2. 컴퓨터 생각하기

(1) 알고리즘

◆ 학 습 목 표

(1) 알고리즘의 개념을 이해할 수 있다.

(2) 알고리즘의 만족조건을 파악할 수 있다.

(3) 문제의 해결방안을 알고리즘으로 세울 수 있다.

(4) 알고리즘의 표현방법 3가지를 이해할 수 있다.

(5) 특정한 예시를 통해서 문제 해결 방안을 구상할 수 있다.

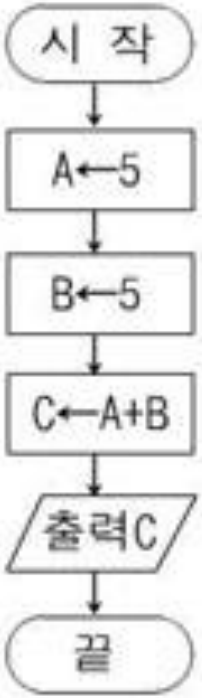
(6) 평성평가를 통해서 학생 스스로 순서도를 작성할 수 있다.

◆ 오늘 배울 내용

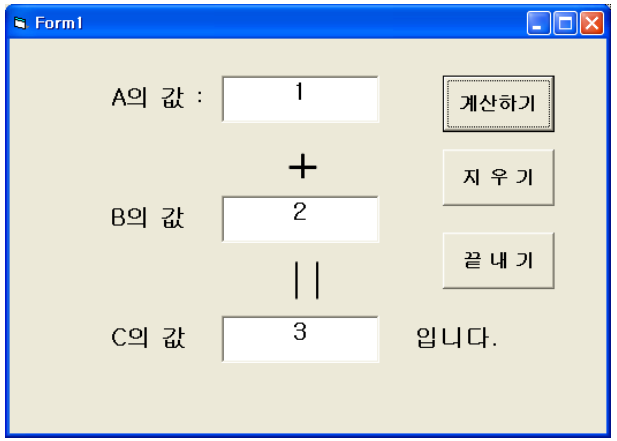
1. 자연언어 알고리즘

① 알고리즘 기술 시작
② 숫자 5를 A변수에 대입한다.
③ 숫자 4를 B변수에 대입한다.
④ A변수의 값과 B변수의 값을 더해서 C변수에 대입한다.
⑤ C변수의 값을 출력한다.
⑥ 알고리즘 기술 끝

2. 순서도 알고리즘



3. 비쥬얼 C를 이용한 알고리즘



1. 알고리즘이란?

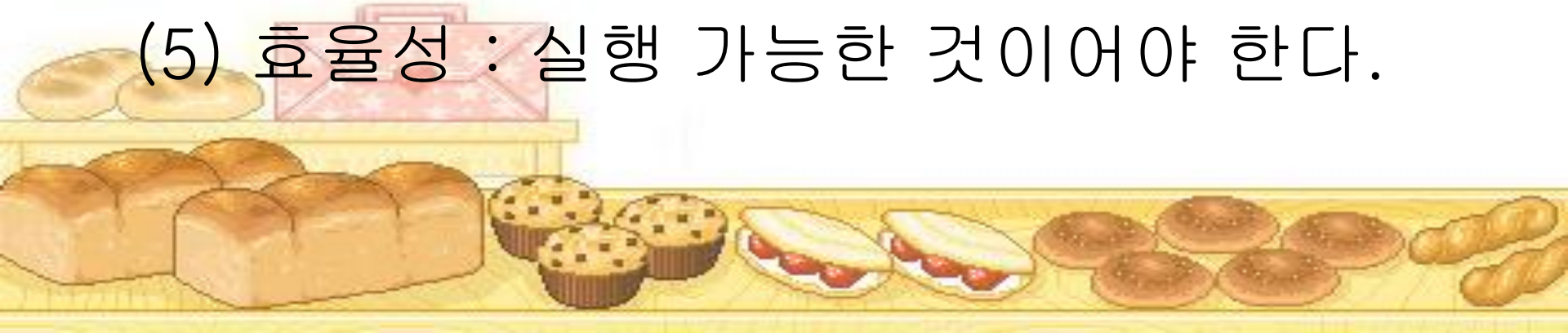
- 알고리즘이란?

알고리즘은 문제를 해결하기 위한 절차나 방법이다.

- “Algorithm”이란 아랍의 수학자, 알콰리즈미의 이름에서 유래되었다.

◆ 알고리즘의 만족조건

(1) 입　력 : 외부에서 제공되는 입력
(2) 출　력 : 하나이상의 출력
(3) 명확성 : “ A+B-2*5%2-C”
(4) 유한성 : ex1) 1 부터 100까지 더하라.
ex2) 2의 배수를 구하여라.
(5) 효율성 : 실행 가능한 것이어야 한다.



2. 알고리즘 표현하기.

1. 자연언어를 이용하는 방법

2. 순서도를 이용하는 방법

3. 특정 프로그래밍 언어를 이용하는 방법

2. 알고리즘 표현하기

(1) 자연언어

Ex)매일 아침 학교에 등교하는 알고리즘을 언어로 표현한다.

① 기상한다.

② 세면을 한다.

③ 아침 식사를 한다.

④ 과제물이나 준비물을 빠뜨렸는지 확인하고
빠뜨렸으면 반드시 챙긴다.

⑤ 교통 수단을 선택한다.(지하철, 버스, 도보, 택시, 자전거)

⑥ 학교에 도착한다.

⑦ 교실에 입실한다.

(1) 자연언어

① 자연언어의 장점과 단점

- 장점 : 이해하기가 쉽다.

- 단점 : 작업과정을 명확하게 기술하기 어렵다.
  컴퓨터로 구현할 수 없다.

(2) 순서도

- 순서도는 문제해결 하는 방법과 절차와 내용을 기호로 나타낸 것이다.

- 순서도는 ISO에서 정한 표준 기호를 사용한다.

(2) 순서도

① 순서도의 장점과 단점

- 장점 : 이해하기가 쉽다.
- 단점 : 특별한 구조나 세밀한 부분까지 표현하기가 어렵다.

  변경이 곤란하다. 대규모 알고리즘을 표현하기 곤란하다.

(2) 순서도

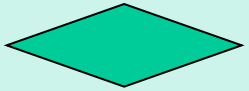
② 순서도의 기호 배우기

- 순서도에서 사용하는 기호들

순서도기호 배우기

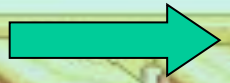

순서도의 시작과 끝을 나타낸다.


자료에 값을 변경하거나 계산한다.


선택을 해야 하는 경우에 어느 것을 택할 것인지를 결정한다.


자료의 값을 출력한다.


자료의 입력 및 출력을 표시한다.


실행 순서를 나타낸다.

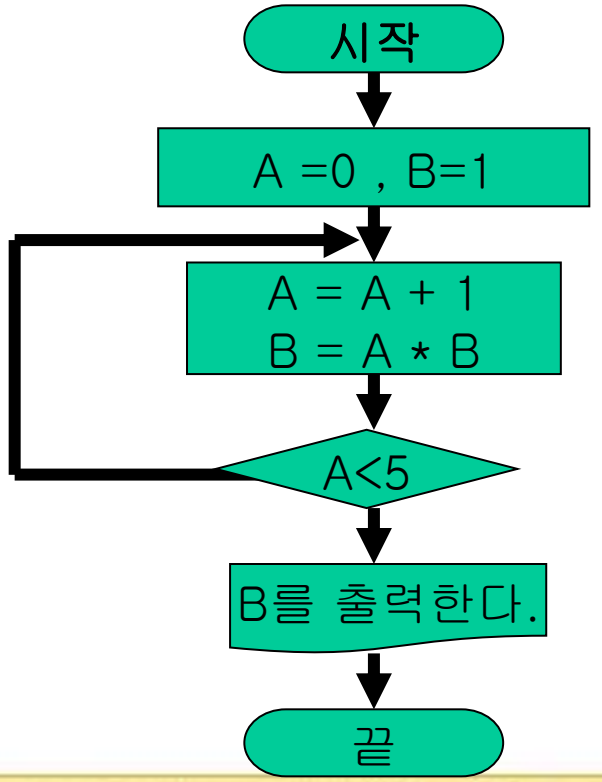
(2) 순서도

③ 순서도 만들기(예시1)

❖ 자연 언어

① A변수에 0을 대입한다.
② B변수에 1을 대입한다.
③ A변수를 1증가시킨다..
④ B변수에 A*B값을 넣는다..
⑤ A이 5이 될 때까지 이 단계를 반복한다.
⑥ N이 5이 되면 합계 B를 출력한다.
⑦ 여기서 B는 1에서 5까지의 곱이 된다

❖ 순서도



(2) 순서도

③ 순서도 만들기(예시2)

❖ 자연 언어

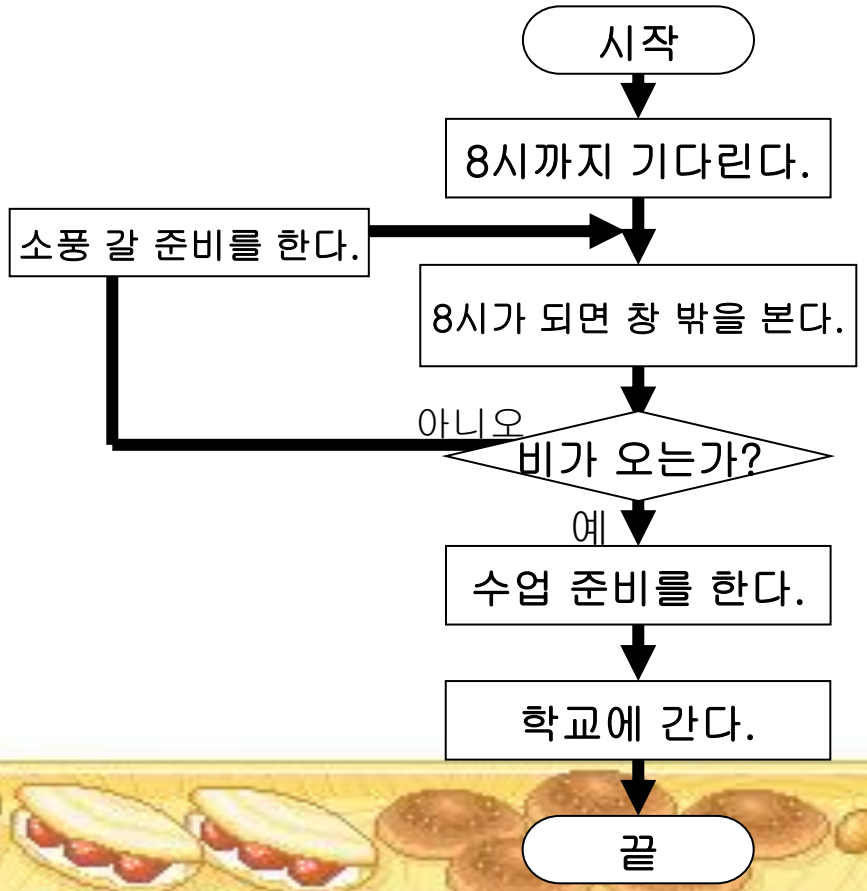
① 아침 8시가 되기를 기다린다.

② 8시가 되면 창 밖을 본다.

③ 만일 비가 오면 수업 준비를 하고 학교에 간다.

④ 그렇지 않다면 소풍 갈 준비를 하고 학교에 간다.

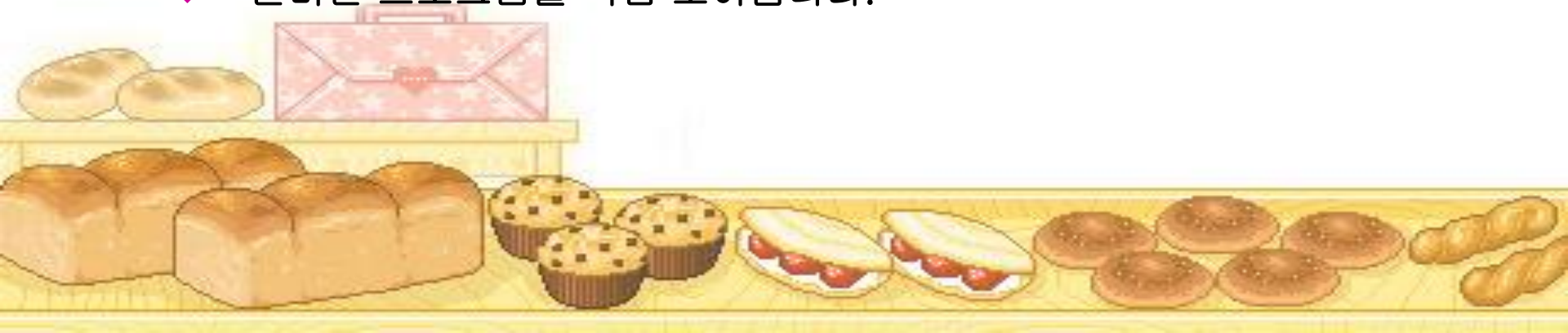
❖ 순서도



(3) 프로그램을 이용하는 방법

- 프로그래밍 언어를 이용하여 알고리즘을 만들 수 있다.
- C언어, 비쥬얼 베이직 등

❖ 준비한 프로그램을 직접 보여줍니다.



목차

학습목표

- 알고리즘의 정의와 필요성을 인지한다.
- 아주 기초적인 알고리즘 수행시간 분석을 할 수 있도록 한다.
- 점근적 표기법을 이해한다.
- 달력 프로그램으로 알고리즘을 생각해본다.

알고리즘이란?

- 문제 해결 절차를 체계적으로 기술한 것
- 문제의 요구조건
  - 입력과 출력으로 명시할 수 있음
  - 알고리즘은 입력으로부터 출력을 만드는 과정을 기술

입출력의 예

- 문제
  - 100명의 학생의 시험점수의 최대값을 찾으라
- 입력
  - 100명의 학생들의 시험점수
- 출력
  - 위 100개의 시험점수들 중 최대값

알고리즘 공부의 목적

- 특정한 문제를 위한 알고리즘의 습득
- 체계적으로 생각하는 훈련
- 지적 추상화의 레벨 상승
  - Intellectual abstraction
  - 연구나 개발에 있어 정신적 여유를 유지하기 위해 매우 중요한 요소

바람직한 알고리즘

- 명확해야 한다
  - 이해하기 쉽고 가능하면 간명하도록
  - 지나친 기호적 표현은 오히려 명확성을 떨어뜨림
  - 명확성을 해치지 않으면 일반언어의 사용도 무방
- 효율적이어야 한다
  - 같은 문제를 해결하는 알고리즘들의 수행시간이 수백만 배 이상 차이 날 수 있다

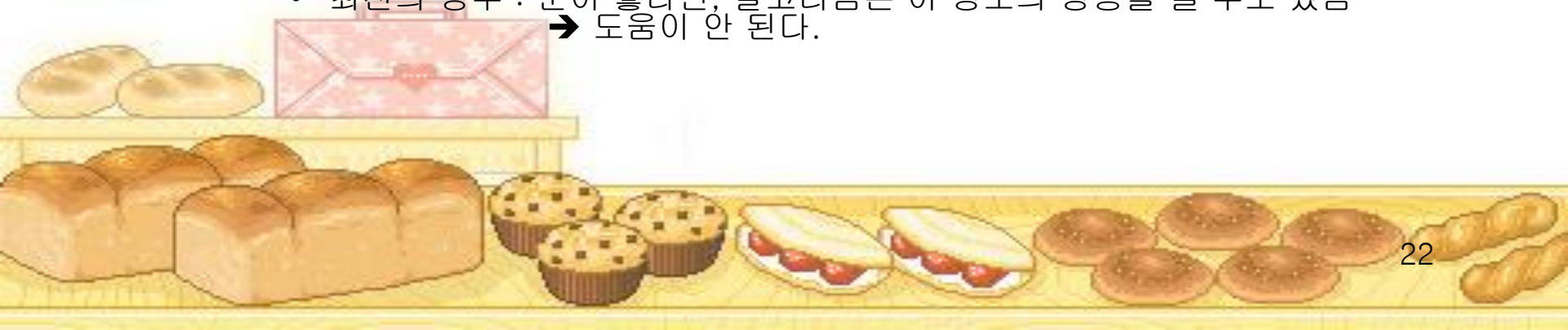
알고리즘의 성능에 대하여

- 알고리즘의 성능을 가리는 5가지 측정 기준
  - 정확성 : 정확하게 동작하는가?
  - 작업량 : 얼마나 적은 연산을 수행하는가?
  - 메모리 사용량 : 얼마나 적은 메모리를 사용하는가?
  - 단순성 : 얼마나 단순한가?
  - 최적성 : 더 이상 개선할 여지가 없을 만큼 최적화되어 있는가?

알고리즘 수행 시간의 분석

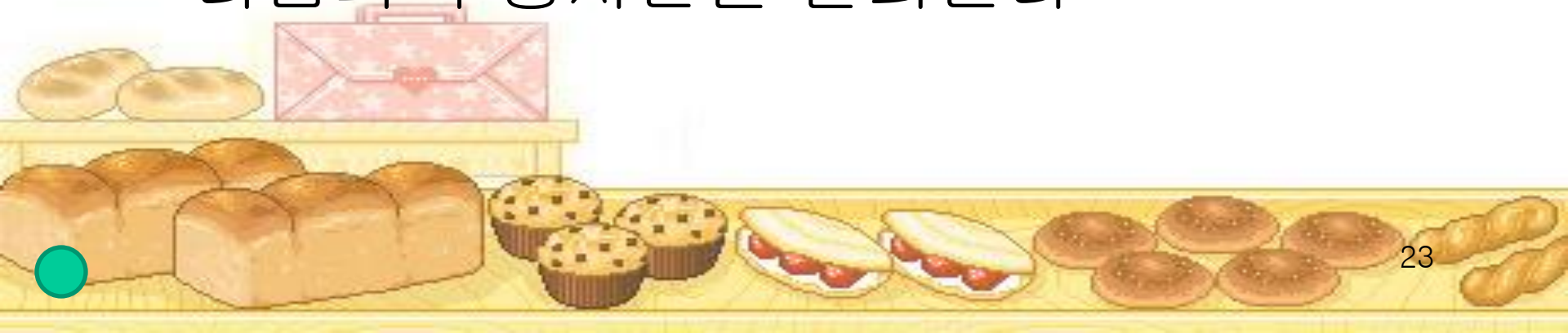
- 왜 스톱워치로는 알고리즘 수행 시간을 측정 못하는가?
  - 똑같은 알고리즘이라 하더라도 성능이 좋은 CPU와 성능이 나쁜 CPU에서 각각 테스트를 해보면 알고리즘이 종료되는 시간에 차이가 발생할 것이기 때문
  - 알고리즘을 실제로 구현하기 전에는 성능을 예측할 수 없음. 다시 말해, 소프트웨어 설계자는 성능을 미리 알 수 없음

- 알고리즘 수행시간으로써 활용될만한 적절한 후보는 "작업량"
  - 최악의 경우, 평균의 경우, 최선의 경우에 대한 작업량을 측정한다.
    - 최악의 경우 : 어떤 경우에도 이만큼의 성능은 "보장"
    - 평균의 경우 : 대체적으로 알고리즘이 내는 성능
    - 최선의 경우 : 운이 좋다면, 알고리즘은 이 정도의 성능을 낼 수도 있음
      ➔ 도움이 안 된다.



알고리즘의 수행시간

- 알고리즘의 수행시간을 좌우하는 기준은 다양하게 잡을 수 있음
  - 예: for 루프의 반복횟수, 특정한 행이 수행되는 횟수, 함수의 호출횟수, …
- 몇 가지 간단한 경우의 예를 통해 알고리즘의 수행시간을 살펴본다



알고리즘의 수행시간

```
sample1(A[ ], n)
{
        k = n/2 ;
        return A[k] ;
}
```

✓ n에 관계없이 상수 시간이 소요된다.

알고리즘의 수행시간

```
sample2(A[ ], n)
{
        sum ← 0 ;
        for i ← 1 to n
                sum← sum+ A[i] ;
        return sum ;
}
```

↘ n에 비례하는 시간이 소요된다.

알고리즘의 수행시간

```
sample3(A[ ], n)
{
        sum ← 0 ;
        for i ← 1 to n
                for j ← 1 to n
                        sum← sum+ A[i]*A[j] ;
        return sum ;
}
```

✓ n^2에 비례하는 시간이 소요된다.

알고리즘의 수행시간

```
sample4(A[ ], n)
{
        sum ← 0 ;
        for i ← 1 to n
                for j ← 1 to n {
                        k ← A[1 ... n] 에서 임의로 n/2 개를 뽑을 때 이 중 최대값 ;
                        sum ← sum + k ;
                }
        return sum ;
}
```

- ✓ n^3에 비례하는 시간이 소요된다.

알고리즘의 수행시간

```
sample5(A[ ], n)
{
        sum ← 0 ;
        for  i ← 1 to n
                for  j ←  i+1 to n
                        sum← sum+ A[i]*A[j] ;
        return sum ;
}
```

✓ n^2에 비례하는 시간이 소요된다

알고리즘의 수행시간

```
factorial(n)
{
        if (n=1) return 1 ;
        return n*factorial(n-1) ;
}
```

✓ n에 비례하는 시간이 소요된다.

알고리즘의 분석

- 무결성 확인
- 자원 사용의 효율성 파악
  - 자원
    - 시간
    - 메모리, 통신대역, …

알고리즘의 분석

- 크기가 작은 문제
  - 알고리즘의 효율성이 중요하지 않다
  - 비효율적인 알고리즘도 무방
- 크기가 충분히 큰 문제
  - 알고리즘의 효율성이 중요하다
  - 비효율적인 알고리즘은 치명적
- 입력의 크기가 충분히 큰 경우에 대한 분석을 점근적 분석이라 한다