(정의) (알고리즘 $A$의 복잡도)
**항상 끝나는**(type 1, **recursive**, **algorithm**, μ-recursive **total** function) 크기 $n$인 문제 $P$를 푸는 알고리즘(프로그램) $A$가 걸리는 시간이 $O(f(n))$일 때, **알고리즘** $A$의 시간 복잡도(time-complexity)를 $f(n)$이라 정의한다. 이 때 시간 복잡도 $f(n)$을 문제 $P$의 **알려진 최소**(worst-case) 복잡도라고 부른다.

(사실) 알고리즘의 복잡도 순서는 아래와 같다.
$O(1) \leq O(\log(n)) \leq O(n) \leq O(nlog(n)) \leq O(n^2) \leq \cdots \leq O(2^n) \leq O(n^n) \leq \cdots$

(정의) (문제 $P$의 복잡도)
어떤 문제 $P$를 푸는 **가장 빠른**(**optimal**) 알고리즘의 복잡도를 그 문제 $P$의 **최대 복잡도**(upper-bound)라고 부른다.

(예) Bubble sorting **알고리즘**의 복잡도는 $O(n^2)$, 이 경우 sorting 문제의 **알려진 최소** 복잡도는 $O(n^2)$라고 볼 수 있다. 그러나 quick sort 알고리즘의 복잡도가 $O(nlog(n))$이고, 이것은 bubble sorting **알고리즘**보다 더 **작고**, 이것이 **최소**임(**optimal**)임이 증명 되었으므로, sorting **문제**의 **최대 복잡도**는 $O(nlog(n))$으로 줄어든다.

(정의) **알고리즘**이 **결정적**(deterministic)할 때, 이 알고리즘의 복잡도를 **결정적** 복잡도라고 부르고, 알고리즘이 **비결정적**(non-deterministic)할 때, 이 알고리즘의 복잡도를 **비결정적** 복잡도라고 부른다.

(정의) 문제의 복잡도가 $O(n^k)$, $k \in N$ 또는 $O(p(n))$[1], 다항식 이하인(polynomial) **결정적** 알고리즘이 **알려진** 경우 이 문제를 **풀기 쉬운**(**tractable**) 문제라고 부른다. **가장 빠른**(**optimal**) **결정적** 알고리즘의 복잡도가 $O(2^n)$, 지수 이상인(exponential) 문제를 **풀기 어려운**(**intractable**) 문제라고 부른다.

(정의) **결정적** 알고리즘의 복잡도가 $O(p(n))$으로 알려진 문제들의 집합을 **P**, **비결정적** 알고리즘의 복잡도가 $O(p(n))$으로 알려진 문제들의 집합을 **NP**[2]라 한다.

(사실) **P** $\subseteq$ **NP**이지만 **P** $=$ **NP**나 **P** $\neq$ **NP**(**P** $\subset$ **NP**)라는 증명은 없다.

(정의) Polynomial time reduction
문제 $P_1$의 **모든** yes-instance, $Y_{P_1}$과 no-instance $N_{P_1}$을 문제 $P_2$의 모든 yes-instance, $Y_{P_2}$와 no-instance $N_{P_2}$로 바꾸어 주는 함수 $h: D_1 \rightarrow D_2$[3]를 수행하

---

1) $p(n) = a_0 n^k + a_1 n^{k-1} + \cdots + a_k$, $k \in N$, $0 \leq \forall i \leq k: a_i \in R$.

2) **NP** 문제의 알려진 worst-case 알고리즘은 decision tree를 이용하여 여러($k$) 개의 clone들을 반복적으로 배당하는 exponential 복잡도이다. Nondeterminism의 최대값이 $k$일 때, $O(k^{p(n)})$.

3) $Y_{P_1} \subseteq Y_{P_2} \wedge N_{P_1} \subseteq N_{P_2}$이므로 $P_2$가 $P_1$보다 더 어렵거나 느리거나 같다.

기 위한 **알고리즘**이 다항식 이하일 때, 문제 $P_1$의 복잡도를 문제 $P_2$로 복잡도로 **다항식 시간 안에서 줄였다**고 하고(**Polynomial time reduction**; PTR), $P_1 \leq_P P_2$로 쓴다.

(사실) $P_1 \leq_P P_2$라고 하자.

$$P_2 \in \mathbf{P} \Rightarrow P_1 \in \mathbf{P}.$$
$$P_1 \notin \mathbf{P} \Rightarrow P_2 \notin \mathbf{P}.$$

(정의) 문제 $P$가 아래 조건을 만족하면 NP-complete문제라고 부른다.

(1) $P \in \mathbf{NP}$.

(2) $\forall P' \in \mathbf{NP},\ P' \leq_P P$.

즉 NP-complete는 **NP** 중 **가장 어려운** 문제를 말한다.

NP-complete 정의에 **씨앗**이 되는 가장 이해하기 쉬운 NP-complete 문제는, $n$-변수를 가진 논리식을 true로 만드는 $n$개의 변수 값들이 있는가를 묻는 **satisfiability**(SAT)문제이다. NP-complete 정의의 시작을 임의의 **NP** 문제를 SAT로 Polynomial-time reduction하여 시작 한다[4]($\forall P \in \mathbf{NP},\ P \leq_P SAT$).

(정리 10.4) $P$가 NP-complete이고, $P \leq_P P_1$이면 $P_1$도 NP-complete이다.

(증명) $\forall P' \in \mathbf{NP},\ P' \leq_P P,\ P \leq_P P_1.\ \therefore P' \leq_P P_1$.

정리 10.4는 어떤 문제가 NP-complete임을 증명하는데 쓰이는 방법이다. 이 때 $P$는 SAT인 것이 보통이나 이미 알려진 어떤 NP-complete 문제이어도 좋다.

(정리 10.5) $P$가 NP-complete이고, $P \in \mathbf{P}$이면 $\mathbf{P} = \mathbf{NP}$이다.

(증명) $\forall P' \in \mathbf{NP},\ P' \leq_P P \in \mathbf{P},\ \mathbf{NP} \subseteq \mathbf{P}.\ \therefore \mathbf{P} = \mathbf{NP}$.

**NP** 문제의 **알려진 최소**(worst-case) 복잡도가 exponential[5]이므로 **intractable**로 볼 수 있으나, **NP**중 가장 어려운 NP-complete 문제 중 **하나**만 **P**로 밝혀지면 **모든 NP** 문제는 **P**이므로 **NP**는 **tractable**이라고 주장할 수 있다[6].

(정의) 문제 $P$가 아래 조건을 만족하면 NP-hard문제라고 부른다.

(2) $\forall P' \in \mathbf{NP},\ P' \leq_P P$.

NP-hard는 **NP**는 아닐 수 있지만, 모든 **NP**보다 더 **어려운** 문제를 말한다.

따라서 정리 10.5의 증명과 같이 어떤 NP-hard인 문제 $P$가 **P**임이 밝혀지면 $\mathbf{P} = \mathbf{NP}$이고 **tractable**이라고 주장할 수 있다.

---

4) 다음 장에서 나올 Cook's Theorem의 증명

5) Decision tree를 생각하자.

6) 그러나 그런 증명은 **아직** 없다.