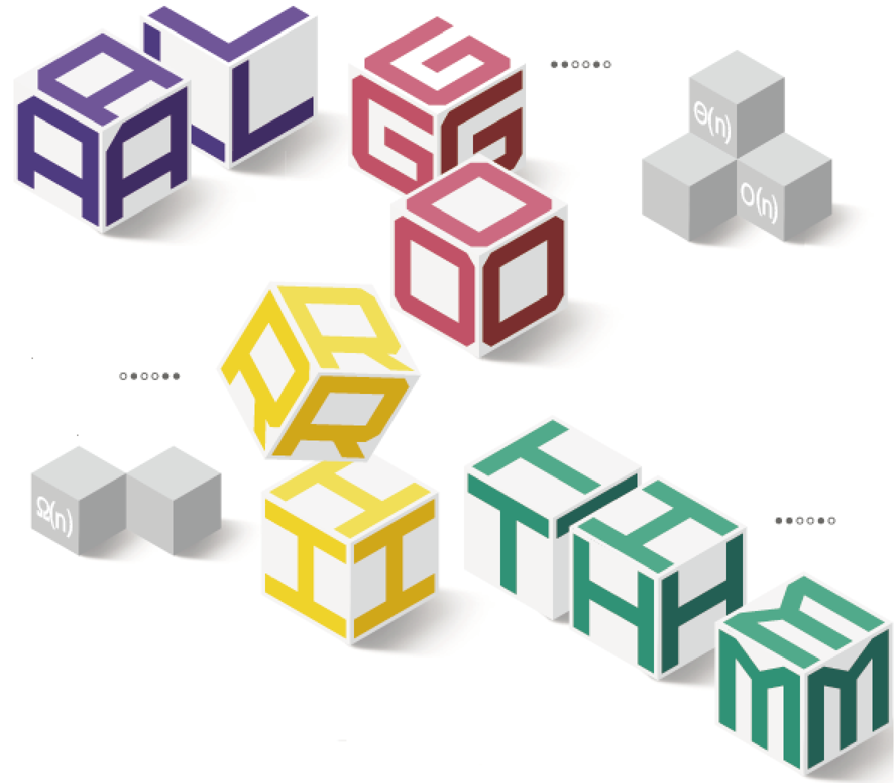

관계 중심의 사고법

쉽게 배우는 알고리즘

3장. 점화식과 점근적 복잡도 분석

학습목표

- 재귀 알고리즘과 점화식의 관계를 이해한다
- 점화식의 점근적 분석을 이해한다

(선수내용) 재귀호출,
merge 소트,
로그 함수
(수업내용)반복 대치로 merge 소트 분석

점화식의 이해

- 점화식
  - 어떤 함수를 자신보다 더 작은 변수에 대한 함수와의 관계로 표현한 것
- 예
  - $a_n = a_{n-1} + 2$
  - $f(n) = n\, f(n-1)$
  - $f(n) = f(n-1) + f(n-2)$
  - $f(n) = f(n/2) + n$

병합 정렬의 수행 시간

```
mergeSort(A[ ], p, r)      ▷ A[p ... r]을 정렬한다.
{
    if (p < r) then {
        q ← ⌊(p + q)/2⌋;  -----------------  ①  ▷ p, q의 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)
{
    정렬되어 있는 두 배열 A[p ... q]와 A[q+1 ... r]을 합하여
    정렬된 하나의 배열 A[p ... r]을 만든다.
}
```

수행 시간의 점화식: $T(n) = 2T(n/2)$ + 오버헤드

✓크기가 n인 병합 정렬 시간은 크기가 $n/2$인 병합 정렬을 두 번하는 시간과 나머지 오버헤드를 더한 시간이다

점화식의 점근적 분석 방법

- **반복 대치**
  - **더 작은 문제에 대한 함수로 반복해서 대치해 나가는 해법**
- 추정후 증명
  - 결론을 추정하고 수학적 귀납법으로 이용하여 증명하는 방법
- 마스터 정리
  - 형식에 맞는 점화식의 복잡도를 바로 알 수 있다

반복 대치

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

$T(1) \le c$

문제가 작은 문제로 나누어지면, ***T*(*n*)** *에서* ***T*(*n*−1)** 상수 ***c***가 추가됨

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

T*(*n*)** *이* ***1*개로 나누어지면* 상수 ***c***가 ***n -1***가 추가됨

반복 대치

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

문제가 **2**개의 작은 문제로 나누어지면

T(n) *에서* ***2T(n/2)*** 과 ***n***로 문제가 변경됨

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2(2T(n/2^2) + n/2) + n = 2^2T(n/2^2) + 2n \\ &= 2^2(2T(n/2^3) + n/2^2) + 2n = 2^3T(n/2^3) + 3n \\ &\dots \\ &= 2^kT(n/2^k) + kn \\ &= \cancel{n} + n\log n \\ &= \Theta(n\log n) \end{aligned}$$

k번 나누어짐

$$\longrightarrow \quad = nT(1) + \log n \cdot O(n)$$

$n/2^k$ 는 1

2^k 는 n 로 치환

(실습) 반복대치

- 이진검색
  - 배열에 저장된 자료를 검색하기 위해 자료의 중간위치와 반복비교를 하면 자료를 찾음
  - 비교수행 연산 후 다음 비교 대상 자료는 현재 자료의 반(1/2) 로 축소됨
  - 자료 개수가 n 인 경우 n/2, n/4, n/8 … 로 감소
  - 자료 개수가 1000개인 경우 최대 10번 만에 자료를 검색할 수 있음

O(logn)

Comparisons	Approximate Number of Items Left
1	$\frac{n}{2}$
2	$\frac{n}{4}$
3	$\frac{n}{8}$
...	
i	$\frac{n}{2^i}$

(실습) 반복 대치

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

문제가 2개의 작은 문제로 나누어지면

T(n) *에서* **2T(n/2)** 과 ***n***로 문제가 변경됨

$$\begin{aligned} T(n) &= T(n/2) + 1 \\ &= T(n/2^2) + 2 \\ &= T(n/2^3) + 3 \\ &\cdots \\ &= T(n/2^k) + \mathrm{k} \\ &= \cancel{1} + \log n \\ &= \Theta(\log n) \end{aligned}$$

k *번*
나누어짐

$$\longrightarrow \quad = T(1) + \log n$$

$n/2^k$ *는* 1
(2^k *는* n) *치환*

(실습) 이진검색 구현

```
def binarySearch(arr, item):
    if len(arr) == 0:
        return False
    else:
        midpoint = len(arr)//2
        if arr[midpoint]==item:
          return True
        else:
          if item<arr[midpoint]:  # midpoint 보다 작은 곳에서 찾음
            return binarySearch(arr[:midpoint],item)
          else:
            return binarySearch(arr[midpoint+1:],item)


testlist = [0, 1, 2, 8, 13, 17, 19, 32, 42,]
print(binarySearch(testlist, 3))  # 키값을 찾지 못함
print(binarySearch(testlist, 13)) # 키값을 찾음
```

Thank you