§4.2. Examples of Algorithms

ex) Sort 13, 5, 8, 11, 20, 6.

Idea of insertion algorithm 삽입 정렬

Suppose $S = (s_1, s_2, \dots, s_n)$ is a sequence of numbers.

Insert s_i into $s_1, \dots, s_{i-1}$ that have been already sorted. 이미 정렬된

$i=1$. 13

$i=2$. 13, 5 → 5, 13.

$i=3$. 5, 13, 8 → 5, 8, 13.

$i=4$. 5, 8, 13, 11 → 5, 8, 11, 13.

$i=5$ 5, 8, 11, 13, 20

$i=6$, 5, 8, 11, 13, 20, 6

→ 5, 6, 8, 11, 13, 20. sorted!

Algorithm : Insertion Sort.

Input : S, n ($S=(S_1,\dots,S_n)$, n : 길이 Integer)

Output : S (sorted)

```
insertion_sort (S, n) {
  for i = 2 to n {
      val = S_i  ← 삽입하는 요소
      j = i - 1
      while (j ≥ 1 ∧ val < S_j) {
          S_{j+1} = S_j  → 크다면 오른쪽으로 옮김
          j = j - 1  → 반복 j≥1 까지
      }
      S_{j+1} = val
  }
  return S
}
```

13, 5, 8, 11, 20, 6.

j i

13, [5]

val

S_0 S_1 S_2 S_3 ...

[5] [13]

$i=3$. 비교

S_1 S_2 S_3

[5] [13] 8 → 5 8 13

val

5 < 8 → stop

$i=4$. 비교

S_1 S_2 S_3 S_4

5 8 13 11 → 5 8 11 13

val

8 < 11 → stop

$i=5$.

S_1 S_2 S_3 S_4 S_5

5 8 11 14 20 → 5 8 11 14 20

val

$i=6$.

S_1 S_2 S_3 S_4 S_5 S_6

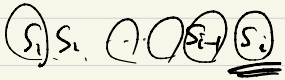
5 8 11 14 20 6 → 5 6 8 11 14 20

val

□

Algorithm : Insertion Sort.

```
Input : S, n    (S=(S1,...,Sn), n : integer)
Output : S (sorted)
insertion_sort (S, n) {
  for i = 2 to n {
      val = Si
      j = i-1
      while (j≥1 ∧ val<Sj) {
          Sj+1 = Sj
          j = j-1
      }
      Sj+1 = val
  }
  return S
}
```



The number of times the body of while loop is executed is
- the best case time : 0
- the worst case time : $i-1$

The for-loop is executed $n-1$ times.

the number of comparisons $val < S_j$. is
- best case : $n-1$ times. → while문 1번만
- worst case : $\sum_{i=2}^{n}(i-1) = \binom{n}{2} = \frac{n(n-1)}{2}$.

nC_2

§4.3. Analysis of Algorithms

Consider an algorithm with input size n.

$f(n)$ = the total number of executions.

($\propto$ 시간 $\propto$ $\frac{1}{\text{성능}}$)

We are not interested in computing exact value of $f(n)$. ($f(n)$의 정확한 값에는 관심 X)

We want to estimate $f(n)$ when n is large.

ex) n	$t(n)=60n^2+5n+1$	$60n^2$
10	6051	6,000
100	600,501	600,000
1000	60,005,001	60,000,000
10000	6,000,050,001	6,000,000,000

We say that $t(n)$ is of order n^2.

(we ignore constants)

$t(n) = \Theta(n^2)$.

(최고차항만 고려 (값이 같지는 않지만))

Def) f, g : functions $f: \mathbb{Z}^+ \to \mathbb{R}$

Worst time $g: \mathbb{Z}^+ \to \mathbb{R}$

We write $f(n) = O(g(n))$

(we say $f(n)$ is of order at most $g(n)$ 최대가 $g(n)$

or $f(n)$ is big-O of $g(n)$)

if there exists a constant $C_1 > 0$ such that

$|f(n)| \le C_1 |g(n)|$

for all but finitely many $n \in \mathbb{Z}^+$.

(Equivalently, it's true for all $n \ge N$ for some N.)

(We also say that it's true for sufficiently large n.)

충분히 큰 n에 대해

$f(n) = \Omega(g(n)) \iff f(n)$ is omega of $g(n)$

Best time $\iff \exists\, C_2 > 0$ such that

$|f(n)| \ge C_2 |g(n)|$

for all but finitely many n.

Note: $f(n) = \Omega(g(n)) \iff g(n) = O(f(n))$.

($\because (\Rightarrow)$ $|g(n)| \le \frac{1}{C_2} |f(n)| \Rightarrow g(n) = O(f(n))$.)

$f(n) = \Theta(g(n))$ 세타

차수가 같을 때 즉, grow (weight)가 동일할 때

$\Leftrightarrow$ $f(n)$ is theta of $g(n)$

$\Leftrightarrow$ $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

ex) $f(n) = 2n^2 + 10n + 20$.

Then $f(n) = \Theta(n^2)$.

pf). $|f(n)| = |2n^2 + 10n + 20| \overset{?}{\leq} C_1 n^2$

$\leq 100n^2 + 100n^2 + 100n^2 \leq 300\, n^2$ 이렇게 증명해도 되는데 간단하게 표현하자..

$|f(n)| \leq |2n^2 + 10n^2 + 20n^2| = 32|n^2|$

Taking $C_1 = 32$ we get $f(n) = O(n^2)$.

$|f(n)| \overset{?}{\geq} C_2 |n^2|$

$|f(n)| \geq |2n^2| = 2|n^2|$.

Taking $C_2 = 2$ we get $f(n) = \Omega(n^2)$.

Therefore $f(n) = \Theta(n^2)$. □

Thm If $p(n)$ is a polynomial in n of degree d then $p(n) = \Theta(n^d)$.

d차식

Pf) Let $p(n) = a_d n^d + \cdots + a_1 n + a_0$.

$$|p(n)| \le |a_d n^d| + \cdots + |a_1 n| + |a_0|$$
$$\le |a_d| |n^d| + \cdots + |a_1| |n^d| + |a_0| |n^d|$$
$$= (|a_d| + \cdots + |a_0|) |n^d|$$

Constant

$\Rightarrow p(n) = O(n^d)$.

$$|p(n)| = |a_d n^d + a_{d-1} n^{d-1} + \cdots + a_0|$$
$$= |a_d| \cdot \left| n^d + \frac{a_{d-1}}{a_d} n^{d-1} + \cdots + \frac{a_0}{a_d} \right|$$

Idea: $n^d = \frac{n^d}{2} + \frac{n^d}{2} = \frac{n^d}{2} + \frac{n^d}{2d} + \cdots + \frac{n^d}{2d}$ (d개)

Let's take n large enough so that $\frac{n}{2d} > \max\left(\left|\frac{a_{d-1}}{a_d}\right|, \ldots, \left|\frac{a_0}{a_d}\right|\right)$.

상수 0

fixed number.

Then $\frac{n^d}{2d} = \frac{n}{2d} \cdot n^{d-1} > \left|\frac{a_i}{a_d}\right| n^{d-1}$ $\forall i = 0, \ldots, d-1$

$\ge \left|\frac{a_i}{a_d}\right| n^i$

$n > 0$

Then, for $n > 2d \cdot \max\left(\left|\frac{a_{d-1}}{a_d}\right|, \ldots, \left|\frac{a_0}{a_d}\right|\right)$,

$$|p(n)| = |a_d| \cdot \left| n^d + \frac{a_{d-1}}{a_d} n^{d-1} + \cdots + \frac{a_0}{a_d} \right|$$
$$= |a_d| \cdot \left| \frac{n^d}{2} + \frac{n^d}{2d} + \cdots + \frac{n^d}{2d} + \frac{a_{d-1}}{a_d} n^{d-1} + \cdots + \frac{a_0}{a_d} \right|$$
$$= |a_d| \cdot \left| \underbrace{\frac{n^d}{2}}_{>0} + \underbrace{\left(\frac{n^d}{2d} + \frac{a_{d-1}}{a_d} n^{d-1}\right)}_{>0} + \cdots + \underbrace{\left(\frac{n^d}{2d} + \frac{a_0}{a_d}\right)}_{>0} \right|$$
$$> |a_d| \cdot \frac{n^d}{2} = \underbrace{\frac{|a_d|}{2}}_{= C_2} \cdot n^d.$$

$\Rightarrow p(n) = \Omega(n^d)$

$\Rightarrow p(n) = \Theta(n^d)$.

d차 > d-1차

$\frac{n^d}{2d}$가 흡수해서

뒤에까 음수여도

전체는 양수!

□

$f(n)$

ex) $1^k + 2^k + \cdots + n^k \le \underbrace{n^k + n^k + \cdots + n^k}_{n} = n^{k+1}$

Thus $f(n) = O(n^{k+1})$

$f(n) = 1^k + \cdots + \left\lceil \frac{n}{2} \right\rceil^k + \left(\left\lceil \frac{n}{2} \right\rceil + 1\right)^k + \cdots + n^k$

$\ge \left\lceil \frac{n}{2} \right\rceil^k + \left(\left\lceil \frac{n}{2} \right\rceil + 1\right)^k + \cdots + n^k$

$\left\lceil \frac{n+1}{2} \right\rceil \Leftarrow \left(n - \frac{n}{2} + 1\right)$

$\ge \left\lceil \frac{n}{2} \right\rceil^k \cdot \left\lceil \frac{n+1}{2} \right\rceil \ge \left(\frac{n}{2}\right)^{k+1} = \frac{1}{2^{k+1}} n^{k+1}$

Thus $f(n) = \Omega(n^{k+1})$

$\Rightarrow f(n) = \Theta(n^{k+1})$.

Note: In fact, $1^k + 2^k + \cdots + n^k$ is known to be a polynomial in n of degree $k+1$.

$\lg$ = Base 2, $\log$ = Base 10

ex) $\lg n! = \Theta(n \lg n)$

pf) $\lg n! = \lg 1 + \lg 2 + \cdots + \lg n$

$\le \lg n + \lg n + \cdots + \lg n = n \lg n$

$\Rightarrow \lg n! = O(n \lg n)$.

$\lg n! = \lg 1 + \lg 2 + \cdots + \lg n$

$\ge \lg \left\lceil \frac{n}{2} \right\rceil + \lg\left(\left\lceil \frac{n}{2} \right\rceil + 1\right) + \cdots + \lg n$

$\ge \left(\lg \left\lceil \frac{n}{2} \right\rceil\right)\left\lceil \frac{n+1}{2} \right\rceil \ge \left(\lg \frac{n}{2}\right) \cdot \frac{n}{2}$

$= \frac{n}{2}(\lg n - \lg 2) \ge \frac{n}{2}\left(\lg n - \frac{\lg n}{2}\right)$

$= \frac{n}{4} \lg n = \frac{1}{4} n \lg n$ ← $\frac{\lg n}{2} > \lg 2$

$\Rightarrow \lg n! = \Omega(n \lg n)$.

$\Rightarrow \lg n! = \Theta(n \lg n)$. □

ex) Find a theta notation in terms of n for the number of times the statement $x = x+1$ is executed.

```
for i = 1 to n        (i = 1 ~ n)
    for j = 1 to i    ← i times
        x = x+1
```

sol) For given i, the for-loop is executed i times.

$\Rightarrow$ $1+2+\cdots+n = \frac{n(n+1)}{2} = \Theta(n^2)$.

ex)
```
i = n
while (i ≥ 1) {
    x = x+1
    i = ⌊i/2⌋
}
```

If $n = 2^k$, then

$i = 2^k, 2^{k-1}, 2^{k-2}, \ldots, 2^1, 1, 0$ (2 times)

$k+1$ times.

If $2^{k-1} \le n < 2^k \Rightarrow$ k times.

$k-1 \le \lg n < k \Rightarrow \lfloor \lg n \rfloor = k-1$.

$\#\text{times} = k = \lfloor \lg n \rfloor + 1 = \Theta(\lg n)$.

⌊⌋, 상수는 무시됨

ex)
```
j = n
while (j ≥ 1) {
    for i = 1 to j
        x = x + 1      ← # times
    j = ⌊j/2⌋
}
```

Sol). for문 1st $j = n \longrightarrow n$ times
2nd $j = \lfloor n/2 \rfloor \longrightarrow \lfloor n/2 \rfloor$ times $\le n/2$
3rd $j = \lfloor \lfloor n/2 \rfloor / 2 \rfloor \le n/2^2$
⋮

등비급수합

$$\#\text{times} \le n + \frac{n}{2} + \frac{n}{4} + \cdots = n \cdot \frac{1}{1-\frac{1}{2}}$$

$\Rightarrow O(n)$.

Clearly, $\Omega(n)$. ← 1st for문에서 무조건 n번 돌기에

Therefore $\Theta(n)$.

ex)
```
i = 2
while (i < n) {
    i = i^2
    x = x + 1      ← #
}
```

$i = 2^{2^0} \rightsquigarrow 2^{2^1}$ (1step, 2step) $\rightsquigarrow (2^{2^1})^2 = 2^{2^2} \rightsquigarrow (2^{2^2})^2 = 2^{2^3}$ ...

$2^{2^2} \times 2^{2^2} = 2^{2^2+2^2} = 2^{2\times 2^2} = 2^{2^3}$

After kth step i becomes 2^{2^k}.

If $2^{2^{k-1}} < n \le 2^{2^k} \Rightarrow \#\text{times} = k$.

$2^{k-1} < \lg n \le 2^k$

$k-1 < \lg\lg n \le k$

$\#\text{times} = k = \lceil \lg\lg n \rceil = \Theta(\lg\lg n)$.

§4.4. Recursive Algorithms (재귀 알고리즘)

A recursive algorithm is an algorithm that contains a recursive function.

ex) $n!$ is obtained by multiplying n to $(n-1)!$

Algorithm: Computing n factorial.

```
input : n    (n≥0 integer)
output : n!
factorial (n) {
    if (n==0)  terminal condition!
        return 1
    return n * factorial(n-1)
}
```

ex) $f(n)$ = # ways to write n as a sum of 1s and 2s.

For example, $5 = 1+2+1+1 = 1+2+2 = 2+1+2$.

$f(1) = 1$: $1 = 1$

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

$f(3) = 3$: $3 = 1+1+1 = 2+1 = 1+2$

$f(4) = 5$: $4 = 1+1+1+1 = 1+1+2 = 1+2+1 = 2+1+1 = 2+2$

If $n \geq 3$, $f(n) = f(n-1) + f(n-2)$.

$n = \underbrace{1+\cdots}_{n-1}$ ⇝ $f(n-1)$

$= \underbrace{2+\cdots}_{n-2}$ ⇝ $f(n-2)$

$\Big\} \oplus$ n

$f(1), f(2), f(3), f(4), f(5), f(6), f(7)$

1 2 3 5 8 13 21 ...

피보나치 수열

$f(n)$ = the nth Fibonacci number

Algorithm Computing Fibonacci numbers.

Input : n ($n \geq 1$)

Output : f(n)

```
Fib(n) {
    if (n==1)
        return 1
    if (n==2)
        return 2
    return Fib(n-1)+Fib(n-2)
}
```

Example 4.4.7 Use mathematical induction to show that

$$\sum_{k=1}^{n} f_k = f_{n+2} - 1 \qquad \text{for all } n \geq 1.$$

SOLUTION For the basis step ($n = 1$), we must show that

$f_3 = 2$

$$\sum_{k=1}^{1} f_k = f_3 - 1.$$

Since $\sum_{k=1}^{1} f_k = f_1 = 1$ and $f_3 - 1 = 2 - 1 = 1$, the equation is verified.

For the inductive step, we assume case n

$$\sum_{k=1}^{n} f_k = f_{n+2} - 1$$

and prove case $n + 1$

$$\sum_{k=1}^{n+1} f_k = f_{n+3} - 1.$$

Now

$$\begin{aligned}\sum_{k=1}^{n+1} f_k &= \sum_{k=1}^{n} f_k + f_{n+1} \\ &= (f_{n+2} - 1) + f_{n+1} \qquad \text{by the inductive assumption} \\ &= f_{n+1} + f_{n+2} - 1 \\ &= f_{n+3} - 1.\end{aligned}$$

The last equality is true because of the definition of the Fibonacci numbers:

$$f_n = f_{n-1} + f_{n-2} \qquad \text{for all } n \geq 3.$$

Since the basis step and the inductive step have been verified, the given equation is true for all $n \geq 1$. ◀

12강 22

f_n = n을 만들 수 있는 방법의 수