12.1 알고리즘이란 무엇인가?

교과목명	이산수학	분반		담당교수	김 외 현
학부(과)		학번		성명	

정의 알고리즘

주어진 문제를 해결하기 위해 필요한 여러 가지 단계들을 체계적으로 명시해 놓은 것

참고 알고리즘의 특성

(1) 입력: 문제를 풀기 위한 입력이 있어야 함
(2) 출력: 문제를 해결했을 때 답이 나와야 함
(3) 유한성: 유한 번의 명령이 수행된 후에는 끝나야 함
(4) 정확성: 주어진 문제를 정확하게 해결해야 함
(5) 확정성: 각 단계가 실행된 후에는 결과가 확정됨
(6) 일반성: 같은 유형의 문제에 모두 적용됨
(7) 효율성: 정확하면서도 효율적이어야 함

참고 알고리즘과 프로시저

(1) 알고리즘
어떤 문제 해결에 대한 Yes 또는 No가 분명한 문제 해결 방법론
(2) 프로시저
어떤 문제 해결에 대한 Yes나 No, 또는 프로그램이 종료될지 아닐지도 모르는 문제 해결 방법론

참고 알고리즘

- 순서도, 유사 코드, 언어 등 여러 가지 방법으로 표현될 수 있음
- 누구나 이해할 수 있도록 명확하게 기술하는 것이 매우 중요함
- 어떤 문제가 주어졌을 때 그 문제를 풀기 위한 방법론인 알고리즘이 항상 하나만 있는 것은 아님
- 컴퓨터 프로그램의 경우에는 어떤 알고리즘이 가장 효율적인지를 선택되어야 함
- 일반적으로 수행 시간, 메모리 용량, 자료의 종류, 프로그래머의 성향에 따라 가장 알맞은 알고리즘을 선택됨

참고 생활 속 알고리즘의 예

① 아침에 눈을 떠서 학교에 가려고 할 때 언제 일어나서 식사를 하고, 버스나 지하철을 이용하여 정해진 시간에 학교에 도착할 수 있을지를 계획하는 일
② 초등학교 시절부터 덧셈과 곱셈을 하는 방법, 최대공약수를 구하는 방법, 소수를 구하는 방법 등의 기초적인 수학의 연산
③ 전자레인지 등 전자제품의 사용 설명서에 나타난 사용 방법
④ 라면을 맛있게 끓이려고 할 때 물의 양, 불의 세기, 끓이는 시간, 스프를 넣는 시기 등의 단계적 음식 조리법
⑤ 바둑이나 게임을 잘할 수 있는 알파고와 같이 생각하는 방법론
⑥ 어떤 목적지로 이동하려고 할 때 어느 지하철역에 환승하는 것이 더 효율적인지 등

예제

1. 다음 중 알고리즘에 대한 설명으로 옳지 않은 것은?
① 문제의 해법을 단계적으로 나누어서 설명한 것이다.
② 알고리즘은 프로시저와 같은 것이다.
③ 컴퓨터 언어로 바꾸면 프로그램이 될 수 있다.
④ 문제 해결을 위한 방법을 실행 순서대로 적은 것이다.

2. 알고리즘이 가져야 할 7가지 주요 특성을 적어보자.

12.2 알고리즘의 표현과 효율성

교과목명	이산수학	분반		담당교수	김 외 현
학부(과)		학번		성명	

정의 **유사코드**

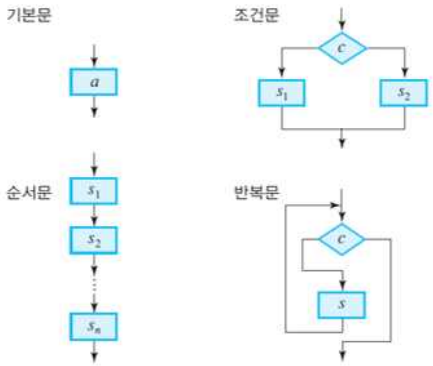
알고리즘을 프로그래밍 언어와 유사한 형태로 풀어서 써 놓은 것으로서, 사고의 흐름을 간결하고 효과적으로 전달하는 표현

- 알고리즘을 개략적으로 표현하는데 쓰임
- 유사코드로 적는 것은 알고리즘의 각 단계를 차례로 적는 것임
- 일반적으로 C언어나 자연어와 유사하게 기술 가능함

정의 **순서도**

처리하고자 하는 문제를 분석한 후 처리 순서를 단계화하고, 상호간의 관계를 표준 기호를 사용하여 입력, 처리, 결정, 출력 등의 박스와 연결선으로 일목요연하게 나타낸 도표

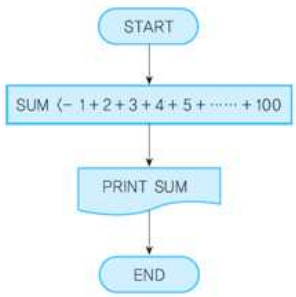
참고 **순서도의 유형**



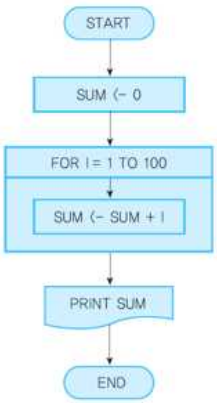
참고 **알고리즘의 효율성 비교**

1부터 100까지 정수의 합을 구하는 방법

(1) 일반적인 계산 방법



(2) for 문을 사용한 계산 방법



예제

3. 다음 중 알고리즘의 표현 방법과 가장 거리가 먼 것은?

① 순서도
② 함수
③ 유사코드
④ 언어

12.3 알고리즘 분석

교과목명	이산수학	분반		담당교수	김 외 현
학부(과)		학번		성명	

정의 **효율성 분석**

비용이 적게 드는 알고리즘을 찾기 위해 필요

- 효율성을 분석하기 위해서는 알고리즘 수행 시 필요한 시간복잡성과 공간복잡성의 두 가지 요소를 검토함
- 수행 시간과 그에 따르는 기억 장소의 크기는 알고리즘이 처리하는 입출력 자료의 크기에 따라 달라짐
- 일반적으로 알고리즘을 분석할 때 입력의 개수를 n으로 생각하고 효율성을 그에 대한 함수로 나타냄

참고 **알고리즘의 수행 횟수와 입력 크기에 따라 걸리는 시간**

수행 시간 \ 입력 크기	5	10	50	100	1000
1	10^{-6}	10^{-6}	10^{-6}	10^{-6}	10^{-6}
$\log_2 n$	2×10^{-6}	3×10^{-6}	6×10^{-6}	7×10^{-6}	10^{-5}
n	5×10^{-6}	10^{-5}	5×10^{-5}	10^{-4}	10^{-3}
$n\log_2 n$	10^{-5}	3×10^{-5}	3×10^{-4}	7×10^{-4}	10^{-2}
n^2	3×10^{-5}	10^{-4}	3×10^{-3}	10^{-2}	1
n^3	10^{-4}	10^{-3}	0.1	1	16.7분
2^n	3×10^{-5}	10^{-3}	36년	4×10^{16}년	3×10^{287}년
$n!$	10^{-4}	3.6	9.7×10^{50}년	?	?

알고리즘을 분석에서 가장 중요한 것

⇒ 주어진 문제를 해결할 수 있는 알고리즘 중 각 알고리즘의 수행 시간을 계산하여 가장 적은 수행 시간이 걸리는 알고리즘을 찾는 것

예제

4. 다음과 같이 n개의 양의 정수 중에서 가장 작은 수를 찾는 알고리즘에서 기억 장소의 크기와 수행 횟수를 구해보자.

```
Find_MIN (int array[ ], int MIN)
{
   int i ;
   MIN = array[0];
   for (i = 1; i < n; i++)
      if (MIN > array[i]) MIN = array[i];
   return (MIN);
}
```

5. 다음의 각각의 프로그램들에서 sum += i; 명령문에 대한 수행 횟수를 구해보자.

(1)
```
sum += i;
```

(2)
```
for (i = 0; i < n; i++)
   sum += i;
```

(3)
```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        sum += i;
```

12.4 알고리즘의 복잡성

교과목명	이산수학	분반		담당교수	김 외 현
학부(과)		학번		성명	

정의 **빅오(Big-Oh)**

알고리즘의 복잡성을 측정하는데 사용

- $f(n)= O(n)$
  : $f(n)$은 차수가 n 또는 $f(n)$은 n의 빅오

정의 **시간 복잡성 표기법**

$\forall n$에 대하여 $n \geq n_0$,
$f(n) \leq c \cdot g(n)$이 되는 양의 정수 c와 n_0가 존재

$\Leftrightarrow$ $f(n)= O(g(n))$

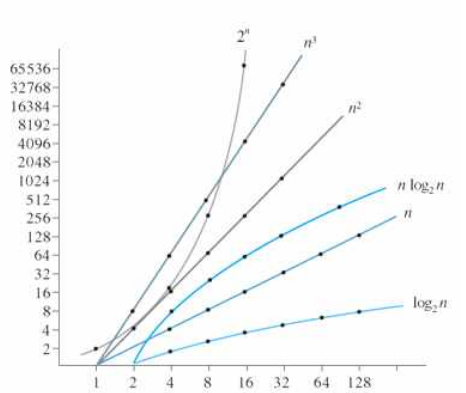
참고 **알고리즘 복잡성의 순서**

$O(1) < O(\log_2 n) < O(n) < O(n \log_2 n) < O(n^2) < O(n^3) < \cdots < O(2^n) < O(n!)$

- $O(1)$: 상수 복잡도
  알고리즘의 수행 시간이 입력 자료의 크기에 관계없이 항상 일정함을 의미
- 오른쪽으로 갈수록 n이 커짐에 따라 수행 시간이 급격히 증가함
- $O(\log n)$ 알고리즘의 수행 시간이 더 적게 필요하므로 보다 효율적인 알고리즘

참고 **입력 크기에 따른 각 함수의 증가 비율**

$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	4294967296



예제

6. 예제4의 알고리즘 수행 시간을 O를 사용하여 나타내어보자.

7. 다음 함수들을 O의 개념으로 나타내어보자.

(1) $f(n)= 3n+2$

(2) $f(n)= n^2+16n+1$

(3) $f(n)= 16n+5n\log_2 n$

8. 다음 함수들의 Big-Oh를 구해보자.

(1) $f(n)= \frac{1}{n}+63$

(2) $f(n)= 3n+2n\log n$

(3) $f(n)= 0.6n^3+28n^2+31n+468$

12.5 재귀 함수의 복잡성

교과목명	이산수학	분반		담당교수	김 외 현
학부(과)		학번		성명	

정의 **재귀 함수**

$f(n)$: 주어진 문제를 해결하는 데 필요한 함수

$f(n) = (f(n-1), f(n-2), \cdots, f(1)$ 중 한 개 이상의 내용이 포함)

$\Rightarrow$ $f(n)$: 재귀 함수

참고 **재귀 함수와 관련된 일상생활의 예**

(1) 올해의 물가는 작년의 물가에 비하여 몇 % 올랐다.

(2) 오늘까지 수행된 결과는 어제까지 수행된 결과와 오늘 할 일을 합한 것이다.

(3) 올해 아동의 키는 작년의 키에다 여러 요소들을 합한 것이다.

참고 재귀 함수를 이용하여 주어진 문제를 해결할 때는 현재의 함수와 그 전의 함수의 연관 관계를 파악하는 것이 중요함

예제

9. 주어진 재귀 함수가 다음과 같을 때 O를 구해보자.

$f(2) = 1$

$f(n) = (n-1) + f(n-1),\ n \geq 3$

10. 다음 알고리즘에서 재귀 함수식과 O를 구해보자.

```
int sum (int n)
{
   if (n != 0) return (1 + sum(n-1));
   else return 0;
}
```

12.6 탐색 알고리즘

교과목명	이산수학	분반		담당교수	김 외 현
학부(과)		학번		성명	

정의 **탐색**

주어진 파일 또는 원소들 중에서 어떤 특정한 원소를 찾는 것

참고 **탐색의 방법**

(1) 순차 탐색
원소들이 정렬되어 있지 않을 경우에 원소들을 처음부터 비교하여 찾는 방법

(2) 이진 탐색
원소들이 정렬되어 있을 경우에 찾는 방법, 순차 탐색보다 빠름

(1) $O(n)$ 알고리즘

- 순차 탐색은 배열에 있는 특정한 원소를 찾기 위하여 배열의 처음 원소부터 차례로 모든 원소들을 비교하여 탐색함
- 이 방법은 효율적이지는 않지만 효과적임
- 모든 원소들을 조사하는 탐색을 선형 탐색이라고 함

참고 **순차 탐색 알고리즘**

```
int seq_search (int array[ ], int search_num, int n)
/* n개의 원소를 가진 배열 array에서 search_num을 순서대로 찾아서
   있으면 그의 지수를 반환하고 없으면 -1을 반환한다 */
{
   int i;
   array[n] = search_num;
   for(i = 0; array[i] != search_num; i++);
   return ((i < n)?i : -1);
}
```

- 프로그램에서 for 문을 제외하면 다른 문장은 한 번만 수행함
- for 문에서의 비교 횟수를 조사함으로써 전체 수행 횟수를 알 수 있음
- 먼저 찾으려는 원소가 배열에 없다고 생각하면 i는 0부터 n까지 증가하며 비교하므로 $n+1$번의 비교함
- 만약 찾으려는 원소 search_num이 배열에 존재한다면, array[0]=search_num일 때는 한 번의 비교가 필요함
- array[n-1]=search_num일 때는 n번의 비교가 필요함
- 일반적으로 array[i]=search_num일 때 필요한 비교 횟수는 $i+1$임

평균 비교 횟수

$$\frac{1}{n}\sum_{i=0}^{n-1}(i+1) = \frac{n+1}{2} = O(n)$$

(2) $O(\log_2 n)$ 알고리즘

원소들이 순서대로 정렬되어 있을 때는 처음부터 찾을 필요가 없이 중간의 원소와 비교하여 그보다 작을 때에는 그 원소의 왼쪽 원소들 중에서, 클 때는 오른쪽 원소 중에서 다시 같은 형식으로 찾으면 훨씬 시간을 절약하여 찾을 수 있음

정의 **이진 탐색**

배열 가운데의 원소값과 찾으려는 값을 비교하여, 비교된 결과에 따라 왼쪽 원소의 배열 또는 오른쪽 원소의 배열 중에서 다시 찾기를 계속함

참고 **이진 탐색 알고리즘**

```
int binary_search (int array[ ], int search_num, int left, int right)
/* 배열 array에서 search_num이 있는가를 탐색한다. 있을 시에는 그 위치의
   지수(index)를 반환하고, 없을 시에는 -1을 반환한다. */
{
   int middle;
   while (left <= right) {
      middle = (left+right)/2;
      if (array[middle] < search_num)
         left = middle + 1;
      elseif (array[middle] > search_num)
         right = middle - 1;
      else
         return middle;
      }
   return -1;
}
```

정의 **분할 정복 알고리즘**

전체 집합을 찾으려는 원소와 비교하여 부분 집합으로 나누어 찾는 알고리즘

- 다시 나누어진 부분집합에서 같은 방법으로 적용됨
- 큰 문제를 작은 문제로 나누어 해결할 수 있으므로 효율적임

참고 **이진 탐색 알고리즘의 수행 횟수와 검색할 배열의 크기**

수행 횟수	검색해야 할 배열의 크기
1	n
2	$\frac{n}{2}$
3	$\frac{n}{4}$
4	$\frac{n}{8}$
⋮	⋮
k	$\frac{n}{2^{k-1}}$

정리 **이진 탐색 알고리즘의 수행 시간**

$= O(\log_2 n)$

예제

11. 배열의 원소들이 다음과 같은 순서를 가진다고 한다$(5, 8, 11, 20, 25, 33, 48, 50, 57)$. 이 경우 찾고자 하는 원소가 11일 때 이진 탐색 알고리즘을 이용하여 수행 횟수에 따른 left와 right 값이 변하는 과정을 살펴보자.

참고 **이진 탐색 트리**

- 이진 탐색 알고리즘을 이용하여 원소를 찾는 과정은 이진 탐색 트리를 이용하면 간단히 표현함
- 이진 탐색 트리는 처음 배열의 중간 원소를 루트(root)로 놓음
- 각 서브 트리의 루트는 둘로 나누어진 부분 집합들의 중간 원소들로 이루어짐
- 루트로부터 트리의 노드로 가는 경로는 이진 탐색 알고리즘의 비교 횟수와 동일하므로 최악의 경우 트리의 깊이에 해당되는 비교 횟수인 $O(\log_2 n)$이 됨
- $O(\log_2 n)$인 이진 탐색은 $O(n)$인 순차 탐색보다 빠르고 효율적임

12.7 정렬 알고리즘

교과목명	이산수학	분반		담당교수	김 외 현
학부(과)		학번		성명	

정의 **정렬**

임의로 나열되어 있는 데이터들을 주어진 항목에 따라 크기 순서대로 작은 순서부터(오름차순) 또는 큰 순서부터(내림차순) 늘어놓는 것

참고 **정렬된 데이터의 응용**

(1) 데이터를 탐색할 때

(2) 리스트에 있는 다른 항목들을 비교할 때

참고 **정렬의 분류**

(1) 내부 정렬

정렬하려는 데이터들이 모두 주기억장치에 저장되어 있는 경우 사용

$O(n^2)$, $O(n\log_2 n)$

(2) 외부 정렬

정렬하려는 데이터의 크기가 커서 주기억장치에 다 저장할 수 없을 경우에 사용

(1) $O(n^2)$ 알고리즘

버블 정렬, 삽입 정렬

정의 **버블 정렬 알고리즘**

인접하는 두 개의 원소를 비교해 기준에 따라 순서를 바꾸는 방법

```
void bubble_sort (int array[ ], int n)
/* 데이터들을 버블 정렬을 이용하여 오름차순으로 정렬한다 */
{
   int i, j, temp;
   for (i = 0; i < n-1; i++)
      for (j = 0; j < n-i-1; j++)
         if (array[j] > array[j+1]) {
            temp = array[j];
            array[j] = array[j+1];
            array[j+1] = temp;
         }
}
```

예제

12. $n=5$이고 원소가 $10, 7, 19, 5, 16$일 때, 버블 정렬 알고리즘을 이용하여 수행하는 단계를 나타내어보자.

⑵ $O(n\log_2 n)$ 알고리즘

퀵 정렬, 병합 정렬, 힙 정렬

정의 퀵 정렬

- 모든 정렬 방법 중에서 평균 수행 시간이 가장 빠른 방법임
- 현재 정렬 과정에서 처리되고 있는 피벗 키 p를 기준으로 원소를 두 부분으로 나눔
- 나누어진 두 부분을 따로 다시 퀵 정렬로 재귀함으로써 모든 원소가 순서대로 정렬될 때까지 실행함
- 퀵 정렬의 평균 수행 시간 $= O(n\log_2 n)$
- 정렬해야 할 원소들의 값에 따라 최악의 경우에는 $O(n^2)$이 될 수 있음

정의 병합 정렬

- 크기가 1인 정렬된 두 파일을 병합하여 크기가 2인 파일들을 생성함
- 이 파일들에 대하여 병합 과정을 반복 시행하여 한 개의 파일을 만들어내는 방법임
- 병합이란 여러 개의 정렬되어 있는 배열 자료들을 혼합하여 하나의 정렬된 배열로 합하는 작업을 의미함
- 두 개의 서로 다른 정렬된 배열을 합하여 하나의 정렬된 배열로 만드는 병합 방식을 2-way 병합 과정임
- n-way 병합은 n개의 서로 다른 정렬된 배열을 하나의 정렬된 배열로 합치는 병합 방식을 의미함

정의 힙 정렬

- 자료를 정리할 때 모든 자료들을 동시에 처리하지 않고 모든 자료 중에서 가장 큰 자료를 찾아 출력시킴
- 나머지 자료 중에서 앞의 과정을 반복하여 가장 큰 자료(실제로는 두 번째 큰 자료)를 찾아 출력시키면서 정렬하는 방법임

- 병합 정렬과 힙 정렬의 평균 수행 시간 $= O(n\log_2 n)$