개발 역량강화를 위한 알고리즘 정복 CAMP

CHAPTER01 – Basic Iterative Algorithm Design

Dongyi Kim
dykim@codingmonster.net

Problem 01A. 최대값 함수

아래 함수의 정의를 참고하여 완성해봅시다.

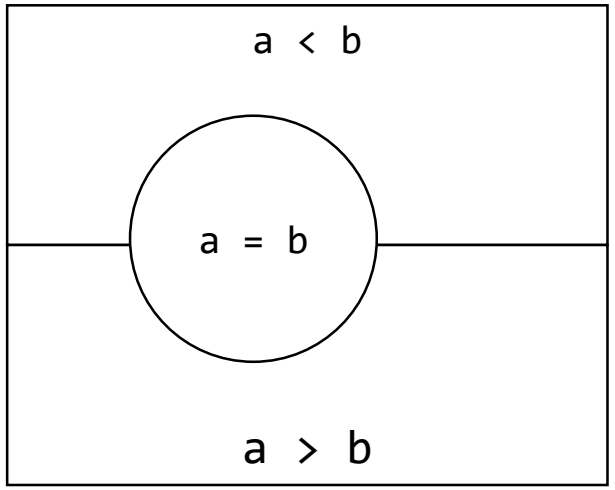
이 함수 내에서 발생할 수 있는 모든 경우의 수를 충분히 고려해보세요.

```java
/**
 * 두 정수 a와 b중 더 큰 값을 반환하는 함수
 * @param a
 * @param b
 * @return a와 b중 더 큰 값
 */
public static int getMax(int a, int b)
{




}
```

Problem 01A. 최대값 함수

파라미터 a와 b의 대소관계에 따라서 세 가지 경우의 수가 발생할 수 있다.

Q1. 각 경우의 수에 따라서 반환해야 할 값은 무엇일까?



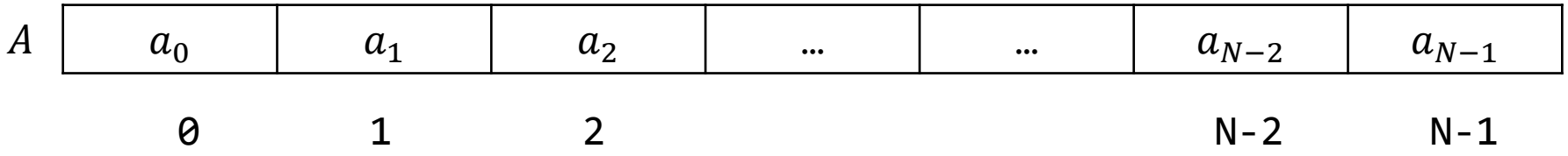
Q2. 이 함수는 문제에서 주어진 조건하에서 모든 입력을 올바르게 처리할 수 있는가?

배열 이해하기

배열은 가장 기본적인 자료구조 입니다.

- 원하는 데이터들을 번호(index)로 구분하여 일렬로 나열할 수 있으며,
  빠르게 각 원소를 읽거나 쓸 수 있습니다.
- 0부터 차례로 1씩 증가하며 원소의 인덱스를 가지기에, 반복문으로 차례로 데이터를 조회하기 용이합니다.

본 챕터에서는 배열에 저장된 다양한 데이터를 반복문을 활용해 처리하는 방법들에 대하여 알아봅니다.



반복문 이해하기

반복문은 항상 순차적으로 실행됩니다.

- 아래와 같은 가장 기본적인 구조의 반복문을 이용해 귀납적이고 순차적인 알고리즘을 설계할 수 있습니다.
- 항상 반복문이 우리가 정의한 범위와 순서에 대해 규칙적이고 모순없이 수행된다는 가정으로 알고리즘을 설계해야 합니다.

```
for(int i = 0 ; i < N ; i ++)
{
    /**
     *  이 반복문 안에서는 i가
     *   - 0 부터 (N-1)까지
     *   - (증가하는) 차례대로
     *   - 정확히 한 번씩
     *   나타난다
     */

    SomeCodes(i); //i에 대한 코드 뭉치
}
```

```
for(int i = 0 ; i < N ; i ++)
{
    /**
     * [0, N-1]의 모든 i가
     * 차례로
     * 한 번씩 나타난다.
     */



    // A[0] ~ A[N-1]이 차례로 한 번씩 출력된다.
    System.out.printf("%d", A[i]);
}
```

Problem 01B. 배열의 최대값

정수 배열 A의 원소 중 최대값 M은 아래를 만족한다.

$M \in A$ 이고 모든 i에 대하여 $M \geq A[i]$이다.

반복문으로 배열 전체의 최대값을 계산하려면 어떻게 해야 할까?

```
/**
 * 배열의 최대값을 계산하는 함수.
 *
 * @param data
 * @param n
 * @return data[0] ~ data[n-1]중 최대값.
 */
public static int getMax(int[] data, int n)
{




}
```

반복문과 조건문

반복문 내부의 조건문을 통해 모든 i중 특정한 조건을 만족하는 i만을 필터링할 수 있습니다.
즉, 조건문 내부의 실행문은 모든 i들 중 '조건을 만족한 i들'에 대해서만 수행됩니다.

```
for(int i = 0 ; i < n ; i++)
{
    // A  - 모든 i에 대해 수행

    if( Some Conditions … )
    {
        // B – 조건을 만족한 i에 대해 수행
    }

    //C – 모든 i에 대해 수행
}
```

Problem 01C. 카운팅 하기

조건문을 활용하여 배열 내부의 원소들 중 특정 조건을 만족하는 원소의 수를 계산할 수 있습니다. 앞서 언급한 반복문 내부 조건문의 특징을 이용해 답을 설계해봅시다.

```java
/**
 * @param data 각 사람의 키를 저장한 배열
 * @param n    사람들의 수
 * @param m    미주의 키
 * @param s    지수의 키
 * @return     미주 혹은 지수와 키가 일치하는 사람의 수
 */
public static int getCount(int[] data, int n, int m, int s)
{
    int count = 0;
    for(int i = 0 ; i < n ; i++)
    {




    }
    return count;
}
```

Problem 01D. 합 구하기

마찬가지로, 모든 원소를 차례로 한 번씩 순회한다는 특징을 활용해 모든 원소의 합을 계산하는 알고리즘을 작성해봅시다.

```java
/**
 * 정수 배열의 모든 원소의 합을 계산하는 함수
 *
 * @param data
 * @param n
 * @return data[0] ~ data[n-1]의 합
 */
public static int getSum(int[] data, int n) {
    int answer = 0;

    for(int i = 0 ; i < n ; i++)
    {









    }

    return answer;
}
```

Problem 01E. 합 구하기2

원하는 정답을 출력하기 위해서 필요한 정보는 무엇인가?

- ‘탑승 가능한’ 사람의 수 C
- ‘탑승 가능한’ 사람의 몸무게의 합 S
- 동아리의 놀이기구 탑승 가능 여부
  - 함께 놀이기구 탑승이 가능하다 ⇔ (S <= Q)

‘이 정보들은 주어진 데이터를 통해 계산할 수 있는가’를 고려해야 한다.

Problem 01E. 합 구하기2

현재 알고 있는 정보는 무엇인가?

- 사람의 수 N
- 각 사람의 몸무게 data[0] ~ data[n-1]
- 놀이기구 탑승 제한 체중 P
- 놀이기구 최대 하중 Q

- 탑승 가능한 사람의 조건
  - data[i] <= P

- '탑승 가능한' 사람의 수 C
  - P이하인 data[i]의 개수
- '탑승 가능한' 사람의 몸무게의 합 S
  - P이하인 data[i]의 총 합

```
public static void solve(int[] data, int n, int p, int q)
{
    int c = 0; //탑승 가능한 사람의 수
    int s = 0; //탑승 가능한 사람의 몸무게 총합

    for(int i = 0 ; i < n ; i++)
    {   //모든 몸무게 data[i]에 대하여

        if(                )
        {  //탑승 가능한 사람의 몸무게 data[i]에 대하여



        }
    }

    System.out.printf("%d %d\n", c, s);
    if(s <= q)
    {
        System.out.println("YES");
    }else{
        System.out.println("NO");
    }
}
```

데이터 탐색하기

배열의 원소들 중 구체적인 조건을 만족하는 데이터 혹은 그 위치를 찾는 방법을 고민해봅시다.

- 데이터의 위치를 계산해야 하는 경우, 아래의 경우들로 인해 데이터의 위치정보가 사라지거나 변하지 않도록 주의해야 합니다.
  - 입력 실수, 정렬, 순서 없는 자료구조 사용 등

탐색 문제에서는 아래의 사항을 꼭!!!! 유의하고 문제를 해결해야 합니다.

1. 조건을 만족하는 원소가 없는 경우
2. 조건을 만족하는 원소가 두 개 이상인 경우
3. 원소의 번호가 0번부터 시작하는지 1번부터 시작하는지.

문제의 조건과 나의 알고리즘에 따라서 적절히 처리해주어야 합니다.

Problem 01F. 탐색하기1

문제의 조건을 확인하여 코드를 완성해보자. 존재하지 않는 경우 어떻게 처리해야 할까?

```java
/**
 * 배열에서 특정 원소의 위치를 찾는 함수
 * @param data 중복 없는 정수 배열 data[0] ~ data[n-1]
 * @param n    배열의 크기 n
 * @param m    배열에서 찾고자 하는 원소
 * @return     원소가 존재한다면 인덱스를, 존재하지 않으면 -1을 반환한다.
 */
public static int findIndex(int[] data, int n, int m)
{
    int index = -1;

    for(int i = 0 ; i < n ; i ++)
    {



    }

    return index;
}
```

Problem 01G. 탐색하기2

탐색 문제에서 정답이 될 수 있는 후보가 여러가지라면? 정답을 정할 기준이 필요하다.

- 전부, 최대, 최소, 최초, 마지막 등 …
- 문제와 나의 알고리즘에 따라 적절하게 선택해야 한다.

Think> 문제에서 데이터의 번호를 1번부터 계산하는 경우, 배열의 0~(N-1)번이 아닌 1~N번 칸에 데이터를 저장하면 어떨까?

1. 그 배열 혼자 다른 인덱스 체계를 사용함에 있어서 실수하지 않도록 조심해야 한다.
2. 배열 전체에 대한 기능들(정렬함수 등)이 비정상적으로 동작할 수 있다.
3. 배열의 길이가 하나 늘어나므로 그로인한 혼선을 주의해야 한다.
    1. data.length == N+1 ?

위의 상황들을 고려하여 결정해야 한다.

Problem 01G. 탐색하기2

데이터를 차례로 탐색했을 때,

- 가장 처음 등장한 위치
- 가장 마지막에 등장한 위치를

어떻게 간단히 구할 수 있을까?

```java
/**
 * 배열에서 소속이 "AJOU"인 첫 원소와 마지막 원소를 출력하는 함수
 * @param school 각 사람들의 소속학교 정보 배열
 * @param n      사람들의 수
 */
public static void printIndexes(String[] school, int n)
{
    int first = -1; //존재하지 않으면 -1
    int last = -1;  //존재하지 않으면 -1

    for(int i = 0 ; i < n ; i++)
    {
        if(                                )
        {









        }
    }

    System.out.printf("%d %d\n", first, last);
}
```

Problem 01H. 탐색하기3

단순한 검사가 아닌 데이터들 간의 비교를 통한 탐색을 고려해봅시다.

평균과의 거리 $D_i = |data[i] - \mu|$가 가장 작은 인덱스 x 계산하기

- 평균을 계산한 이후
- 평균과 차이가 가장적은 i들 중
- 가장 작은(왼쪽에 있는) i

```
for(int i = 0 ; i < n ; i ++)
{
    if(  Di < Dx  ) //등호가 들어가면 어떻게 될까요?
    {    //더 작은 거리를 가진 i가 나타났을 때만
         x = i;
    }

}
```

Problem 01H. 탐색하기3

가능하다면 계산 과정에서 실수를 사용한 계산은 최소화하는 것이 좋습니다. 부동 소수점의 오차로 인해 예상치 못한 오동작이 생길 수 도 있습니다.

대부분의 비교식은 조금 변환하면 정수의 계산으로 바꿀 수 있습니다.

$D_i = |data[i] - \mu|$ 라면

- $nD_i = |n \times data[i] - S|$ 가 된다. S는 모든 원소의 합.
- n과 D모두 양수이므로 성립한다.
- 모든 연산의 결과는 정수가 된다.

대소관계를 비교하는 경우, 대소관계만 유지 된다면 식을 변형해도 된다.

$$|data[i] - \mu| < |data[j] - \mu| \Leftrightarrow |n \times data[i] - S| < |n \times data[j] - S|$$

항상, 실수 계산은 최소화한 후 최종계산에서만 실수로 변환하는게 좋다.

Problem 01H. 탐색하기3

```
public static int findIndex(int[] data, int n)
{
    int x = 0; //존재하지 않는 경우는 없으므로, 일단 0이라고 가정
    int S = 0; //모든 데이터의 합

    for(int i = 0 ; i < n ; i++)
    {
        S += data[i];
    }

    for(int i = 0 ; i < n ; i ++)
    {
        int dx =                                    //i까지의 원소들 중 평균과의 최소거리
        int di =                                   //현재 원소(data[i])와 평균과의 거리
        if( di < dx )
        {
            x = i;
        }
    }

    return x + 1; //실제 번호는 1부터 시작하므로 증가
}
```

Problem 01I. 선택 정렬 구현하기

아래의 과정을 반복하는 알고리즘을 선택 정렬(Selection Sort)라고 한다. 이 알고리즘이 어떻게 배열을 오름차순으로 정렬함을 보장할 수 있을까?

또, 배운 내용을 바탕으로 아래 과정을 구현할 수 있을까?

1. 주어진 범위에서 **최소 값의 위치를** 찾는다.
2. 최소 값을 해당 범위의 **가장 앞 숫자와 자리를 바꾼다**.
3. 이후, 해당 범위의 가장 앞 자리를 제외한 나머지 범위에 대해 위의 **과정을 반복**한다.

Problem 01I. 선택 정렬 구현하기

입력으로 주어진 원본 배열 A와 이를 오름차순으로 정렬한 배열 B가 있다고 하자. 원소의 순서가 다를 뿐 두 배열은 집합적으로는 같은 집합이다.

즉, 어떤 인덱스 k에 대하여 배열 B의 원소 b_k는 오름차순으로 (k+1)번째로 작은 값 이며. 마찬가지로 배열 A에서 오름차순으로 (k+1)번째로 작은 값도 b_k 와 같다.

순서가 다르지만 같은 원소들을 가진 배열이므로, 항상 성립한다.

Problem 01I. 선택 정렬 구현하기

배운 내용으로 주어진 범위에서 최소값을 쉽게 찾을 수 있다. 또한, 해당 최소값을 제외한 범위에서 다시 최소값을 찾으면 그 다음으로 작은 값을 찾을 수 있다.

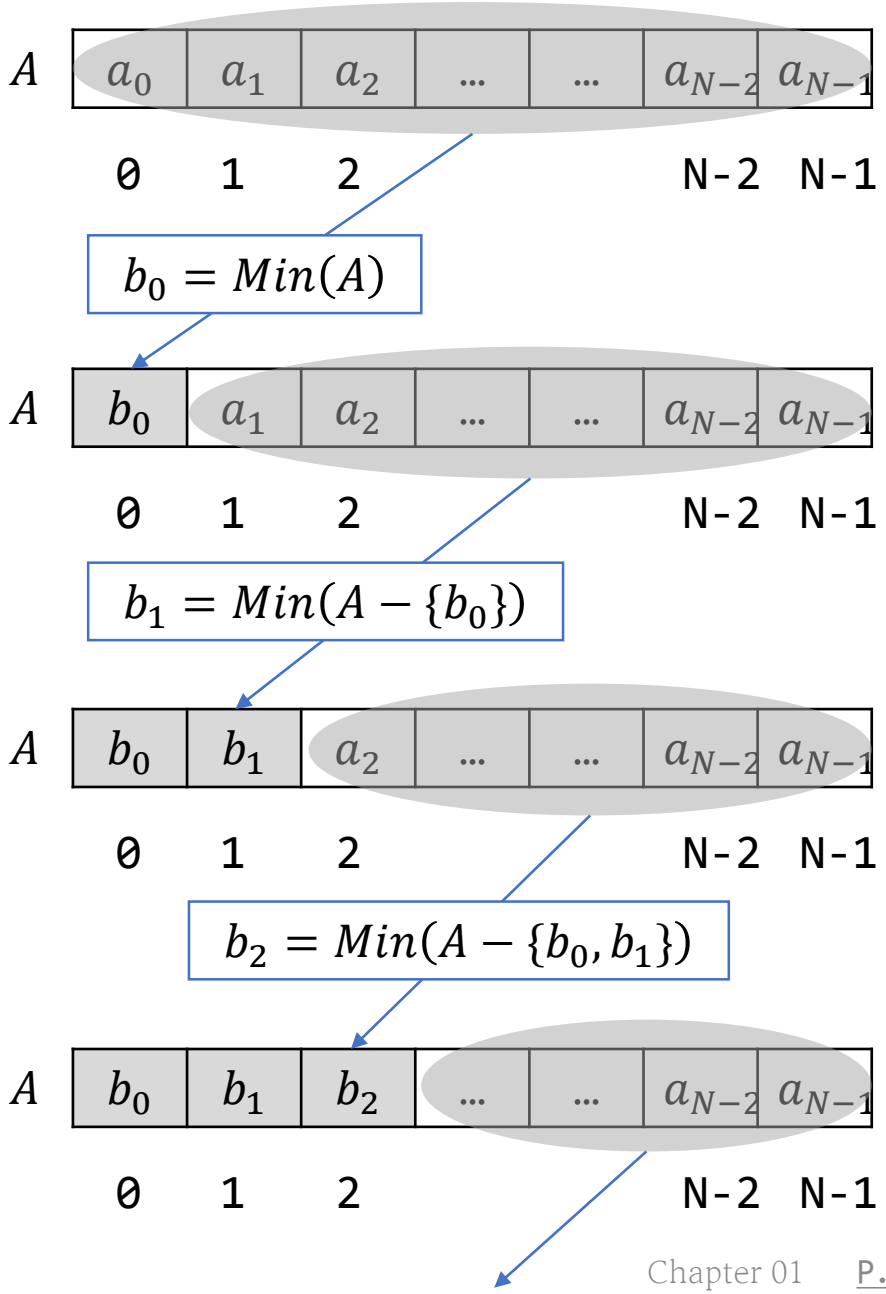
$b_0 = Min(A) = Min(B)$

$b_1 = Min(A - \{b_0\}) = Min(B - \{b_0\})$

$b_2 = Min(A - \{b_0, b_1\}) = Min(B - \{b_0, b_1\})$

…

작은 숫자부터 차례로 나열하면, 오름차순이다.



Problem 01I. 선택 정렬 구현하기

주어진 범위 안에서 최소값을 찾아 인덱스를 반환하는 함수를 구현해보자.

```
/**
 * 주어진 범위의 최소값의 위치를 반환하는 함수
 * @param data  데이터 배열
 * @param n     배열의 크기
 * @param begin 탐색 할 가장 첫(왼쪽) 인덱스
 * @param end   탐색 할 가장 마지막(오른쪽) 인덱스
 * @return  data[begin] ~ data[end] 중 가장 작은 원소의 인덱스
 */
public static int getMinIndexInRange(int[] data, int n, int begin, int end)
{
    int index = begin;









    return index;
}
```

Problem 01J. 합 구하기3

복잡하게 나열 된 문제의 요구사항을 규칙적으로 분석해보자.
순차적인 규칙을 가지는 경우 대부분의 경우 반복문으로 구현될 수 있다.

각 항을 차례로 나열해보자. 각 항은 하나의 반복문으로 계산 될 수 있다.

- $1 +$
- $1 + 2 +$
- $1 + 2 + 3 +$
- …
- $1 + 2 + \cdots + (i - 1) + i$
- …
- $1 + 2 + 3 + \ \ldots + (N - 1) + N$

- $RangeSum(1, 1) +$
- $RangeSum(1, 2) +$
- $RangeSum(1, 3) +$
- …
- $RangeSum(1, i) +$
- …
- $RangeSum(1, N)$

Problem 01J. 합 구하기3

1부터 i까지의 정수 합을 구하는 함수를 구현해보자.

그리고 고민해보자.

- 합을 구하는 과정에서 결과가 int의 범위를 벗어나지는 않는가?
- 이런 누적합을 더 빠르고 간결하게 구하는 방법은 없는가?

```
public static int getRangeSumFromOne(int i) {
    int sum = 0;









    return sum;
}
```

Problem 01J. 합 구하기3

앞서 구현한 함수를 사용해 문제에서 주어진 정답을 계산하는 함수를 완성해보자. 그리고 아래의 사항들을 고민해보자.

- 합을 구하는 과정에서 결과가 int의 범위를 벗어나지는 않는가?
- 이런 누적합을 더 빠르고 간결하게 구하는 방법은 없는가?

```
public static _____ getAnswer(int N)
{




}
```

Problem 01J. 합 구하기3

Think> 문제의 입력 범위에서 N의 최대값이 1,100이어도 long형 데이터를 사용해야 할까?

32비트(4바이트) int형 변수의 표현 범위는 -2,147,483,648 ~ 2,147,483,647이다.

Appendix. Primitive Data Types of Java

타입	크기	표현 범위	설명
byte	1 byte	$[-128, 127]$	가장 기본적인 데이터 표현 단위
short	2 byte	$[-32768, 32767]$	작은 범위의 정수형 데이터
int	4 byte	$[-2^{31}, 2^{31}-1]$ 약 +- 21억	일반적인 정수형 데이터
long	8 byte	$[-2^{63}, 2^{63}-1]$	큰 범위의 정수형 데이터
float	4 byte	0x0.000002P-126f ~ 0x1.fffffeP+127f	실수형 데이터
double	8 byte	0x0.0000000000001P-1022 ~ 0x1.fffffffffffffP+1023	정밀도 높은 실수형 데이터
char	2 byte (c는 1byte)	$[0, 2^{16}-1]$	문자 데이터
boolean	1 bit	true(1), false(0)	논리 데이터

Fin.

감사합니다.

* 이 PPT는 네이버에서 제공한 나눔글꼴과 아모레퍼시픽의 아리따부리 폰트를 사용하고 있습니다.