한라대학교
정보통신소프트웨어
교수 이경호

알고리즘과 알고리즘 성능 평가

알고리즘

- 어떤 문제를 해결하기 위해 명확히 정의된(well-defined) 유한 개의 규칙과 절차의 모임.
  - 알고리즘은 기술된 문자가 수학적인지 비수학적인지, 또 사람의 손으로 문제를 해결할 것인지, 컴퓨터로 해결할 것인지에 관계없이 적용된다.
  - 특히 컴퓨터로 문제를 푸는 경우에는 알고리즘을 형식적으로 표현하는 것이 프로그램을 작성하는 데 중요한 요소가 된다.
  - 이 알고리즘의 좋고 나쁨에 따라 같은 결과를 구하는 처리에서도 시간이나 조작성에 큰 차이가 날 수가 있다.

- 알고리즘은 다음 조건을 만족해야 한다.

① 입력 : 외부에서 제공되는 자료가 있을 수 있다.

② 출력 : 적어도 한 가지 결과가 생긴다.

③ 명백/명확성 : 각 명령들은 애매모호하지 않아 다른 해석이 있을 수 없으며, 같은 입력일 경우 항상 같은 결과가 나온다.

④ 유한성 : 기술한 명령대로 수행하면 한정된 단계를 처리한 후에 종료된다.

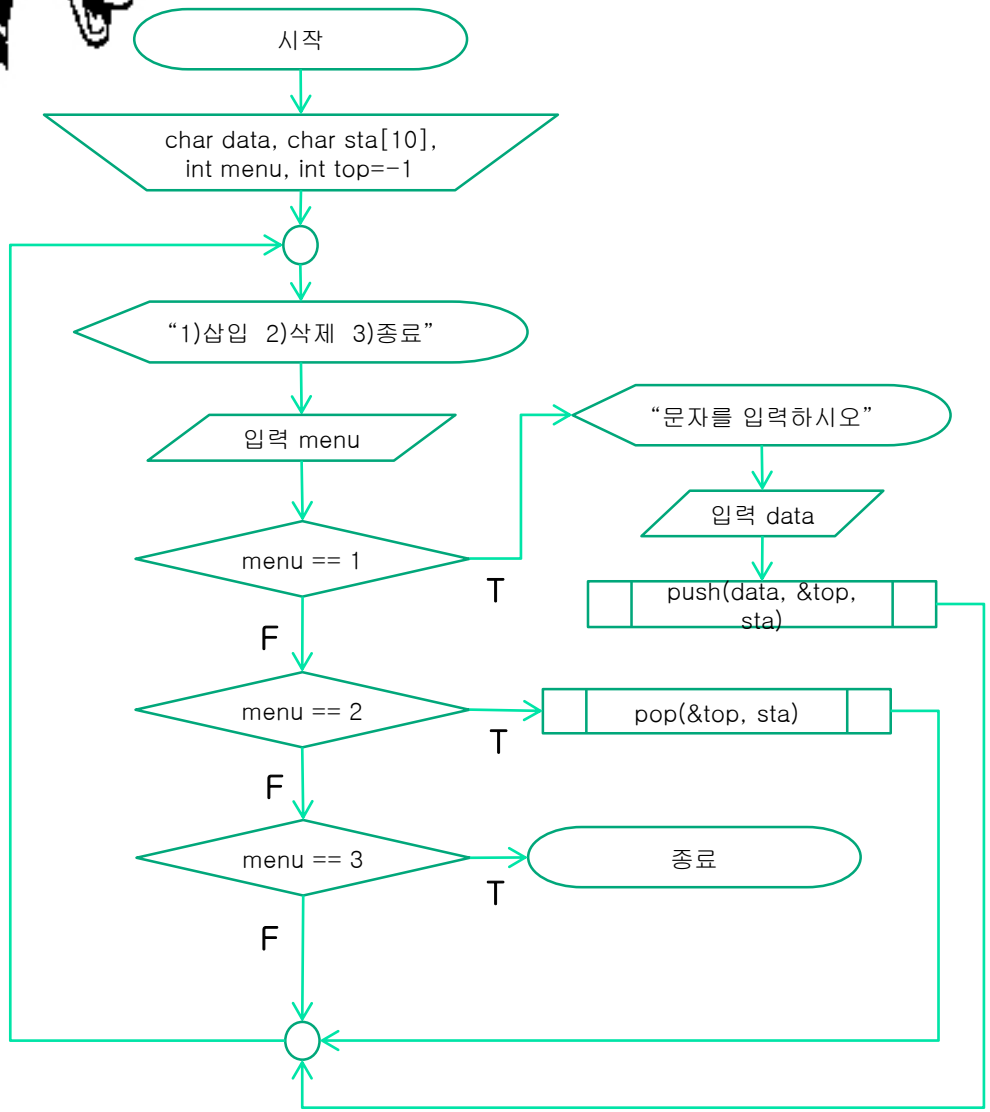
⑤ 효과성 : 기술한 모든 명령들은 실행 가능한 것이어야 한다.

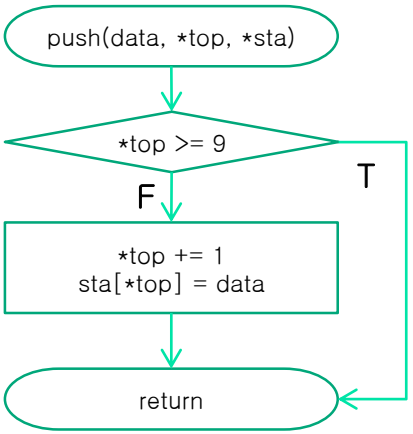
알고리즘의 표현 방법

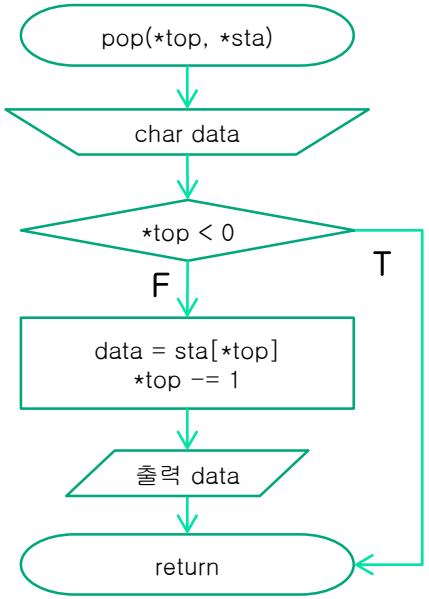
*. 앞 5가지 알고리즘 조건을 지키는 한 어떤 표현도 가능

배열로 구현한 스택 흐름도

시작(main) 함수의 무한 반복은 while(1) 이나 for(;;) 으로 구현 가능합니다.







가상 코드(pseudo code)

```
void printPerm (a_prefix_string, a_set_S) {
   if |S| is 0
      print the prefix string;
   else
      for each element x in S
         printPerm(the_prefix_string + x, S - {x} );
}
```

알고리즘 작성 보편 방법

- 문제 파악
  - 문제를 잘 읽고 문제에서 요구하는 바를 확정한다.
    - 불확실한 부분은 문의 하거나 문제를 정의하기 위한 가정을 한다.
- 개념적 단계
  - 문제를 컴퓨터에 표현(변수와 자료구조, 허용 연산의 운용 전략:알고리즘)
  - 작업의 모듈화 및 중심 작업 파악과 중심 작업을 기준으로 한 작업 간 관계 파악
    - 문제들의 트리 구조가 형성.
  - 문제 해결을 위한 절차를 분석해 보면, 1) 어떤 형태로 자료를 표현하고(자료구조), 여기에 2) 어떤 순서로 일을 하면 결과를 얻을 수 있는지(알고리즘) 파악된다.
  - 알고리즘은 '순차적', '선택적', '반복적' 처리가 어우러져 전체 절차를 만든다.
  - 파악된 문제를 '시작 상태 ' 에서 '어떤 절차들'을 거쳐 '완료 상태 ' 로 갈 수 있는지 분석한다. ('A~하고, B~하고, C~하면 된다.' 형식이 되도록 노력해 볼 것)
  - A,B,C… 각각을 '순차적', '선택적', '반복적' 처리를 통해 일 처리가 될 수 있도록 기술한다.
- 논리적 단계
  - 프로그램으로 변경하기 쉬운 형태(흐름도, 가상코드 등)로 정확히 표현하고, 가상 실행을 하여 정확성을 확인해 본다.

알고리즘의 성능 평가

- 질문
  - 세 사람이 같은 문제를 해결하는 프로그램을 작성하였다. 그런데 각각 다른 컴퓨터에서 프로그램을 수행하여 수행 시간을 측정하였다. 이 때 나온 시간으로 프로그램의 성능을 논의 한다면 무엇이 문제일까?
  - 일부지만 같은 데이터(모든 데이터를 수행 할 수 없는 한계)로 같은 컴퓨터에서 수행하여 나온 결과로 보편적 성능으로 논의 한다면 문제는 없을까?
  - 수행 시간만이 프로그램의 성능을 말할 수 있는 지표인가?
  - 기계에 독립적이고, 자료에 독립적인 성능 평가 방법은 없을까?

알고리즘의 성능분석

- 알고리즘의 성능 분석 기법
  - 알고리즘 성능 평가의 두 관점
    1. **시간 복잡도 분석**: 수행 시간 분석
    2. **공간 복잡도 분석**: 수행 시 필요로 하는 메모리 공간의 양 분석

  - 알고리즘의 복잡도 분석
    - 알고리즘이 수행하는 연산의 횟수를 측정하여 비교하되
    - 연산의 횟수는 입력에 의존적이 포함된 n의 함수 형태로 표현

시간의 복잡도 분석의 예

- n을 n번 더하는 문제:

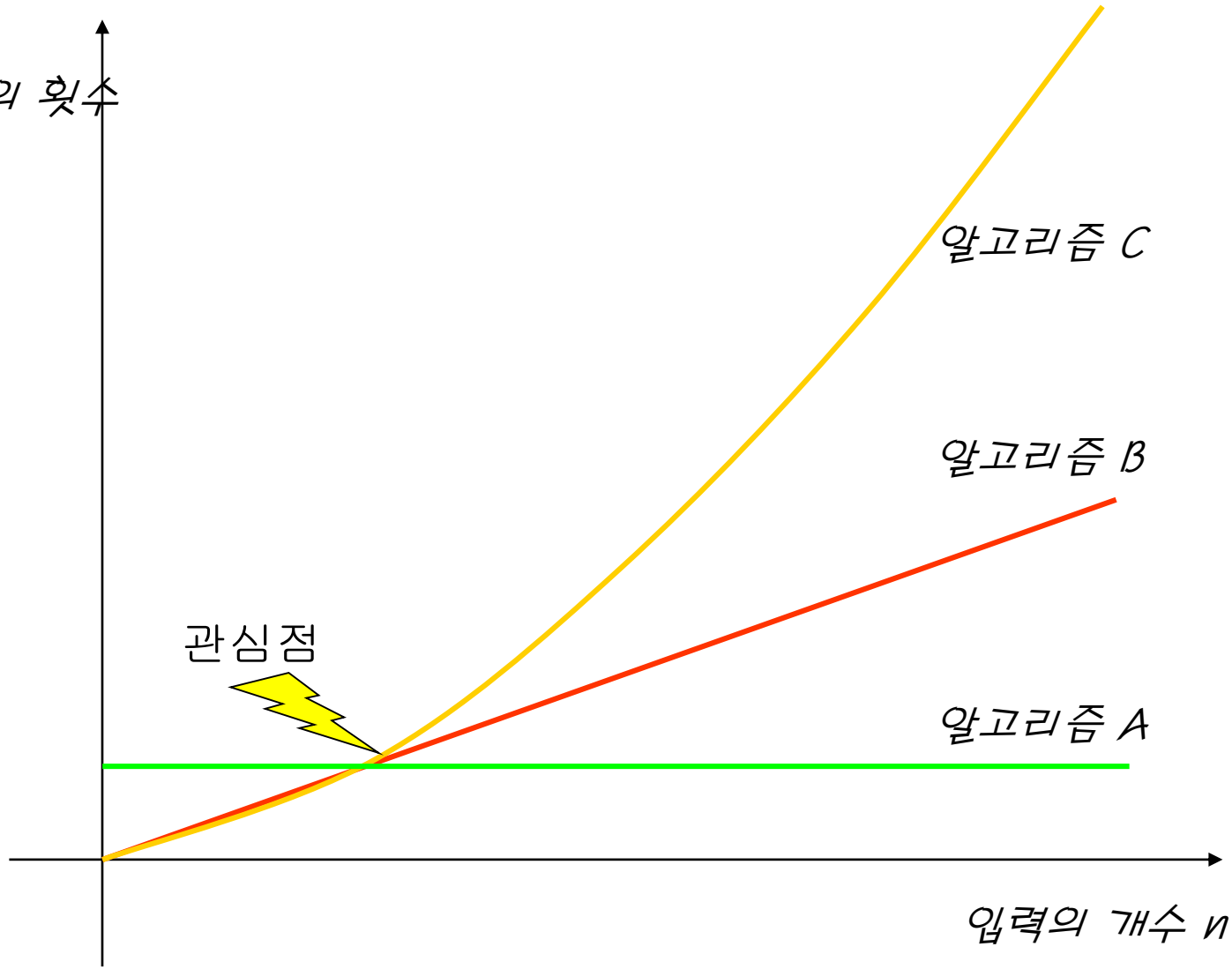
  각 알고리즘이 수행하는 연산의 개수를 계산하는 함수 구성.
  단 for 루프 제어 연산은 고려하지 않음.

알고리즘 A	알고리즘 B	알고리즘 C

sum ←n*n;	sum ← 0;
	for i ← 1 to n do
	  sum ←sum + n;
		sum ← 0;
		for i←1 to n do
		  for ←1 to n do
		  sum ←sum + 1;

	알고리즘 A	알고리즘 B	알고리즘 C
대입연산	1	n + 1	n*n + 1
덧셈연산		n	n*n
곱셈연산	1		
나눗셈연산			
전체연산수	2	2n + 1	$2n^2 + 1$

입력에 의존한 연산 수 변화 표현



시간의 복잡도 표기 고찰

- 성능 평가 시 다양한 다항식으로 표현되는 문제를 어떻게 처리해야 할까?
  - 자료와 기계에 독립적으로 성능은 평가하며 단순화 분류 아이디어는 무엇일까?
- 자료의 개수가 많은 경우에는 차수가 가장 큰 항이 가장 영향을 크게 미치고 다른 항들은 상대적으로 무시될 수 있다.
- (예) n=1,000 일 때, 아래 $T(n)$의 값은 1,001,001이고 이중에서 첫 번째 항인 n^2의 값이 전체의 약 99%인 1,000,000이고 두 번째 항의 값이 1000으로 전체의 약 1%를 차지한다.
- 따라서 보통 시간복잡도 함수에서 가장 영향을 크게 미치는 항만을 고려하면 충분하다.

n=1000인 경우

$T(n) = n^2 + n + 1$

99%

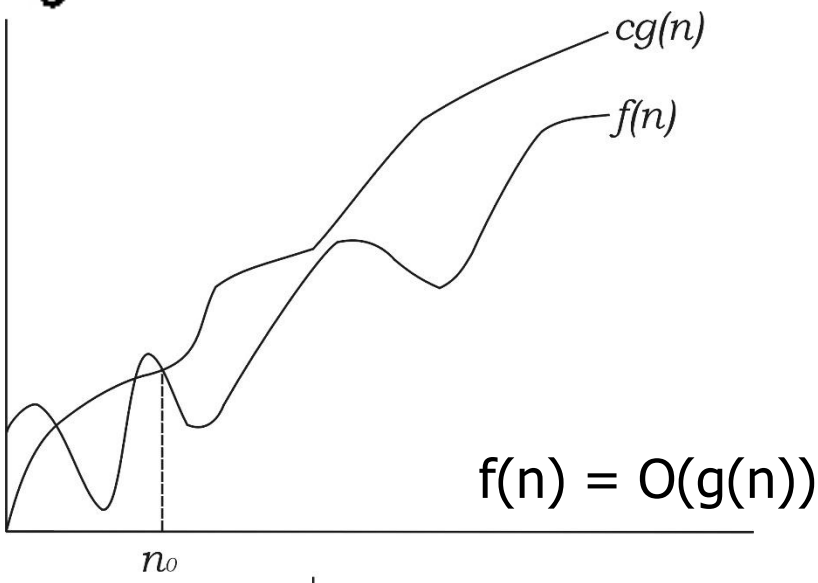
1%

입력의 개수 n

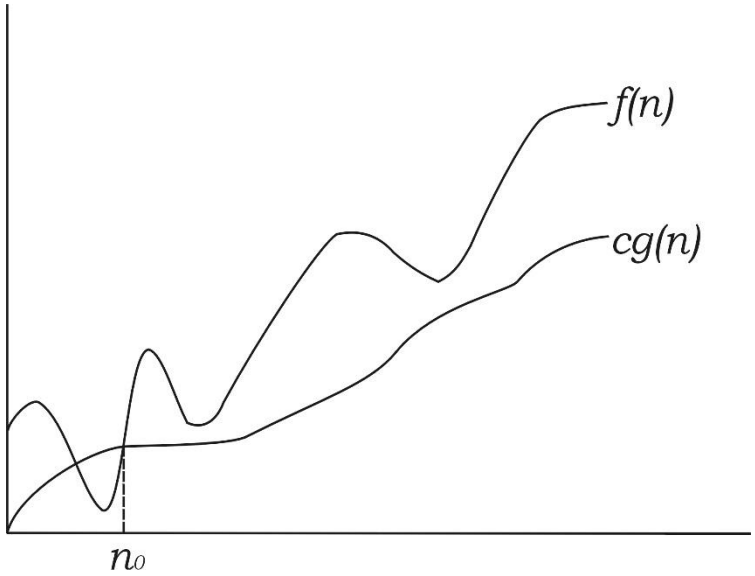
알고리즘의 성능 평가

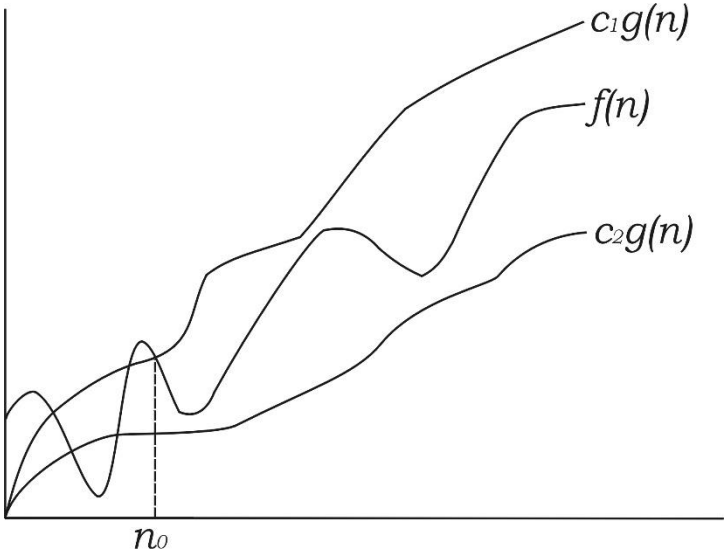
- $O(\ g(n)\ )$
  - Tight *or* loose upper bound
  - $O(\ g(n)\ ) = \{\ f(n) \mid \exists\, c > 0,\ n_0 \geq 0$ s.t. $\forall\, n \geq n_0,\ cg(n) \geq f(n)\ \}$
  - $f(n) \in O(g(n)$을 관행적으로 $f(n) = O(g(n))$이라고 쓴다.
  - 직관적 의미
    - $f(n) = O(g(n)) \Rightarrow f$는 g보다 빠르게 증가하지 않는다
    - 상수 비율의 차이는 무시
- $\Omega(\ g(n)\ )$
  - Tight *or* loose lower bound
- $\Theta(\ g(n)\ )$
  - Tight bound

알고리즘의 성능 평가



f(n) = O(g(n))





빅오 표기법의 종류

- O(1) : 상수형
- O(log n) : 로그형
- O(n) : 선형
- O(n log n) : 로그선형
- $O(n^2)$: 2차형
- $O(n^3)$: 3차형
- $O(n^k)$: k차형
- $O(2^n)$: 지수형
- O(n!) : 팩토리얼형