[1] 알고리즘 기초

1) 알고리즘이란?

□ 알고리즘(algorithm)

- 주어진 문제(problem)에 대한 답(solution)을 찾는 단계적 절차(procedure) 혹은 기법
- 어원 : Al-Khwarizmi (알-콰리즈미 : 십진법을 발명한 9세기 페르시아의 수학자)
- "An algorithm is a finite, deterministic, and effective problem-solving method suitable for implementation as a computer program" *(Sedgewick)*
- 프로그램에서 어떤 특정한 일을 수행하는 개별적인 모듈(module)과 유사
- 자료구조(data structure)
  - ✓ 정보(information)를 저장하는 방법
  - ✓ 추상자료형(ADT:Abstract Data Type) : 데이터에 대한 동작(연산)
  - ✓ 알고리즘을 구현하기 위해 사용하는 추상자료형(ADT)

□ 알고리즘의 학습 목적

- 문제의 추상화(problem abstraction)
- 데이터의 추상화(data abstraction)
- 주어진 문제를 해결하는 알고리즘의 설계 기법
- 알고리즘의 효율성 분석 기법
- 중요한 문제에 대한 효율적 알고리즘의 이해
- 적절하고 효율적인 자료구조를 이용한 알고리즘의 구현
- 문제의 난이도에 대한 이해 : NP-완비성 개념

□ 알고리즘에서 다루는 중요한 문제들

- 고전적 문제 : 정렬문제, 선택문제, 검색문제 등
- 최적화 문제 : 배낭문제, 스케줄링 문제, 행렬연산문제, 파일압축문제, LCS 문제 등
- 그래프 문제 : 그래프 탐색 문제, 최소신장트리 문제, 최단경로 문제, 순회외판원 문제 등

□ 대표적 알고리즘 설계 기법

- 분할정복(divide-and-conquer)
- 동적프로그래밍(dynamic programming)
- 탐욕적 기법(greedy algorithm)

2) 알고리즘의 표현

□ 알고리즘의 표현

- 주로 의사코드(pseudocode)로 표현
  - ✓ 특정 언어의 문법에 구애받지 않고, 영어 혹은 한글로도 작성 가능
  - ✓ 명확하고 간결하게 작성

✓ 일반적으로 입출력과 오류처리는 포함하지 않는다.
✓ 블록은 중괄호(brace) 혹은 들여쓰기(indentation)을 사용

- 컴퓨터 지향적이어야 한다.
  ✓ 적절한 데이터 구조에 대한 고려가 필요
  ✓ 프로그램으로의 구현(implementation)을 염두에 두어야 함

□ 예제 : 최댓값 구하기

- 구체적(concrete) 문제 : 알고리즘 중간고사 성적 1등은 몇 점인가?
- 추상화(abstract) 문제 : n개의 숫자들 중 최댓값 구하기
- 문제는 입력(input)과 출력(output)으로도 명시할 수 있다.
  ✓ 입력 : n개의 숫자로 이루어진 배열 $S[\,]$
  ✓ 출력 : 배열 $S[\,]$의 원소들 중 최댓값
- 문제 해결 방법 : 배열의 각 원소를 순서대로 스캔하면서 최댓값을 업데이트한다.
- 알고리즘 vs 코드

자연어로 표현한 알고리즘	의사코드

tmp에 배열의 첫 번째 원소를 저장한다.
배열의 각 원소를 하나씩 옆으로 옮겨가며,
  만일 비교하는 원소가 tmp보다 크면,
  tmp를 바꾼다.
tmp가 최대값이다.
	max(S[1 … n])
	  tmp ← S[1]
	  for i=2, …, n
	  if(S[i]>tmp) tmp ← S[i]
	  tmp를 반환

C 코드	Java 코드

double max(double a[], int n) {
  int i;
  double tmp = a[0];
  for(i=1:i<n;i++)
  if(a[i] > tmp)
  tmp=a[i];
  return tmp;
}
	public static double max(double[] a) {
	  int n = a.length;
	  double tmp = a[0];
	  for(int i=1;i<n;i++)
	  if(a[i] > tmp)
	  tmp = a[i];
	  return tmp;
	}

3) 알고리즘의 분석

□ 알고리즘의 분석

- 알고리즘의 정확성을 분석하는 것이 아니라 효율성을 분석하는 것
- 알고리즘이 자원(resource)을 얼마나 소모하는지 분석
  ✓ 자원 : 소요시간, 메모리(공간), 통신대역 등

□ 시간복잡도 분석 (time complexity analysis)

- 알고리즘의 실행시간(running time) 효율성 분석
  ✓ 알고리즘의 실제 실행시간을 측정하는 방법 : 실행환경의 영향을 많이 받음
  ✓ 시간복잡도 분석은 물리적 시간 대신 명령문의 실행 횟수를 분석
  ✓ 실행시간은 입력(input)에 따라 달라진다.
- 입력 크기(input size)

- ✓ 검색 혹은 정렬 문제 : 배열에 저장된 원소(element)의 수, 흔히 n으로 표시
- ✓ 그래프 문제 : 정점과 간선의 수

▪ 시간복잡도 분석
- ✓ 입력크기에 따른 중요 연산의 수행 횟수를 계산
- ✓ 주로 $T(n)$으로 표기
- ✓ 모든 경우 시간복잡도

▪ 동일한 입력 크기라도 입력 사례에 따라 실행 횟수가 달라지는 경우
- ✓ 최악의 경우 시간복잡도 $W(n)$: 최대 실행 횟수
- ✓ 평균의 경우 시간복잡도 $A(n)$: 평균 실행 횟수
- ✓ 최선의 경우 시간복잡도 $B(n)$: 최소 실행 횟수

□ 복잡도 카테고리(complexity category)

▪ 점근적 분석(asymptotic analysis)
- ✓ 입력 크기 n이 충분히 클 때 실행 시간이 증가하는 비율을 표시
- ✓ 빅오(O) 표기법 : 시간 복잡도의 최고차항을 상수를 무시하고 표시한 것
- ✓ 예) $T(n) = 3n^2 + 2n\lg n + 5n + 3 = O(n^2)$
- ✓ 예) $T(n) = n^5 + 2^n + 100n + 5 = O(2^n)$
- ✓ 다차(polynomial) 시간 알고리즘 vs 지수(exponential) 시간 알고리즘

▪ $k > j > 2$, $b > a > 1$일 때,

$O(1) < O(\lg n) < O(n) < O(n\lg n) < O(n^2) < O(n^j) < O(n^k) < O(a^n) < O(b^n) < O(n!)$

□ 반복 알고리즘의 시간 복잡도 분석

▪ 배열의 맨 마지막 원소 출력하기

```
sample1(A[1...n])
   A[n]을 출력;
```

- ✓ 배열은 임의접근이 가능하므로 입력 크기 n과 상관없는 상수시간이 소요됨
- ✓ $T(n) = c = O(1)$

▪ 배열의 각 원소 출력하기

```
sample2(A[1...n])
   for i=1...n
      A[i]를 출력;
```

- ✓ 단순 반복(simple iteration)은 반복 횟수에 비례하는 시간이 소요됨
- ✓ $T(n) = n = O(n)$

▪ 배열을 아래 형태로 출력하기

A[1]

A[1] A[2]

A[1] A[2] A[3]

....

```
sample3(A[1...n])
  for i=1...n
    for j=1...i   //i번 반복
      A[j]를 출력;
      줄바꾸기;
```

✓ 중첩 반복(nested iteration)의 경우, 먼저 내부 반복(inner loop)의 횟수를 계산한 후 이에 대하여 외부 반복(outer loop) 횟수를 계산

✓ $T(n) = 1+2+\cdots n = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$

▪ 아래 패턴으로 배열 원소 출력하기

A[n] A[n/2] A[n/4] … A[1]

```
sample4(A[1...n])
  j = n;
  while (j >= 1)
    A[j]를 출력;
    j= ⌊ j/2 ⌋ ;
```

✓ $n = 2^k$ ($\Leftrightarrow k = \log_2 n$)인 경우, $j = 2^k, 2^{k-1}, \ldots, 2, 1$이므로 반복회수는 $(k+1)$번

✓ 예를 들어, $n = 2^5$이라고 가정하면, $j = 2^5, 2^4, \ldots, 2, 1$이므로 반복회수는 6번

✓ 반복의 횟수가 일정 비율로 줄어드는 경우(예를 들어, 절반씩), 차수는 로그(logarithmic)!!

✓ $T(n) = O(\log_2 n) = O(\lg n)$

4) 알고리즘의 구현

□ C와 Java의 공통점

▪ 컴파일 언어

✓ C : 컴파일하면 목적(object) 파일 생성

✓ Java : 컴파일하면 자바 바이트코드(bytecode) 생성

▪ 정적 타이핑(static typing) 언어 : 명시적 선언, 컴파일 시간에 타입 체크

□ C와 Java의 차이점

▪ 프로그래밍 패러다임

✓ C : 절차적 프로그래밍(procedural programming)

✓ Java : 객체지향 프로그래밍(OOP:Object-Oriented Programming)

▪ 데이터의 처리

✓ C : 데이터와 절차를 분리, 함수의 인수로 데이터를 주고받음

✓ Java : 데이터와 절차를 묶어서 처리

□ 함수 혹은 메서드

▪ 주어진 인수(argument)들을 사용하여 정해진 문장들을 실행하고 보통 결과값을 반환(return)

▪ 값에 의한 호출 vs 참조에 의한 호출

- 자바 메서드 : static method vs instance method
  - ✓ 인스턴스 메서드(instance method)
    - class의 instance에 속한 메서드
    - 호출 : 인스턴스명.메서드명()
  - ✓ 정적 메서드(static method) = 함수
    - class에 속한 메서드로서 class 객체 생성 없이도 호출이 가능한 메서드
    - 호출 : 클래스명.메서드명() ex) Math.min()
- 함수(메서드)의 중복 정의(overloading)
  - ✓ 같은 이름으로 여러 함수 정의
  - ✓ 인수 개수와 타입, 반환형이 달라도 된다.
  - ✓ C는 함수의 중복 정의를 허용하지 않고, C++는 허용
  - ✓ 자바는 중복 정의를 허용

□ 알고리즘의 실행시간(단위:초) 측정하기

C	clock_t start, end;
	double elapsed = 0;
	start = clock();
	mergeSort(...);
	end = clock();
	elapsed = (end – start) / (double)CLOCKS_PER_SEC;
		#include <time.h>

Java	long start = System.currentTimeMillis();
	mergeSort(...);
	long end = System.currentTimeMillis();
	double elapsed = (end – start) / 1000.0;	

□ $[0,1)$ 난수 발생하기

C	a[i] = rand() / (double)RAND_MAX;	#include <stdlib.h>
Java	a[i] = Math.random();	