* 1. Algorithms

**Problem**: question to which we seek an answer

ex) list is to be sorted in “nondecreasing order” instead of increasing order

**Solution**: 문제의 인스턴스에 대한 solution이 question의 answer이다.

**Algorithm**: 문제의 파라미터 값이 클 경우 인지적 능력만으론 불가능하다. To produce a computer program that can solve all instances of a problem, we must specify a general step-by-step procedure for producing the solution to each instance. This step-by-step procedure is called an algorithm. We say that the algorithm solves the problem.

* 1. the importance of developing efficient algorithms
     1. sequential search versus binary search

이진 탐색이 순차 탐색보다 빠름.

Ex) 32개의 요소가 있다고 가정. x라는 요소가 배열에 없을 경우 순차탐색은 32번 비교. X가 원소일 경우 비교횟수는 32보다 크지 않음. 이진 탐색의 경우 x와 가운데 원소를 비교하며 loop를 돌음. Loop당 한번의 비교. X가 모든 원소보다 클 경우 6번의 비교를 함. 이진 탐색에서의 최대 횟수 6 = log232 + 1.

순차는 n. 이진은 log2 n + 1

* + 1. the Fibonacci sequence

차트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

1.3analysis of algorithms

1.3.1 complexity analysis

**Input size**: it is easy to find a reasonable measure of the size of the input

Ex) the number of items in the array

**Basic operation**: instruction or group of instructions

ex) search 알고리즘에서 compare induction

**Time complexity analysis**: determination of how many times the basic operation is done for each value of the input size

* + 1. applying the theory

**overhead instructions**: initialization instructions before a loop, negligible  
**control instructions**: incrementing an index to control a loop

1.4 order

1.4.1 an intuitive introduction to order

**Pure quadratic**: contain no linear term

**Complete quadratic**: contain linear term

Quadratic term dominates function 3n^2 + 5n. Although the function is not a pure quadratic function, we can classify it with the pure quadratic functions. Throw away low-order terms when classifying complexity functions.

1.4.2 a rigorous introduction to order

도표이(가) 표시된 사진

자동 생성된 설명

도표이(가) 표시된 사진

자동 생성된 설명

**Big O**: asymptotic upper bound



If g(n) ∈ O(f(n)), we say that g(n) is big O of f(n)

ex) 5n 2 ∈ O(n^ 2)

**Omega**:



If g(n) ∈ Ω(f(n)), we say that g(n) is omega of f(n)

ex) n 3 ∈ Ω(n ^2)

**order**:







If g(n) ∈ Θ(f(n)), we say that g(n) is order of f(n)

Properties of Order:

1. g(n) ∈ O(f(n)) if and only if f(n) ∈ Ω(g(n)).

2. g(n) ∈ Θ(f(n)) if and only if f(n) ∈ Θ(g(n)).

3. If b > 1 and a > 1, then loga n ∈ Θ(logb n).

4. If b > a > 0, then This implies that all exponential complexity functions are not in the same complexity category.



5. For all a > 0 This implies that n! is worse than any exponential complexity function.



6. Consider the following ordering of complexity categories: where k > j > 2 and b > a > 1. If a complexity function g(n) is in a category that is to the left of the category containing f(n), then



7. If c ≥ 0, d > 0, g(n) ∈ O(f(n)), and h(n) ∈ Θ(f(n)), then

