
- 최종적으로 ③의 결론을 도출하려면 ②항이 음수 (정확하게는 0이하)이면 됨
  즉, c가 $\frac{1}{\log 2}$ 이상이기만 하면 됨

# 3 점화식의 점근적 분석 방법

## 3 추정 후 증명

$$
\begin{aligned}
T(n) &= 2T(n/2) + n \\
&\le 2c(n/2)\log(n/2) + n \quad (1) \\
&= cn\log(n/2) + n \\
&= cn\log n - cn\log 2 + n \\
&= cn\log n + (-c\log 2 + 1)n \quad (2) \\
&\le cn\log n \quad (3)
\end{aligned}
$$

- 충분히 큰 n에 대해 $T(n) \le cn\log n$인 상수 c를 잡을 수 있으므로 증명이 완료됨

# 3 점화식의 점근적 분석 방법

## 4 마스터 정리

- 마스터 정리는 특정한 모양을 가진 재귀식에서
  바로 결과를 알 수 있는 아주 유용한 정리
- 마스터 정리는 식 $T(n) = aT(n/b) + f(n)$ 에 적용됨
- 입력의 크기가 n인 문제를 풀기 위해
  입력의 크기가 n/b인 문제를 a개 풀고,
  나머지 f(n)의 오버헤드가 필요한 알고리즘들이
  해당됨
- 아주 많은 알고리즘이 이에 해당되어 마스터 정리는
  아주 유용한 방법임

## 4 마스터 정리

- 예) 병합 정렬이 대표적인 예임

  a=b=2, f(n)=n인 경우이며 결과는 상수 a, b와
  식 f(n)의 모양에 따라 정해짐

  $T(n) = 2T(n/2) + f(n)$

# 3 점화식의 점근적 분석 방법

## 4 마스터 정리

- $a \geq 1$, $b \geq 1$ 에 대해 $T(n) = aT(n/b) + f(n)$ 인 점화식에서 $n^{\log_b a} = h(n)$이라 할 때 $h(n)$과 $f(n)$의 상대적 무게에 따라 전체 복잡도가 결정됨

**마스터정리** T(n)의 점근적 복잡도는 다음과 같음

① 어떤 양의 상수 $\varepsilon$에 대하여 $f(n)/h(n) = O(1/n^{\varepsilon})$이면, $T(n) = \Theta(h(n))$이다.

② 어떤 양의 상수 $\varepsilon$에 대하여 $f(n)/h(n) = \Omega(n^{\varepsilon})$이고, 어떤 상수 $c(< 1)$와 충분히 큰 모든 n에 대해 $af(n/b) \leq cf(n)$이면 $T(n) = \Theta(f(n))$이다.

③ $f(n)/h(n) = \Theta(1)$이면 $T(n) = \Theta(h(n) \log n)$이다.

# 3 점화식의 점근적 분석 방법

## 4 마스터 정리

### 마스터정리 근사버전

① $\lim_{n\to\infty} \frac{f(n)}{h(n)}$ =0이면, T(n) = Θ(h(n))이다.

② $\lim_{n\to\infty} \frac{f(n)}{h(n)}$ =∞이고, 충분히 큰 모든 n에 대해 af(n/b) ≤ cf(n)이면 T(n) = Θ(f(n))이다.

③ $\frac{f(n)}{h(n)}$ = Θ(1)이면 T(n) = Θ(h(n) log n)이다.

### 마스터정리 대략정리

① h(n)이 더 무거우면 h(n)이 수행 시간을 결정한다.
② f(n)이 더 무거우면 f(n)이 수행 시간을 결정한다.
③ h(n)과 f(n)이 같은 무게이면 h(n)에 log n을 곱한 것이 수행 시간이 된다.