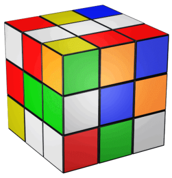
도사 정리

한국기술교육대학교 컴퓨터공학부 김상진



```
while(!morning){
    alcohol++
    dance++
} #party
```

```
while(!sleep){
    think++
    solve++
} #cse-mode
```

교육목표

- 도사정리: 재귀 알고리즘의 시간 복잡도를 결정해 주는 블랙박스 툴
  - 재귀 알고리즘의 점화식(recurrence)으로부터 시간 복잡도를 결정
    - 표준 점화식만 가능
- 여섯 가지 예제
- 도사 정리의 증명

$$T(n) \le aT\left(\frac{n}{b}\right) + O(n^d)$$

합병정렬
- 왼쪽 절반 정렬
- 오른쪽 절반 정렬
- 정렬된 양쪽 결합

$$T(n) \le 2T\left(\frac{n}{2}\right) + O(n)$$

⇩

$$a = 2, b = 2, d = 1$$

⇩

$\boxed{\text{도사정리}}$

⇩

$$O(n \log n)$$

정수 곱셈 재방문

- 두 개의 n자리 수 초등학교 곱셈 알고리즘: $\Theta(n^2)$
- 분할 정복 알고리즘
  - $x = 10^{n/2}a + b,\ y = 10^{n/2}c + d$
  - $x \cdot y = 10^n(ac) + 10^{n/2}(ad + bc) + bd$
    - 비용: 4개의 $n/2$자리 수 곱셈 + $O(n)$
      - 추가비용 $O(n)$: 0 추가, 3번 덧셈, 2번의 shift 연산
- 비용을 정형화된 형태로 표현하는 식을 점화식(재귀식)이라 함
- 점화식은 base case, general case 두 가지가 제시되어야 함
- 위에 제시된 분할 정복 알고리즘의 점화식
  - base case: $T(1) = O(1)$
  - general case: $T(n) \le 4T\left(\frac{n}{2}\right) + O(n)$

재귀호출의 수 | 소문제의 크기 | 비재귀 부분의 비용

정수 곱셈 재방문

- Karatsuba 분할 정복 알고리즘
  - $x = 10^{n/2}a + b,\ y = 10^{n/2}c + d$
  - $A = ac,\ B = bd,\ C = (a + b)(c + d) - A - B$
  - $x \cdot y = 10^n A + 10^{n/2} C + B$
  - 비용: 3개의 $n/2$자리 수 곱셈 + $O(n)$
- 점화식
  - base case: $T(1) = O(1)$
  - general case: $T(n) \le 3T\left(\frac{n}{2}\right) + O(n)$
- 실제 재귀 밖에서 일어나는 작업은 기본 분할 정복 방법보다 많지만 증가된 비용(2개의 덧셈과 2개의 뺄셈)을 고려하여도 시간 복잡도는 차이가 없음
- 두 알고리즘의 실제 시간복잡도는 알 수 없지만 Karatsuba가 우수하다는 것은 알 수 있음
  - 두 알고리즘은 모두 합병 정렬보다는 느리다는 것도 알 수 있음

표준 점화식

- 3개의 파라미터로 구성된 대부분 재귀 알고리즘을 분석할 때 사용할 수 있는 점화식은 다음과 같고, 이를 표준 점화식이라 함
- Base case: 다음을 만족하는 n과 독립적인 n_0와 c가 존재
  - $T(n) \le c,\ n \le n_0$
    - 의미: 입력 크기가 충분히 작으지면 $O(1)$에 처리 가능
- General case
  - $T(n) \le aT\left(\frac{n}{b}\right) + O(n^d)$

    비재귀적 부분의 비용이 다차 시간이어야 한다.
    - a: 재귀호출의 수($a \ge 1$)
    - b: 입력 크기가 줄어드는 정도($b > 1$)
    - d: 일반 과정에서 소요되는 시간 복잡도의 지수
      - a, b, d: 모두 n과 독립적인 상수
      - $d = 0$: 상수 시간
  - 제한. 재귀 호출의 입력 크기는 항상 같은 수준으로 줄어야 함
- 예) 합병 정렬: $a = 2, b = 2, d = 1$

The Master Method

- 표준 점화식으로 표현할 수 있는 재귀 알고리즘의 시간 복잡도는 다음과 같음
  - Case 1. $a = b^d,\ T(n) = O(n^d \log n)$
  - Case 2. $a < b^d,\ T(n) = O(n^d)$
  - Case 3. $a > b^d,\ T(n) = O(n^{\log_b a})$

  $$T(n) \le aT\left(\frac{n}{b}\right) + O(n^d)$$
- 경우 1에서 log의 base는 중요하지 않지만 경우 3은 중요
  - 경우 1에서는 base가 바뀌더라도 그 차이가 일정한 상수만큼의 차이만 발생하며, 이 차이는 빅O에 의해 무시됨

  $$\log_b n = \frac{\log_a n}{\log_a b}$$
- 궁금???
  - 왜 a와 b^d를 비교하지?
  - 경우 2는 재귀 비용은 무시된다는 것인데, 어떻게?
    $\Longrightarrow$ 도사 정리의 증명

예1, 2) 합병 정렬, 이진 탐색

- 합병 정렬: $a = 2, b = 2, d = 1$
  - $b^d = 2 = a$: 경우 1
    - $T(n) = O(n\log n)$
- 이진 탐색: $a = 1, b = 2, d = 0$
  - $b^d = 1 = a$: 경우 1
    - $T(n) = O(\log n)$

- Case 1. $a = b^d, T(n) = O(n^d \log n)$
- Case 2. $a < b^d, T(n) = O(n^d)$
- Case 3. $a > b^d, T(n) = O(n^{\log_b a})$

$$T(n) \le aT\left(\frac{n}{b}\right) + O(n^d)$$

예3, 4) 정수 곱셈

- 일반 재귀 알고리즘: $a = 4, b = 2, d = 1$
  - $b^d = 2 < a$: 경우 3
    - $T(n) = O\left(n^{\log_2 4}\right) = O(n^2)$
      - 초등학교 곱셈 알고리즘과 차이가 없음
      - Karatsuba는 $O(n\log n)$과 $O(n^2)$ 사이
- Karatsuba 알고리즘: $a = 3, b = 2, d = 1$
  - $b^d = 2 < a$: 경우 3
    - $T(n) = O\left(n^{\log_2 3}\right) = O(n^{1.59})$

$$\boxed{T(n) \le 4T\left(\frac{n}{2}\right) + O(n)}$$

$$\boxed{T(n) \le 3T\left(\frac{n}{2}\right) + O(n)}$$

- Case 1. $a = b^d, T(n) = O(n^d \log n)$
- Case 2. $a < b^d, T(n) = O(n^d)$
- Case 3. $a > b^d, T(n) = O(n^{\log_b a})$

$$T(n) \le aT\left(\frac{n}{b}\right) + O(n^d)$$

예5) 행렬 곱셈, 예6) 가상의 알고리즘

$T(n) \le 7T\left(\frac{n}{2}\right) + O(n^2)$

- Strassen 재귀 알고리즘: $a = 7, b = 2, d = 2$
  - $b^d = 4 < a$: 경우 3
    - $T(n) = O\left(n^{\log_2 7}\right) = O(n^{2.81})$
      - 일반 행렬 곱셈($O(n^3)$)보다 우수함
  - 참고. 일반 재귀 알고리즘: $T(n) = O\left(n^{\log_2 8}\right) = O(n^3)$
- 가상의 알고리즘: $T(n) \le 2T\left(\frac{n}{2}\right) + O(n^2)$
  - $a = 2, b = 2, d = 2$
  - $b^d = 4 > a$: 경우 2
    - $T(n) = O\left(n^2\right)$
  - $T(n) \le 2T\left(\frac{n}{2}\right) + O(n)$과 비교

$T(n) \le aT\left(\frac{n}{b}\right) + O(n^d)$

- Case 1. $a = b^d$, $T(n) = O(n^d \log n)$
- Case 2. $a < b^d$, $T(n) = O(n^d)$
- Case 3. $a > b^d$, $T(n) = O(n^{\log_b a})$

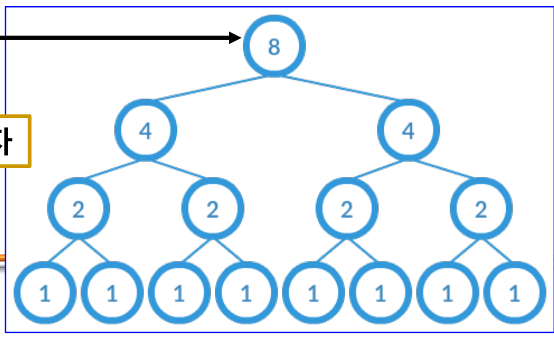
도사 정리의 증명

$T(n) \le aT\left(\frac{n}{b}\right) + O(n^d)$

- 증명 자체를 꼭 이해하고 스스로 할 수 있어야 하는 것은 아님
  - 하지만 증명을 보고 그것을 이해하는 과정에서 여러 가지 유익한 직관을 가지게 됨
- 합병 정렬의 시간 복잡도를 분석할 때를 생각하여 보자
  - 총 레벨의 수: $\log_2 n + 1$ → $\log_b n + 1$
  - 결과 배열의 크기가 n일 때 합병 비용: $6n$
  - 각 레벨 j에서 2^j개의 합병이 필요하고, 입력 배열의 크기는 $n/2^j$
    - 각 레벨의 소문제 수: a^j, 문제의 크기: n/b^j
  - 레벨 j에서 연산 수: $\le 2^j \times 6 \times \left(\frac{n}{2^j}\right) = 6n$

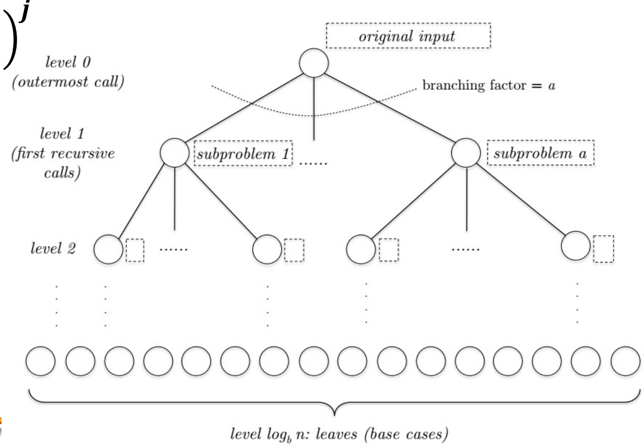
합병 정렬의 분석 방법을 일반화 해보자

Level 0:



일반화 $T(n) \le aT\left(\frac{n}{b}\right) + cn^d$

- 특정 레벨에서 필요한 일 (재귀 호출 제외)
  - $\le a^j \times c \times \left(\frac{n}{b^j}\right)^d = cn^d \cdot \left(\frac{a}{b^d}\right)^j$
    - 각 레벨의 소문제 수: a^j, 소문제의 크기: n/b^j,
      각 소문제마다 비용: $c \times \left(\frac{n}{b^j}\right)^d$
- 모든 레벨에서 소요된 일을 합하면…
  - Total Work $\le cn^d \cdot \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$



식의 해석

- Total Work $\le cn^d \cdot \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$
- a: RSP(rate of subproblem proliferation, 소문제 수 증가률) < evil
- b^d: RWS(rate of work shrinkage, 문제 크기 축소률) < good
- 3가지 경우
  - RSP $=$ RWS: 모든 레벨에서 해야 할 일이 같음
    - 도사 정리 경우 1. 예) 합병 정렬
  - RSP $<$ RWS: 레벨이 증가할수록 해야 할 일은 줄어듦
    - 도사 정리 경우 2.
    - 대부분의 일이 루트 레벨에서 이루어짐
  - RSP $>$ RWS: 레벨이 증가할수록 해야 할 일이 증가함
    - 도사 정리 경우 3
    - 대부분의 일이 단말 레벨에서 이루어짐

경우 1. RSP = RWS

- Total Work $\le cn^d \cdot \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$
  - $a = b^d$이면 $\frac{a}{b^d} = 1$
  - Total Work $\le cn^d \cdot (\log_b n + 1) = O(n^d \log n)$

- 경우 2와 경우 3을 보기 전에
  - $r \ne 1$이면 $1 + r + r^2 + \cdots + r^k = \frac{r^{k+1}-1}{r-1}$
    - $r < 1$이면 $\frac{r^{k+1}-1}{r-1} \le \frac{1}{1-r} = c$임
    - $r > 1$이면 $\frac{r^{k+1}-1}{r-1} \le \frac{r^{k+1}}{r-1} \le r^k(\frac{r}{r-1})$ 임

경우 2. RSP < RWS

- Total Work $\le cn^d \cdot \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$
  - $a < b^d$이면 $\frac{a}{b^d} < 1$
  - Total Work $\le cn^d \cdot c' = O(n^d)$
  - 전체 일은 루트 레벨에서 일어나는 일이 좌우함

경우 3. RSP > RWS

- Total Work $\leq cn^d \cdot \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$
  - $a > b^d$이면 $\frac{a}{b^d} > 1$
  - Total Work $\leq cn^d \cdot \left(\frac{a}{b^d}\right)^{\log_b n} = cn^d \cdot a^{\log_b n} \cdot b^{-d\log_b n} = a^{\log_b n}$
    - $b^{-d\log_b n} = (b^{\log_b n})^{-d} = n^{-d}$
  - $a^{\log_b n}$: 단말의 수
    - 전체 일은 단말 레벨에서 일어나는 일이 좌우함
  - $a^{\log_b n} = n^{\log_b a}$
    - $\log_b a^{\log_b n} = \log_b n \log_b a = \log_b n^{\log_b a} = \log_b n \log_b a$
  - Total work $\leq O(n^{\log_b a})$