2장
알고리즘과 성능 분석

알고리즘과 문제 해결

- **알고리즘 (algorithm)**
  - 특정 문제를 해결하기 위해 기술한 일련의 명령문
- **프로그램 (program)**
  - 알고리즘을 컴퓨터가 이해하고 실행할 수 있는 특정 프로그래밍 언어로 표현한 것
    - Program = algorithm + data structures
- **알고리즘의 요건**
  - 완전성과 명확성
    - 수행 단계와 순서가 완전하고 명확하게 명세되어야 함
    - 순수하게 알고리즘이 지시하는 대로 실행하기만 하면 의도한 결과가 얻어져야 함
  - 입력과 출력
    - 입력 : 알고리즘이 처리해야 할 대상으로 제공되는 데이타
    - 출력 : 입력 데이타를 처리하여 얻은 결과
  - 유한성
    - 유한한 단계 뒤에는 반드시 종료

순환

- **순환 (recursion)**
  - 정의하려는 개념 자체를 정의 속에 포함하여 이용
  - 종류
    - 직접 순환 : 함수가 직접 자신을 호출
    - 간접 순환 : 다른 제 3의 함수를 호출하고 그 함수가 다시 자신을 호출
  - 순환 방식의 적용
    - 분할 정복(divide and conquer)의 특성을 가진 문제에 적합
    - 어떤 복잡한 문제를 직접 간단하게 풀 수 있는 작은 문제로 분할하여 해결하려는 방법.
    - 분할한 문제는 원래의 문제와 그 성질이 같기 때문에 푸는 방법도 동일
  - 순환 함수의 명령문 골격
    - if (simplest case) then solve directly
      else {make a recursive call to a simpler case};

순환 함수의 예 (1)

- **Factorial (n!)**
  - 정의
    - n=0 : 1
    - $n \geq 1$: $n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1 = n \cdot (n-1)!$
  - 순환 함수의 표현

```
factorial(n)
    // n은 음이 아닌 정수
    if (n ≤ 1) then return 1
    else return (n · factorial(n –1));
end factorial()
```

  - 비순환 함수로도 표현 가능
    - 표현이 좀 길지만 제어의 흐름이나 실행 과정을 쉽게 이해할 수 있음

Factorial.java   Factorial1.java

순환 함수의 예 (2)

- **이원 탐색**
  - 정의 : 주어진 탐색키 key가 저장된 위치(인덱스) 찾아내는 방법
    - key = a[mid] : 탐색 성공, return mid
    - key < a[mid] : a[mid]의 왼편에 대해 이진탐색
    - key > a[mid] : a[mid]의 오른편에 대해 이진탐색
  - 순환 함수의 표현

```
binsearch(a[], key, left, right)
// a[mid] = key인 인덱스 mid를 반환
        if (left ≤ right) then {
                mid ← (left + right) / 2;
                case {
                   key = a[mid] : return (mid);
                   key < a[mid] : return (binsearch(a, key, left, mid –1));
                   key > a[mid] : return (binsearch(a, key, mid+1, right));
                }
        }
        else return -1;   // key 값이 존재하지 않음
 end binsearch()
```

탐색

Ars.java

Employee.java

EmployeeSeqSearchTest.java

EmployeeBinSearchTest.java

Student.java

StudentSeqSearchTest.java

StudentBinSearchTest.java

EmployeeSeqSearchTest1.java

EmployeeBinSearchTest1.java

StudentSeqSearchTest1.java

StudentBinSearchTest1.java

순환 함수의 예 (3)

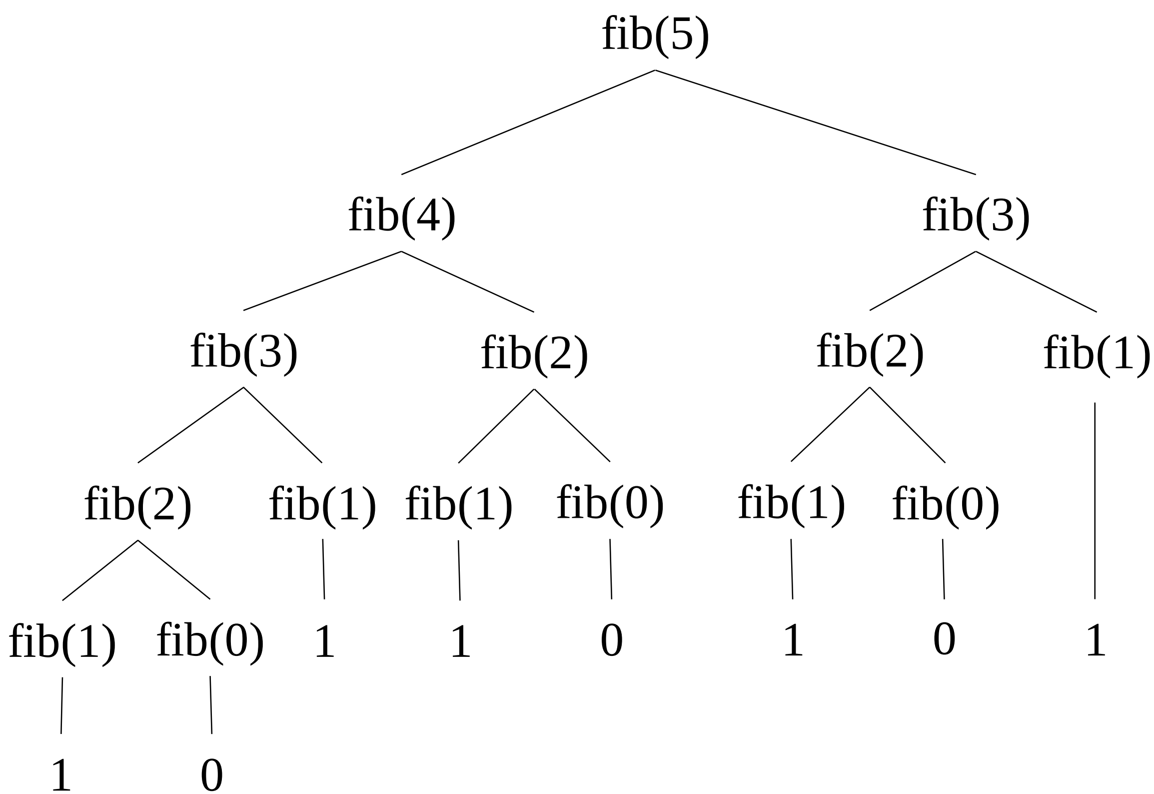
- **피보나치 수열 (Fibonacci sequence)**
  - 각 항은 바로 직전 두 항의 합으로 만들어짐
  - 순환 정의
    - n=0 : $f_0 = 0$
    - n=1 : $f_1 = 1$
    - $n \geq 2 : f_n = f_{n-1} + f_{n-2}$
  - 순환 함수의 표현

```
fib(n)
        if (n ≤ 0) then return 0;
        else if (n=1) then return 1
        else return (fib(n –1)  + fib(n –2));
end fib()
```

  - 이 fib() 함수는 순환 호출이 증가하여 실행시간으로 볼 때 반복적 함수보다 비효율적임
    - 순환적 정의가 순환적 알고리즘으로 문제를 해결하는 데 최적의 방법이 아닐 수도 있다는 예

순환 함수의 예 (4)

- **fib(5)의 실행과정**



성능 분석 (1)

◆ 프로그램의 평가 기준

- 원하는 결과의 생성 여부
- 시스템 명세에 따른 올바른 실행 여부
- 프로그램의 성능
- 사용법과 작동법에 대한 설명 여부
- 유지 보수의 용이성
- 프로그램의 판독 용이

◆ 프로그램의 성능 평가

- 성능 분석 (performance analysis)
  - 프로그램을 실행하는데 필요한 시간과 공간의 추정
- 성능 측정 (performance measurement)
  - 컴퓨터가 실제로 프로그램을 실행하는데 걸리는 시간 측정

성능 분석 (2)

- **공간 복잡도 (space complexity)**
  - 프로그램을 실행시켜 완료하는데 소요되는 총 저장 공간
  - $S_p = S_c + S_e$
    - S_c : 고정 공간
      - 명령어 공간, 단순 변수, 복합 데이타 구조와 변수, 상수
    - S_e : 가변 공간
      - 크기가 변하는 데이타 구조와 변수들이 필요로 하는 저장 공간
      - 런타임 스택(runtime stack)을 위한 저장 공간

- **시간 복잡도 (time complexity)**
  - 프로그램을 실행시켜 완료하는데 걸리는 시간
  - $T_p = T_c + T_e$
    - T_c : 컴파일 시간
    - T_e : 실행 시간
      - 단위 명령문 하나를 실행하는데 걸리는 시간
      - 실행 빈도수 (frequency count)

피보나치수의 예 (1)

- **n번째 항을 계산하는 반복식 프로그램**

```
fib_i(n)
    if (n < 0) then stop;   // error 발생
    if (n ≤ 1) then return n;
    fn2 ← 0; fn1← 1;
    for (i←2; i<=n; i←i+1) do {
        fn ← fn1 + fn2;
        fn2 ← fn1;
        fn1 ← fn;
    }
    return fn;
end fib_i()
```

피보나치수의 예 (2)

- f_n 계산을 위한 실행 빈도수

명령문(행)	실행 빈도수	명령문(행)	실행 빈도수
1	1	8	n-1
2	0	9	n-1
3	1	10	n-1
4	0	11	0
5	1	12	1
6	1	13	0
7	n		

- 실행 시간
  - 4n+2 : O(n)

- **"함수 fib_i()의 시간 복잡도는 O(n)이다"라고 말함**

점근식 표기법 (1)

- **점근식 표기법(Asymptotic notation)**
  - Big-Oh (O)
  - Big-Omega (Ω)
  - Big-Theta (Θ)

- **Big-Oh (O)**
  - f, g가 양의 정수를 갖는 함수일 때, 두 양의 상수 a, b가 존재하고, 모든 $n \geq b$에 대해 $f(n) \leq a \cdot g(n)$ 이면, $f(n) = O(g(n))$
  - 예
    - $f(n) = 3n + 2$: $f(n) = O(n)$ (a=4, b=2)
    - $f(n) = 1000n^2 + 100n - 6$: $f(n) = O(n^2)$ (a=1001, b=100)
    - $f(n) = 6 \cdot 2^n + n^2$: $f(n) = O(2^n)$ (a=7, b=4)
    - $f(n) = 100$: $f(n) = O(1)$ (a= 100, b=1)

점근식 표기법 (2)

- 연산 시간 그룹
  - 상수시간 : O(1)
  - 로그시간 : O(*log*n)
  - 선형시간 : O(n)
  - n로그시간 : O(n*log*n)
  - 평방시간 : $O(n^2)$
  - 입방시간 : $O(n^3)$
  - 지수시간 : $O(2^n)$
  - 계승시간 : O(n!)

- 연산 시간의 크기 순서

  $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$

- **$O(n^k)$: polynomial time**

점근식 표기법 (3)

Big-Omega (Ω)

- f, g가 양의 정수를 갖는 함수일 때,
  두 양의 상수 a, b가 존재하고, 모든 $n \geq b$에 대해 $f(n) \geq a \cdot g(n)$ 이면,
  $f(n) = \Omega(g(n))$
- 예
  - $f(n) = 3n + 2$: $f(n) = \Omega(n)$ (a=3, b=1)
  - $f(n) = 1000n^2 + 100n - 6$: $f(n) = \Omega(n^2)$ (a=1000, b=1)
  - $f(n) = 6 \bullet 2^n + n^2$: $f(n) = \Omega(2^n)$ (a=6, b=1)

점근식 표기법 (4)

◆ Big-Theta(Θ)

- f, g가 양의 정수를 갖는 함수일 때,
  세 양의 상수 a, b, c가 존재하고, 모든 $n \geq c$에 대해
  $a \cdot g(n) \leq f(n) \leq b \cdot g(n)$ 이면, $f(n) = \Theta(g(n))$
- 예
  - $f(n) = 3n + 2$: $f(n) = \Theta(n)$ (a=3, b=4, c=2)
  - $f(n) = 1000n^2 + 100n - 6$: $f(n) = \Theta(n^2)$ (a=1000, b=10001, c=100)
  - $f(n) = 6 \cdot 2^n + n^2$: $f(n) = \Theta(2^n)$ (a=6, b=7, c=4)

fib_i()의 예

- 점근적 복잡도의 계산

세그먼트	점근적 복잡도
1~6	O(1)
7~11	O(n)
12~13	O(1)

$T(Fib\_i) = O(1) + O(n) + O(1) = O(n)$

- 예
  - $O(1) + O(1) = O(1)$
  - $O(1) + O(n) = O(n)$
  - $O(n) + O(n) = O(n)$
  - $O(n) + O(n^2) = O(n^2)$
  - $O(1) + O(n) + O(n^2) = O(n^2)$

- 실행 환경
  - 최선의 경우 (best case)
  - 최악의 경우 (worst case)