12 CHAPTER

알고리즘을 통한 문제 해결

Problem Solving Utilizing Algorithm

단원의 주요 목표

알고리즘의 개념을 이해하고 문제 해결과 관련된 알고리즘 논제들을 고찰한다.

- 알고리즘의 정의와 특성, 표현 방법 등에 관해 알아본다.
- 시간 복잡성과 공간 복잡성으로 알고리즘의 효율성을 판단한다.
- 빅오 개념으로 알고리즘의 복잡성을 측정한다.
- 재귀 함수의 식을 전개하여 복잡성을 계산한다.
- 탐색 알고리즘과 정렬 알고리즘을 학습한다.
- 알고리즘의 응용 분야와 4차 산업혁명과의 관계를 살펴본다.

12 CHAPTER

알고리즘을 통한 문제 해결

Problem Solving Utilizing Algorithm

CONTENTS

12.1 알고리즘이란 무엇인가?

정의 12-1 알고리즘(algorithm)이란 주어진 문제를 해결하기 위해 필요한 여러 가지 단계들을 체계적으로 명시해 놓은 것을 말한다. 알고리즘의 사전적 의미는 '어떤 문제를 해결하는 한 방법의 상세한 특징을 기술하는 것'이다.

알고리즘이 가져야 할 7가지 주요 특성

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

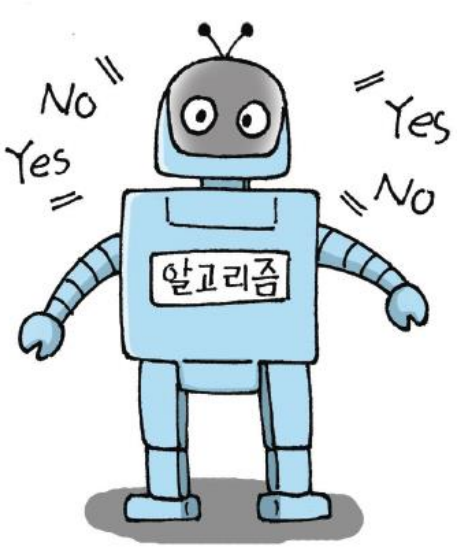
12.1 알고리즘이란 무엇인가?

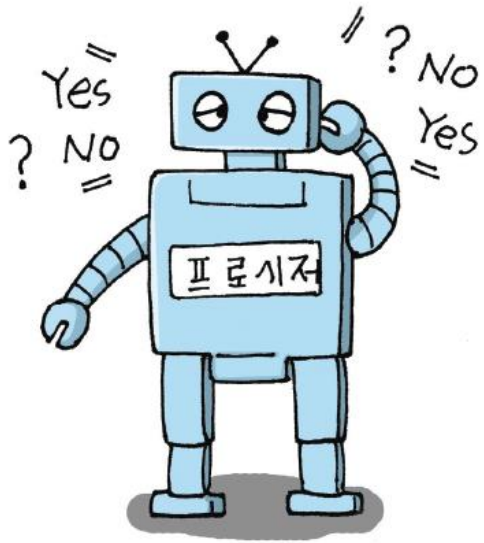
알고리즘과 프로시저의 차이

여기서 잠깐!!

알고리즘과 프로시저(procedure)는 비슷하게 쓰이지만 엄밀하게는 그 의미에 분명한 차이가 있다.

- 알고리즘 : 어떤 문제 해결에 대한 Yes 또는 No가 분명한 문제 해결 방법론이다.
- 프로시저 : 어떤 문제 해결에 대한 Yes나 No, 또는 프로그램이 종료될지 아닐지도 모르는 문제 해결 방법론이다.





12.1 알고리즘이란 무엇인가?

알고리즘(Algorithm)

- 순서도(flow chart), 유사 코드(pseudo code), 언어(language) 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

12.2 알고리즘의 표현과 효율성

일상생활에서 흔히 만나는 알고리즘들

여기서 잠깐!!

우리는 일상생활에서 의식적이든 무의식적이든 다양한 분야에서 알고리즘을 이용하여 살아가고 있다. 생활 속의 몇 가지 알고리즘의 예를 살펴보면 다음과 같다.

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 잘 할 수 있는 알파고와 같이 생각하는 방법론

⑥ 어떤 목적지로 이동하려고 할 때 어느 지하철역에서 환승하는 것이 더 효율적인지 등

그 외에도 우리가 생활 속에서 알고리즘을 이용하는 경우는 우리 생활 주변에 너무나 많다.

12.2 알고리즘의 표현과 효율성

유사 코드

유사 코드(pseudo code)는 알고리즘을 프로그래밍 언어와 유사한 형태로 풀어 써 놓은 것으로서, 정형화된 문법적 측면을 배제하고 사고의 흐름을 간결하고 효과적으로 전달하는 표현이다.

유사 코드를 나타내는 추가적인 표현은 다음과 같다.

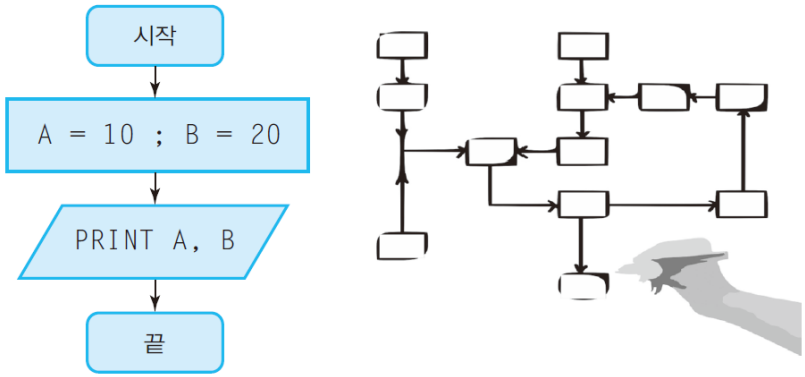
- 유사 코드는 알고리즘을 개략적으로 표현하는 데에 쓰인다.
- 유사 코드로 적는 것은 알고리즘의 각 단계를 차례로 적는 것이다.
- 유사 코드는 일반적으로 C언어나 자연어와 유사하게 기술할 수 있다.

12.2 알고리즘의 표현과 효율성

순서도

순서도(flow chart)란 처리하고자 하는 문제를 분석한 후 처리 순서를 단계화하고, 상호간의 관계를 표준 기호를 사용하여 입력, 처리, 결정, 출력 등의 박스와 연결선으로 일목요연하게 나타낸 도표(diagram)를 말한다.

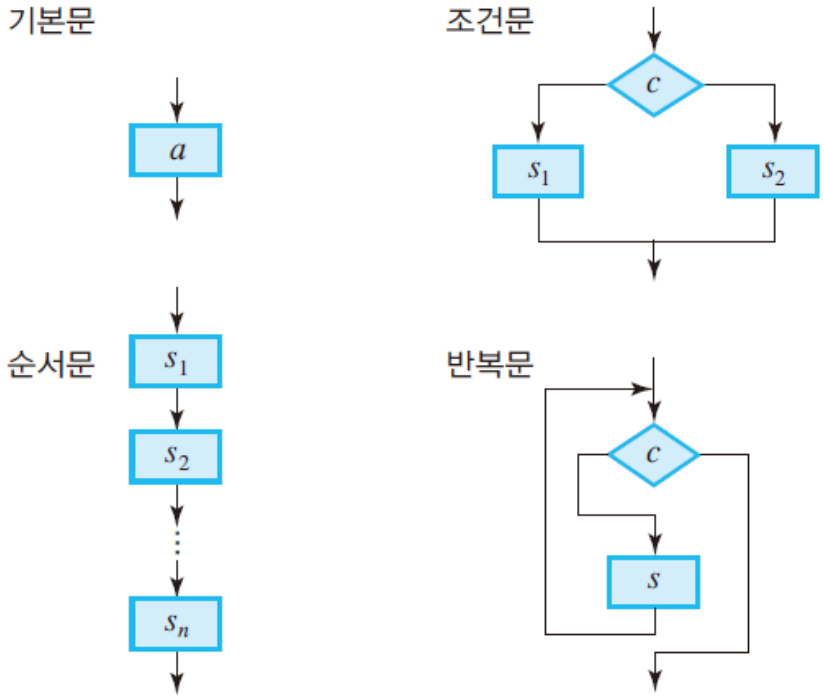
순서도는 처리 순서를 논리적으로 표현하는 도표로서 작업의 흐름을 나타내기 때문에 '흐름도'라고도 불리고 있다. 〈그림 12.1〉과 같은 형태의 순서도는 주로 컴퓨터 프로그램을 위해 많이 쓰이지만, 반드시 컴퓨터의 이용을 전제로 하는 것은 아니다.



〈그림 12.1〉 순서도의 예

12.2 알고리즘의 표현과 효율성

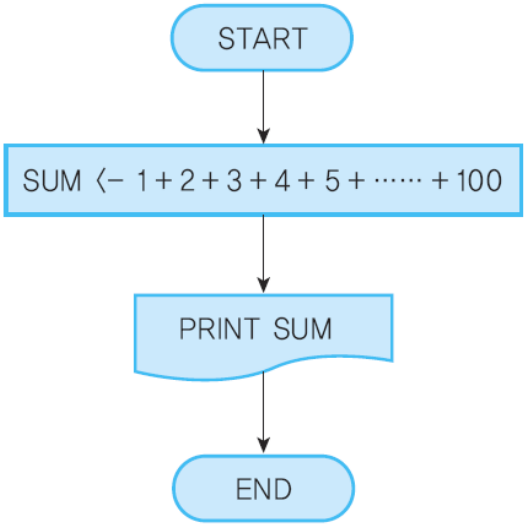
순서도의 유형은 〈그림 12.2〉와 같다.



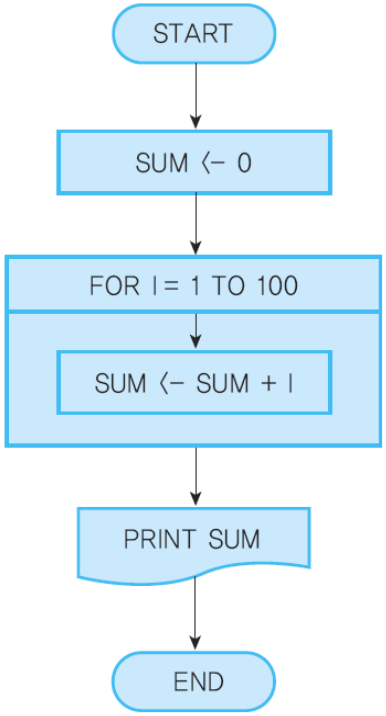
〈그림 12.2〉 순서도의 유형

12.2 알고리즘의 표현과 효율성

- 알고리즘의 효율성을 비교하기 위함
- 왼쪽은 1부터 100까지 정수를 그냥 더하는 방법임
- 오른쪽은 for 문을 사용하여 계산하는 방법이 훨씬 효율적인 알고리즘이라고 말할 수 있음



〈그림 12.1〉 일반적인 계산 방법



〈그림 12.2〉 for 문을 사용한 계산 방법

12.3 알고리즘 분석

- 알고리즘에 대한 평가와 비교는 컴퓨터 프로그램을 통한 문제 해결에 있어서 매우 중요한 관심 분야임
- 어떤 문제를 해결하는 데 있어서 두 가지 질문을 해봄으로 알고리즘을 분석할 수 있음
  - ① 어떤 문제의 해결에 있어서 주어진 알고리즘을 사용하는 데 드는 비용이 얼마인가?
  - ② 그 문제를 해결하는 데 비용이 가장 적게 드는 알고리즘은 무엇인가?
- 비용이란 연산하는 데 필요한 **시간**과 **기억 장소의 크기**를 말함
- 주어진 문제를 해결하는 방법에는 여러 가지 알고리즘이 있으나 그 중에서 비용이 적게 드는 알고리즘을 찾기 위해서는 **효율성 분석 (performance analysis)**이 필요함

12.3 알고리즘 분석

시간 복잡성과 공간 복잡성

- 효율성을 분석하기 위해서는 알고리즘 수행 시 필요한 **시간 복잡성(time complexity)과 공간 복잡성(space complexity)**의 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

여기서 잠깐!!

최근에는 컴퓨터 하드웨어의 발달로 알고리즘 분석에서 어떤 문제를 해결하는 데 필요한 기억 장소의 크기에는 그리 큰 비중을 두고 있지 않다. 따라서 여기서는 주로 알고리즘의 수행 시간에 대하여 분석하고 살펴보기로 한다.

12.3 알고리즘 분석

예제 12-1

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);
}
```

12.3 알고리즘 분석

풀이 위의 프로그램에서 array는 n개의 정수를 저장하는 배열로 선언되었다고 가정한다.

(1) 기억 장소의 크기 : 정수 한 개를 저장하는 데 2바이트(byte)가 필요하다고 가정하면, Find_MIN에서는 배열 array의 n개의 원소와 단수 변수 MIN, i가 필요하므로 2(n+2) 바이트가 필요하다.

(2) 수행 시간 : `MIN = array[0];`와 `return (MIN);`은 반복문에 속해 있지 않으므로 각각 1회씩 수행된다. `if (MIN > array[i]) MIN = array[i];`는 반복문에 속해 있으므로 n-1번 수행된다. 그러므로 전체 수행 횟수는 n+1이다.

12.3 알고리즘 분석

예제 12-2 다음의 각각의 프로그램들에서 `sum += i;` 명령문에 대한 수행 횟수를 구해 보자.

(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;
```

풀이 (1)에서의 `sum += i;` 명령문의 수행 횟수는 입력 개수 n과는 상관없이 1회 수행된다. (2)에서는 n번 수행되며, (3)에서는 n^2 횟수만큼 수행된다.

12.3 알고리즘 분석

〈표 12.1〉 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 분석하는 데 있어서 가장 중요한 것은 주어진 문제를 해결할 수 있는 알고리즘 중 각 알고리즘의 수행 시간을 계산하여 가장 적은 수행 시간이 걸리는 알고리즘을 찾는 것임

12.4 알고리즘의 복잡성

정의 12-2 알고리즘의 복잡성을 측정하는 데 있어서 일반적으로 빅오(Big-Oh) 표현을 사용하며, 대문자 O를 써서 표시한다.

예를 들어, $f(n)=O(n)$로 표시가 되었다면, 우리는 이것을 $f(n)$의 차수(order)가 n이라 하며, 'Big-Oh of n' 이라고 읽는다.

정의 12-3 음수값을 가지지 않는 함수 f와 g에서 모든 n에 대하여 $n \geq n_0$이고, $f(n) \leq c \cdot g(n)$이 되는 양의 상수 c와 n_0가 존재한다면, $f(n)=O(g(n))$이다.

즉, $O(g(n))$은 충분히 큰 수 n이 주어질 때 $g(n)$에 양의 상수 배를 한 함수들 중에서 가장 작은 함수를 의미한다.

12.4 알고리즘의 복잡성

알고리즘 복잡성의 순서

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

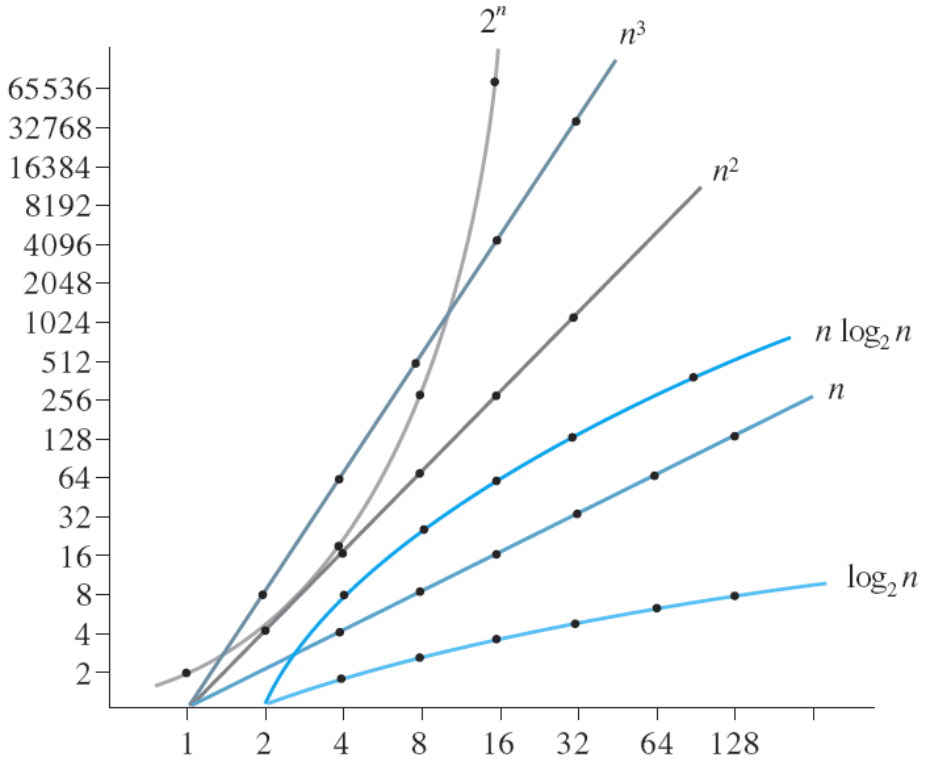
- $O(1)$이라 함은 문제를 해결하는 데 걸리는 수행 시간이 입력 자료의 수 n의 크기에 관계없이 상수임을 나타냄
- 오른쪽으로 갈수록 n이 커짐에 따라 수행 시간이 급격히 증가함
- 어떤 문제를 해결할 수 있는 알고리즘 중에서 하나의 알고리즘은 $O(\log_2 n)$이고 다른 알고리즘은 $O(n)$의 복잡성을 가질 수 있음
- $O(\log_2 n)$ 알고리즘의 수행 시간이 더 적게 필요하므로 보다 효율적인 알고리즘이라고 함

12.4 알고리즘의 복잡성

〈표 12.2〉 입력 크기에 따른 각 함수의 증가값

$\log_2 n$	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

12.4 알고리즘의 복잡성



〈그림 12.3〉 입력 크기에 따른 각 함수의 증가 비율

12.4 알고리즘의 복잡성

예제 ⑫-3 예제 ⑫-1의 알고리즘 수행 시간을 O를 사용하여 나타내어보자.

풀이 예제 ⑫-1의 알고리즘의 수행 시간은 $n+1$이므로, $n \geq 1$일 때 $n+1 \leq 3n$이 성립된다. 그러므로 정의 ⑫-3에 의하여 $n+2=O(n)$이다.

예제 ⑫-4 다음 함수들을 O의 개념으로 나타내어보자.

(1) $3n+2$

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

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

풀이 (1) $n \geq 2$일 때 $3n+2 \leq 4n$이 성립하므로, $3n+2=O(n)$이다.

(2) $n^2+16n+1 \leq 2n^2$이 $n \geq 17$일 때 성립하므로, $n^2+16n+1=O(n^2)$이다.

(3) $16n \in O(n)$이고 $5n\log_2 n \in O(n\log_2 n)$이므로, $16n+5n\log_2 n=O(n\log_2 n)$이다.

12.4 알고리즘의 복잡성

예제 12-5 다음 함수들의 Big-Oh를 구해보자.

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

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

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

풀이 (1) $O(1)$ (2) $O(n\log n)$ (3) $O(n^3)$

12.5 재귀 함수의 복잡성

재귀 함수(Recursive function)

- 주어진 문제를 해결하는 데 있어서 f(n)이 필요한 함수라고 했을 때, f(n)의 식에 f(n-1), f(n-2), ..., f(1) 함수 중 한 개 이상의 내용이 포함되는 함수임
- 재귀 함수에서 현재의 함수 값은 항상 그 전에 있던 함수에 의해 영향을 받음

여기서 잠깐!!

우리가 재귀 함수를 이용하여 주어진 문제를 해결할 때는 현재의 함수와 그 전의 함수의 연관 관계를 파악하는 것이 중요하다. 재귀 함수와 같은 의미로 되부름 함수나 순환 함수 등이 쓰이고 있다.

12.5 재귀 함수의 복잡성

예제 12-6 주어진 재귀 함수가 다음과 같을 때 O를 구해보자.

$$f(2) = 1$$

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

풀이 n에 3, 4, 5를 대입하여 식을 풀어보면 다음과 같다.

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

$$f(4) = 3 + 2 + 1$$

$$f(5) = 4 + 3 + 2 + 1$$

이런 방법으로 식을 전개해 나가면 다음과 같은 식이 된다.

$$\begin{aligned} f(n) &= (n-1) + (n-2) + \cdots + 2 + 1 \\ &= \frac{n(n-1)}{2} \\ &= \frac{n^2}{2} - \frac{n}{2} \\ &= O(n^2) \end{aligned}$$

12.5 재귀 함수의 복잡성

예제 12-7 다음 알고리즘에서 재귀 함수식과 O를 구해보자.

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

풀이 재귀 함수 sum은 if 문을 체크하여 n이 0이 아니면 매개 변수의 크기를 하나 줄여서 다시 자기 자신을 부른다. 그리고 n이 0일 때는 0의 값을 반환(return)한다. 그러므로 $f(0)$의 값은 단 1문장을 수행하므로 $f(0)=1$이 된다. 전체 함수의 식을 $f(n)$이라고 하면 다음과 같이 나타낼 수 있다.

12.5 재귀 함수의 복잡성

$$f(n) = 1 + f(n-1),\ n \geq 1, f(0) = 1$$

이것을 전개하면,

$$\begin{aligned} f(0) &= 1 \\ f(1) &= 1 + f(0) = 1 + 1 \\ f(2) &= 1 + f(1) = 1 + 1 + 1 \\ &\vdots \\ f(n) &= n + 1 \\ &= O(n) \end{aligned}$$

12.6 탐색 알고리즘

탐색(search)

- 주어진 파일 또는 원소들 중에서 어떤 특정한 원소를 찾는 것임
- 탐색의 2가지 방법
  - ✓ **순차 탐색(sequential search)** : 원소들이 정렬되어 있지 않을 경우에 원소들을 처음부터 비교하여 찾는 것임
  - ✓ **이진 탐색(binary search)** : 원소들이 정렬되어 있을 경우에 찾는 방법으로서 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 탐색(linear search)**이라고 함

12.6 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);
}
```

[프로그램 12.1] 순차 탐색 알고리즘

- 프로그램에서 for 문을 제외하면 다른 문장은 한 번만 수행함
- for 문에서의 비교 횟수를 조사함으로써 전체 수행 횟수를 알 수 있음

12.6 탐색 알고리즘

- 먼저 찾으려는 원소가 배열에 없다고 생각하면, i는 0부터 n까지 증가하며 비교하므로 n+1번의 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+1임

평균 비교 횟수

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

$$=\frac{1}{n}\cdot\frac{n(n+1)}{2}$$

$$=\frac{n+1}{2}$$

$$=O(n)$$

12.6 탐색 알고리즘

(2) $O(\log_2 n)$ 알고리즘

- 전화번호부에서 사람의 이름을 찾을 때, 먼저 이름이 있을 곳을 추측하여 찾는데, 그곳이 찾는 사람의 이름보다 앞쪽의 이름이면 뒤쪽을 다시 찾고, 뒤쪽의 이름이면 앞쪽을 다시 찾음
- 이런 식으로 이름을 찾을 때까지 반복함
- 이와 같이 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

이진 탐색(binary search)

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

12.6 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;
}
```

[프로그램 12.2] 이진 탐색 알고리즘

12.6 탐색 알고리즘

분할 정복 알고리즘(Divide and conquer algorithm)

- 전체 집합을 찾으려는 원소와 비교하여 부분 집합으로 나누어 찾는 알고리즘임
- 이 알고리즘은 다시 나누어진 부분 집합에서 같은 방법으로 적용함
- 알고리즘이 효율적인 이유는 큰 문제를 작은 문제로 나누어서 해결할 수 있음

12.6 탐색 알고리즘

이진 탐색 알고리즘의 수행 시간을 살펴보면 while 반복문이 1회 수행될 때마다 탐색해야 될 배열의 크기가 계속 반으로 줄어듬

〈표 12.3〉 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	$\frac{n}{2^{k-1}}$

12.6 탐색 알고리즘

탐색해야 할 배열의 크기가 1일 때 알고리즘이 수행을 끝내므로 알고리즘의 수행 시간임

$$\frac{n}{2^{k-1}} = 1$$

$$n = 2^{k-1}$$

$$\log_2 n = k - 1$$

$$k = 1 + \log_2 n = O(\log_2 n)$$

따라서 이진 탐색 알고리즘의 수행 시간은 $O(\log_2 n)$

12.6 탐색 알고리즘

예제 12-8 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.

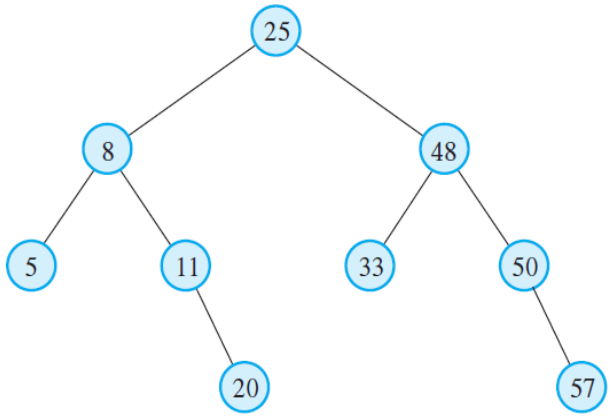
풀이

수행 횟수	left	right
초기값	0	8
1	0	3
2	2	3

- 이진 탐색 알고리즘을 이용하여 원소를 찾는 과정은 **이진 탐색 트리 (binary search tree)**를 이용하면 간단히 표현함
- 이진 탐색 트리는 처음 배열의 중간 원소를 루트(root)로 놓임
- 각 서브 트리의 루트는 둘로 나누어진 부분 집합들의 중간 원소들로 이루어짐

12.6 탐색 알고리즘

이진 탐색 트리



이진 탐색 트리의 루트로부터 트리의 노드로 가는 경로는 이진 탐색 알고리즘의 비교 횟수와 동일하므로 최악의 경우 트리의 깊이(depth)에 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

정렬(sort)

- 임의로 나열되어 있는 데이터들을 주어진 항목에 따라 크기 순서대로 작은 순서부터**(오름차순, ascending order)** 또는 큰 순서부터**(내림차순, descending order)** 늘어놓는 것임
- 정렬되어 있는 데이터들은 다음과 같은 작업을 수행할 때 응용

(1) 데이터를 탐색할 때

(2) 리스트(list)에 있는 다른 항목들을 비교할 때

(1) $O(n^2)$ 알고리즘

$O(n^2)$ 알고리즘으로는 널리 알려진 **버블 정렬(bubble sort)과 삽입 정렬(insertion sort)** 두 가지가 있음

12.6 정렬 알고리즘

```
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;
            }
}
```

[프로그램 12.3] 버블 정렬 알고리즘

12.7 정렬 알고리즘

버블 정렬(Bubble sort)

- **버블 정렬**은 이웃한 두 개의 원소를 비교하여 순서가 서로 다르면 원소의 자리를 서로 바꾸고, 그렇지 않으면 그 위치에 그대로 놓음
- 위의 알고리즘에서 i가 한 번 수행될 때마다 제일 오른쪽 끝에서부터 원소 중 큰 순서대로 정렬이 됨
- 앞의 알고리즘에 대한 수행 시간을 알아보면, i = 0일 때 j에 대한 반복문은 n-1번 수행되며, i = 1일 때 n-2번 수행됨

알고리즘의 전체 수행시간

$$(n-1)+(n-2)+\cdots+2+1=\frac{n(n-1)}{2}=\frac{1}{2}n^2-\frac{1}{2}n=O(n^2)$$

12.7 정렬 알고리즘

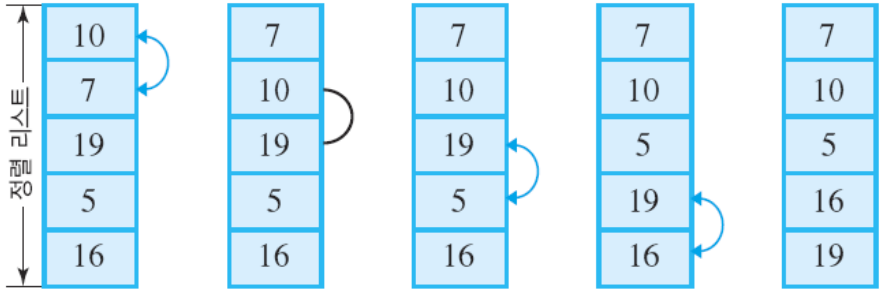
버블 정렬의 실제 예

12.7 정렬 알고리즘

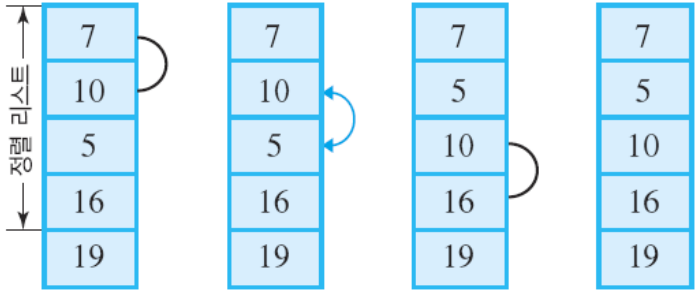
예제 12-9 $n=5$이고 원소가 10, 7, 19, 5, 16일 때, 버블 정렬 알고리즘을 이용하여 수행하는 단계를 나타내어보자.

풀이 버블 정렬은 다음과 같은 네 단계를 거쳐 정렬될 수 있다. 그림에서 화살표가 있는 경우에는 교환을 하고, 화살표가 없는 경우에는 교환할 필요가 없으므로 비교만 나타낸다.

[1단계]

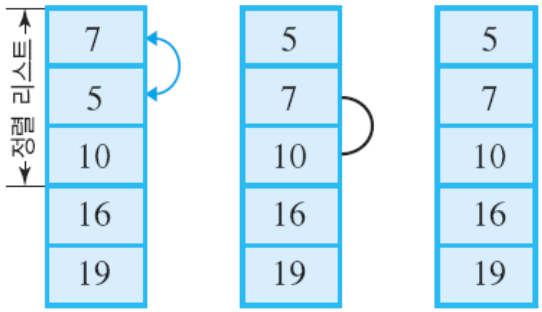


[2단계]

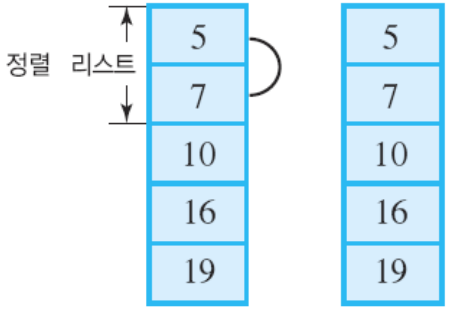


12.7 정렬 알고리즘

[3단계]



[4단계]



그 결과 크기가 큰 숫자가 가장 밑에서부터 차례대로 정렬되는 것을 알 수 있다.

12.7 정렬 알고리즘

(2) $O(n\log_2 n)$ 알고리즘

$O(n\log_2 n)$ 알고리즘에는 퀵 정렬, 병합 정렬, 힙 정렬 등 여러 가지 알고리즘이 있음

퀵 정렬(quick sort)

- 모든 정렬 방법 중에서 평균 수행 시간이 가장 빠른 방법임
- 현재 정렬 과정에서 처리되고 있는 피벗 키(pivot key) p를 기준으로 원소를 두 부분으로 나눔
- 나누어진 두 부분(subfile)을 따로 다시 퀵 정렬로 재귀함으로써 모든 원소가 순서대로 정렬될 때까지 실행함
- 퀵 정렬의 **평균 수행 시간은 $O(n\log_2 n)$**임
- 정렬해야 할 원소들의 값에 따라 **최악의 경우에는 $O(n^2)$**됨

12.7 정렬 알고리즘

병합 정렬(merge sort)

- 크기가 1인 정렬된 두 파일을 병합하여 크기가 2인 파일들을 생성함
- 이 파일들에 대하여 병합 과정을 반복 시행하여 한 개의 파일을 만들어내는 방법임
- **병합(merging)**이란 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 병합**은 *n*개의 서로 다른 정렬된 배열을 하나의 정렬된 배열로 합치는 병합 방식을 의미함

12.7 정렬 알고리즘

힙 정렬(heap sort)

- 자료를 정리할 때 모든 자료들을 동시에 처리하지 않고 모든 자료 중에서 가장 큰 자료를 찾아 출력임
- 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 시간은 $O(n\log_2 n)$임

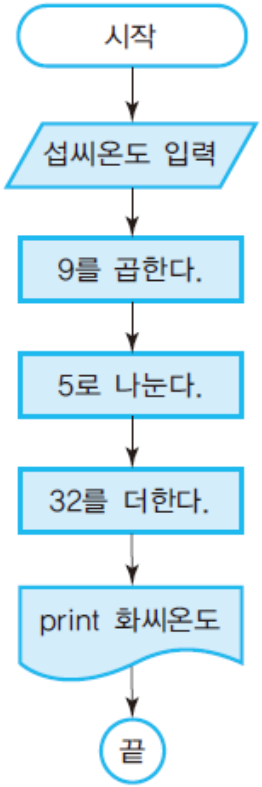
12.8 알고리즘의 응용과 4차 산업혁명과의 관계

(1) 알고리즘의 응용 분야

① 문제 해결의 효율적인 방법론

알고리즘이란 주어진 문제를 해결하기 위해 필요한 여러 가지 단계들을 체계적으로 명시해놓은 것을 말하는데, 일상생활에서 만나는 수많은 문제들의 효과적인 해결 방안들을 포함한다. 예를 들어 덧셈이나 곱셈 등의 수학적 계산 방법, 전자레인지 등 전자제품의 사용 설명서, 단계적 음식 조리법, 지하철 노선의 최단 경로를 찾는 법 등에 응용될 수 있다. 〈그림 12.6〉은 섭씨를 화씨로 변환하는 알고리즘을 나타내는 순서도이다.

12.8 알고리즘의 응용과 4차 산업혁명과의 관계



〈그림 12.6〉 섭씨를 화씨로 변환하는 알고리즘을 나타내는 순서도

12.8 알고리즘의 응용과 4차 산업혁명과의 관계

② 컴퓨터와 공학 분야에의 응용

알고리즘은 컴퓨터를 통한 프로그램의 핵심 단계에서 매우 중요한 역할을 담당한다. 수학에서는 문제를 풀기 위해서 정의나 정리들을 활용하지만 컴퓨터에서는 알고리즘을 사용한다. 가령 정렬이나 탐색을 할 때 알고리즘이 먼저 정해지지 않으면 프로그램을 코딩할 수 없다고 해도 과언이 아니다. 또한 알고리즘은 첨단 반도체 설계와 같은 공학 분야에도 널리 응용되고 있다.

12.8 알고리즘의 응용과 4차 산업혁명과의 관계

(2) 알고리즘과 4차 산업혁명과의 관계 : 로보틱스

로보틱스(Robotics) 기술은 인공지능적인 능력을 가진 로봇 기술과 관련된 4차 산업혁명의 한 분야이다. 특히 지능형 로봇(Intelligent robotics)은 스스로 판단하고 행동하며, 외부 환경에 적응할 수 있는 로봇으로서 다양한 환경에서 여러 가지 업무를 수행할 수 있다. 따라서 〈그림 12.7〉과 같은 로보틱스 분야에서는 알고리즘의 중요성이 매우 크며, 모든 판단과 행동에 있어 효율적인 알고리즘의 적용이 필수적이다.

〈그림 12.7〉 로보틱스

요약 및 생활 속의 응용

- 알고리즘이란 주어진 문제를 해결하기 위해 필요한 여러 가지 단계들을 체계적으로 명시해놓은 것을 말한다.
- 알고리즘이 가져야 할 7가지 주요 특성은 입력, 출력, 유한성, 정확성, 결정성, 일반성, 효율성이다.
- 알고리즘은 유사 코드, 순서도, 언어 등 여러 가지 방법으로 표현될 수 있는데, 누구나 이해할 수 있도록 명확하게 기술하는 것이 매우 중요하다.
- 알고리즘의 복잡성을 측정하는 데 있어서 일반적으로 대문자 O를 써서 빅오(Big-Oh) 표현 방법으로 나타낸다.
- 알고리즘의 복잡성 함수들은 다음과 같은 관계를 가진다.

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

- $O(2^n)$이나 $O(n!)$ 알고리즘은 n값이 커짐에 따라 계산 시간이 엄청나게 커지므로 컴퓨터 수행이 거의 불가능해진다.

요약 및 생활 속의 응용

- 탐색이란 주어진 파일 또는 원소들 중에서 어떤 특정한 원소를 찾는 것을 말하며, 탐색 방법에는 순차 탐색과 이진 탐색 등이 있다.
- 정렬이란 임의로 나열되어 있는 데이터들을 주어진 항목에 따라 크기 순서대로 작은 순서부터 또는 큰 순서부터 늘어놓는 것을 말한다.
- $O(n^2)$ 알고리즘으로 널리 알려진 것으로는 버블 정렬과 삽입 정렬 등이 있다.
- $O(n\log_2 n)$ 알고리즘에는 퀵 정렬, 병합 정렬, 힙 정렬 등 여러 가지 알고리즘이 있다.
- 알고리즘의 생활 속의 응용 예는 매우 많은데 그중 몇 가지는 다음과 같다. 지하철 노선의 최단 경로를 찾는 법, 전자레인지 등 전자 제품의 사용 설명서, 단계적 음식 조리법, 덧셈이나 곱셈 등의 수학적 계산, 수많은 일상생활에서 만나는 문제들의 효과적인 해결 방안 등이다.
- 알고리즘과 4차 산업혁명과의 관계로는 로보틱스를 들 수 있다.

CHAPTER 12
FINISH