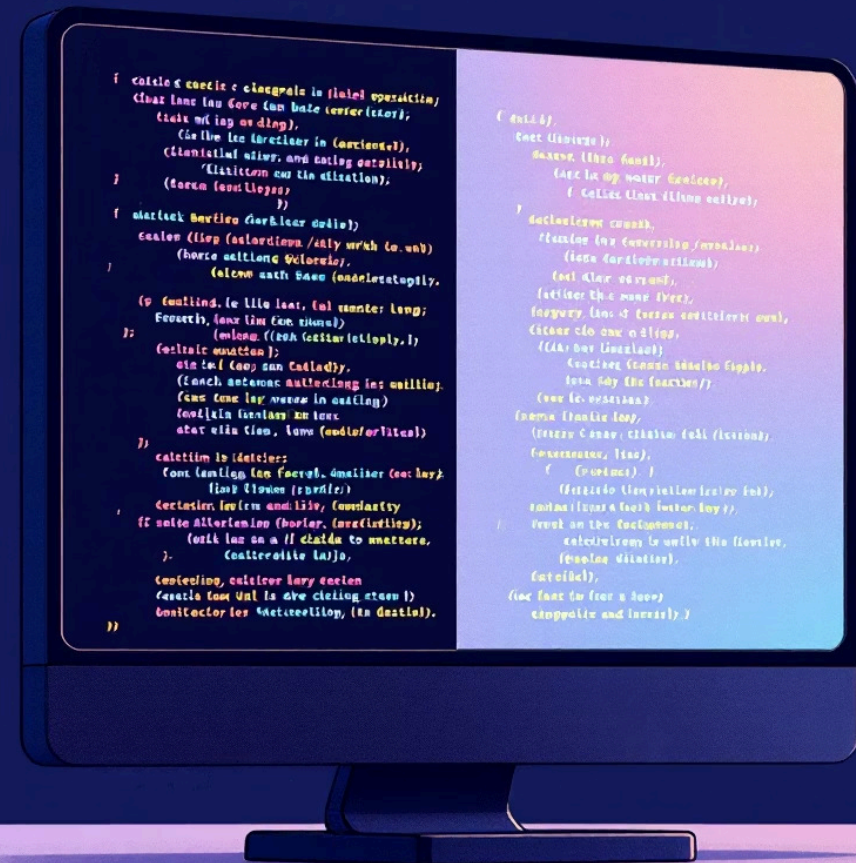


에꼴 수업 1일차(2025.09.06)

소프트웨어학과 손영빈(2021663041)



수 정렬하기

파이썬

```
# 수 정렬하기
n = int(input())
arr = []
for i in range(n):
    arr.append(int(input()))
arr.sort()
for i in arr:
    print(i)
```

자바

```
import java.util.Scanner;
import java.util.Arrays;

public class Main {
    public static void main(String[]
args) {
        Scanner sc = new
Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];

        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        Arrays.sort(arr);

        for (int i : arr) {
            System.out.println(i);
        }
        sc.close();
    }
}
```

자바스크립트

```
// Node.js 환경 기준
const readline =
require('readline');
const rl =
readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let n;
const arr = [];
let count = 0;

rl.on('line', (line) => {
    if (!n) {
        n = parseInt(line);
    } else {
        arr.push(parseInt(line));
        count++;
        if (count === n) {
            arr.sort((a, b) => a - b);
            arr.forEach(num =>
console.log(num));
            rl.close();
        }
    }
});
```

각 언어별 특징: 파이썬은 간결한 코드로 구현되며, 자바는 타입 선언이 필요합니다. 자바스크립트는 비동기 입력 처리를 위해 이벤트 기반 코드가 사용됩니다.

숫자의 합 구하기

파이썬

```
# 숫자의 합 구하기
n = int(input())
arr = list(map(int, input()))
print(sum(arr))
```

파이썬의 `map` 함수와 `sum` 내장 함수를 활용하여 간결하게 구현

자바

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new
Scanner(System.in);
        int n = sc.nextInt();
        String nums = sc.next();

        int sum = 0;
        for (int i = 0; i < n; i++) {
            sum += nums.charAt(i) - '0';
        }

        System.out.println(sum);
        sc.close();
    }
}
```

자바스크립트

```
const readline = require('readline');
const rl =
readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let n;
let lineCount = 0;

rl.on('line', (line) => {
    if (lineCount === 0) {
        n = parseInt(line);
        lineCount++;
    } else {
        const arr =
line.split('').map(Number);
        console.log(arr.reduce((a, b)
=> a + b, 0));
        rl.close();
    }
});
```

숫자의 합을 구하는 과정에서 각 언어의 배열 처리 방식과 합계 산출 방법의 차이를 볼 수 있습니다. 자바스크립트는 `reduce` 메서드를 활용합니다.

구간 합 계산하기

1

파이썬

```
import sys
input = sys.stdin.readline

n, m = map(int, input().split())
arr = list(map(int, input().split()))
arr2 = [0] * (n+1)

for i in range(1, n+1):
    arr2[i] = arr2[i-1] + arr[i-1]

for _ in range(m):
    a, b = map(int, input().split())
    print(arr2[b] - arr2[a-1])
```

2

자바

```
import java.util.Scanner;

public class Main {
    public static void main(String[]
args) {
        Scanner sc = new
Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();

        int[] arr = new int[n];
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        int[] prefixSum = new
int[n+1];
        for (int i = 1; i <= n; i++) {
            prefixSum[i] = prefixSum[i-
1] + arr[i-1];
        }

        for (int i = 0; i < m; i++) {
            int a = sc.nextInt();
            int b = sc.nextInt();

            System.out.println(prefixSum[b] -
prefixSum[a-1]);
        }
        sc.close();
    }
}
```

3

자바스크립트

```
const readline =
require('readline');
const rl =
readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let lines = [];
rl.on('line', (line) => {
    lines.push(line);
});

rl.on('close', () => {
    const [n, m] = lines[0].split('')
.map(Number);
    const arr = lines[1].split('')
.map(Number);

    const prefixSum = new
Array(n+1).fill(0);
    for (let i = 1; i <= n; i++) {
        prefixSum[i] = prefixSum[i-1]
+ arr[i-1];
    }

    for (let i = 0; i < m; i++) {
        const [a, b] = lines[i+2].split('')
.map(Number);
        console.log(prefixSum[b] -
prefixSum[a-1]);
    }
});
```

구간 합 알고리즘은 누적 합(prefix sum)을 이용하여 구간의 합을 $O(1)$ 시간에 계산합니다. 세 언어 모두 동일한 알고리즘 로직을 사용하지만, 입출력 처리 방식에 차이가 있습니다.

수들의 합 (투 포인터)

투 포인터 알고리즘

투 포인터 알고리즘은 두 개의 포인터를 활용하여 배열이나 리스트에서 원하는 결과를 찾는 방법입니다. 여기서는 연속된 자연수의 합이 특정 값이 되는 경우를 찾습니다.

왼쪽(left)과 오른쪽(right) 포인터를 이동시키면서 합을 조절합니다:

- 합이 목표값보다 작으면 right를 증가
- 합이 목표값보다 크면 left를 증가
- 합이 목표값과 같으면 카운트 증가


파이썬 코드 (성공)

```
import sys
input = sys.stdin.readline

n = int(input())
count = 0
left, right = 1, 1
s = 1 # 현재 구간 [left, right]의 합

while right <= n:
    if s == n:
        count += 1
        s -= left
        left += 1
    elif s < n:
        right += 1
    if right > n:
        break
    s += right
    else: # s > n
        s -= left
        left += 1

print(count)
```

 시간 초과 실패 코드

아래 코드는 시간 복잡도가 높아 큰 입력값에서 시간 초과가 발생합니다.

```
import sys
input =sys.stdin.readline
n=int(input())
count=0
s=0
for i in range(0,n):
    s=0
    for j in range(i,n):
        s+=j
        if s==n:
            count+=1
            break
        elif s>n:
            break
print(count)
```

수들의 합 (투 포인터) - 변환 코드

자바

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        int count = 0;
        int left = 1, right = 1;
        int sum = 1; // 현재 구간 [left, right]의 합

        while (right <= n) {
            if (sum == n) {
                count++;
                sum -= left;
                left++;
            } else if (sum < n) {
                right++;
                if (right > n) {
                    break;
                }
                sum += right;
            } else { // sum > n
                sum -= left;
                left++;
            }
        }

        System.out.println(count);
        sc.close();
    }
}
```

자바스크립트

```
const readline = require('readline');
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

rl.on('line', (line) => {
    const n = parseInt(line);

    let count = 0;
    let left = 1, right = 1;
    let sum = 1; // 현재 구간 [left, right]의 합

    while (right <= n) {
        if (sum === n) {
            count++;
            sum -= left;
            left++;
        } else if (sum < n) {
            right++;
            if (right > n) {
                break;
            }
            sum += right;
        } else { // sum > n
            sum -= left;
            left++;
        }
    }

    console.log(count);
    rl.close();
});
```

투 포인터 알고리즘은 세 언어 모두 거의 동일한 로직으로 구현됩니다. 이는 알고리즘의 핵심 로직이 언어와 무관하게 일관되게 적용된다는 것을 보여줍니다. 자바와 자바스크립트에서도 파이썬과 동일한 시간 복잡도 $O(N)$ 으로 동작합니다.

좋다 (투 포인터 활용)



파이썬

```
import sys
input = sys.stdin.readline
n = int(input())
arr = list(map(int, input().split()))
arr.sort()
good = 0

for i in range(n):
    target = arr[i]
    l, r = 0, n - 1
    found = False

    while l < r:
        if l == i:
            l += 1
            continue
        if r == i:
            r -= 1
            continue

        s = arr[l] + arr[r]
        if s == target:
            found = True
            break
        elif s < target:
            l += 1
        else:
            r -= 1

    if found:
        good += 1

print(good)
```

이 알고리즘은 주어진 배열에서 "좋은 수"를 찾는 문제를 해결합니다. "좋은 수"란 배열 내 다른 두 수의 합으로 표현될 수 있는 수를 의미합니다.

시간 복잡도: $O(N^2)$

탐색 과정에서 주의할 점:

- 현재 타겟 인덱스(i)와 같은 포인터는 건너뛰어야 함
- 두 포인터(l, r)가 서로 다른 위치에 있어야 함
- 정렬된 배열에서 투 포인터를 사용하여 효율적으로 탐색

좋다 (투 포인터) - 변환 코드

자바

```
import java.util.Arrays;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];

        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        Arrays.sort(arr);
        int good = 0;

        for (int i = 0; i < n; i++) {
            int target = arr[i];
            int l = 0, r = n - 1;
            boolean found = false;

            while (l < r) {
                if (l == i) {
                    l++;
                    continue;
                }
                if (r == i) {
                    r--;
                    continue;
                }

                int sum = arr[l] + arr[r];
                if (sum == target) {
                    found = true;
                    break;
                } else if (sum < target) {
                    l++;
                } else {
                    r--;
                }
            }

            if (found) {
                good++;
            }
        }

        System.out.println(good);
        sc.close();
    }
}
```

자바스크립트

```
const readline = require('readline');
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let lines = [];
rl.on('line', (line) => {
    lines.push(line);
});

rl.on('close', () => {
    const n = parseInt(lines[0]);
    const arr = lines[1].split(' ').map(Number);

    arr.sort((a, b) => a - b);
    let good = 0;

    for (let i = 0; i < n; i++) {
        const target = arr[i];
        let l = 0, r = n - 1;
        let found = false;

        while (l < r) {
            if (l === i) {
                l++;
                continue;
            }
            if (r === i) {
                r--;
                continue;
            }

            const sum = arr[l] + arr[r];
            if (sum === target) {
                found = true;
                break;
            } else if (sum < target) {
                l++;
            } else {
                r--;
            }
        }

        if (found) {
            good++;
        }
    }

    console.log(good);
});
```

세 언어 모두 동일한 알고리즘을 구현했지만, 자바스크립트에서는 `sort()` 메서드에 비교 함수를 전달해야 한다는 점에 주의해야 합니다. 자바스크립트의 기본 `sort()`는 문자열 정렬이기 때문입니다.

슬라이딩 윈도우와 덱(Deque) 활용

알고리즘 핵심

슬라이딩 윈도우는 고정 크기의 창을 이동시키며 범위 내 데이터를 처리하는 기법입니다.

이 문제에서는 덱(Deque)을 활용하여 윈도우 내 최소값을 효율적으로 관리합니다:

1. 단조 증가 상태 유지 (큰 값은 필요 없음)
2. 현재 윈도우를 벗어난 값 제거
3. 덱의 맨 앞이 항상 현재 윈도우의 최소값

시간 복잡도: $O(N)$

파이썬 코드

```
from collections import deque
import sys

input = sys.stdin.readline

N, L = map(int, input().split())
arr = list(map(int, input().split()))
dq = deque() # (idx, val) ; 값 기준 단조 증가 유지

for i in range(N):
    x = arr[i]
    # (A) 뒤에서 현재 값 이상(>=)은 제거 → 단조 증가 유지
    while dq and dq[-1][1] >= x:
        dq.pop()
    # (B) 현재 원소 삽입
    dq.append((i, x))
    # (C) 윈도우에서 벗어난(너무 왼쪽) 원소 제거
    # 윈도우 시작 = i-L+1 이므로, idx <= i-L 이면 범위 밖
    while dq and dq[0][0] <= i - L:
        dq.popleft()
    # (D) 덱 맨 앞의 값이 현재 윈도우 최소
    if i == N - 1:
        print(dq[0][1]) # 마지막은 개행
    else:
        print(dq[0][1], end=' ') # 나머지는 공백으로 구분
```

슬라이딩 윈도우와 덱(Deque) 활용-변환 코드

자바

```
import java.io.*;
import java.util.*;

public class Main {
    public static void main(String[] args) throws
    Exception {
        BufferedReader br = new BufferedReader(new
        InputStreamReader(System.in));
        StringTokenizer st;

        st = new StringTokenizer(br.readLine());
        int N = Integer.parseInt(st.nextToken());
        int L = Integer.parseInt(st.nextToken());

        int[] arr = new int[N];
        st = new StringTokenizer(br.readLine());
        for (int i = 0; i < N; i++) arr[i] =
        Integer.parseInt(st.nextToken());

        // 덱에 (idx, val)
        Deque<int[]> dq = new ArrayDeque<>();
        StringBuilder sb = new StringBuilder();

        for (int i = 0; i < N; i++) {
            int x = arr[i];

            // 뒤에서 현재값 이상(>=) 제거 → 단조 증가 유지
            while (!dq.isEmpty() && dq.peekLast()[1] >= x)
                dq.pollLast();

            // 삽입
            dq.addLast(new int[]{i, x});

            // 윈도우 벗어난 값 제거 (idx <= i - L)
            while (!dq.isEmpty() && dq.peekFirst()[0] <= i -
            L) dq.pollFirst();

            // 현재 최소(맨 앞)
            sb.append(dq.peekFirst()[1]);
            if (i < N - 1) sb.append(' ');
        }

        System.out.println(sb.toString());
    }
}
```

자바스크립트

```
const fs = require("fs");
const input = fs.readFileSync(0, "utf-
8").trim().split("\n");

const [N, L] = input[0].trim().split(" ").map(Number);
const arr = input[1].trim().split(" ").map(Number);

// 덱에 [idx, val]; shift() 비용 방지를 위해 head 포인터 사용
const dq = [];
let head = 0;
const out = [];

for (let i = 0; i < N; i++) {
    const x = arr[i];

    // 뒤에서 현재값 이상(>=) 제거 → 단조 증가 유지
    while (dq.length > head && dq[dq.length - 1][1] >= x)
        dq.pop();

    // 삽입
    dq.push([i, x]);

    // 윈도우 벗어난 값 제거 (idx <= i - L)
    while (dq.length > head && dq[head][0] <= i - L)
        head++;

    // 현재 최소(맨 앞)
    out.push(dq[head][1]);
}

console.log(out.join(" "));
```

스택 수열

문제 정의

주어진 수열을 1부터 N까지의 자연수를 순서대로 스택에 넣었다가 빼는 연산을 통해 만들 수 있는지 판별하는 문제입니다. 만들 수 있다면 어떤 연산을 수행해야 하는지, 없다면 "NO"를 출력합니다.

알고리즘 핵심

- 스택에 1부터 순서대로 숫자를 **PUSH**합니다.
- PUSH하는 과정에서 현재 스택의 최상단 숫자가 목표 수열의 다음 숫자와 일치하면 해당 숫자를 **POP**합니다.
- 만약 스택의 최상단 숫자가 목표 수열의 숫자와 다르고, 더 이상 PUSH할 숫자가 없다면 주어진 수열은 만들 수 없습니다.
- 각 PUSH 연산에는 '+', POP 연산에는 '-'를 기록합니다.

이 방법으로 스택의 특징을 활용하여 수열 생성 가능 여부를 효율적으로 판단할 수 있습니다.

파이썬 코드

```
n = int(input())
sequence = [int(input()) for _ in range(n)]

stack = []
result = []
current = 1 # 1부터 N까지의 자연수를 나타냄
possible = True

for num in sequence:
    # current가 num보다 작거나 같으면, num이 스택에 들어올 때까지 push
    while current <= num:
        stack.append(current)
        result.append('+')
        current += 1

    # 스택의 최상단이 num과 같으면 pop
    if stack and stack[-1] == num:
        stack.pop()
        result.append('-')
    else:
        # 스택의 최상단이 num과 다르면 만들 수 없는 수열
        possible = False
        break

if possible:
    print('\n'.join(result))
else:
    print('NO')
```

스택으로 오름차순 - 변환 코드

주어진 수열을 만들기 위해 스택에 숫자를 넣고 빼는 과정(+와 - 연산)을 시뮬레이션하는 문제입니다. 스택의 LIFO(Last-In, First-Out) 특성을 이해하고 구현하는 것이 중요합니다.

자바	자바스크립트
<pre>import java.io.BufferedReader; import java.io.InputStreamReader; import java.io.IOException; import java.util.Stack; import java.util.ArrayList; public class Main { public static void main(String[] args) throws IOException { BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); int n = Integer.parseInt(br.readLine()); int[] sequence = new int[n]; for (int i = 0; i < n; i++) { sequence[i] = Integer.parseInt(br.readLine()); } Stack<Integer> stack = new Stack<>(); ArrayList<String> result = new ArrayList<>(); int current = 1; boolean possible = true; for (int num : sequence) { while (current <= num) { stack.push(current); result.add("+"); current++; } if (!stack.isEmpty() && stack.peek() == num) { stack.pop(); result.add("-"); } else { possible = false; break; } } if (possible) { for (String op : result) { System.out.println(op); } } else { System.out.println("NO"); } } }</pre>	<pre>const readline = require('readline'); const rl = readline.createInterface({ input: process.stdin, output: process.stdout }); let lines = []; rl.on('line', (line) => { lines.push(line); }).on('close', () => { const n = parseInt(lines[0]); const sequence = []; for (let i = 1; i <= n; i++) { sequence.push(parseInt(lines[i])); } const stack = []; const result = []; let current = 1; let possible = true; for (const num of sequence) { while (current <= num) { stack.push(current); result.push('+'); current++; } if (stack.length > 0 && stack[stack.length - 1] === num) { stack.pop(); result.push('-'); } else { possible = false; break; } } if (possible) { console.log(result.join('\n')); } else { console.log("NO"); } });</pre>

세 언어 모두 스택의 기본적인 LIFO(Last-In, First-Out) 특성을 활용하여 동일한 로직을 구현합니다. 자바의 경우 `java.util.Stack` 클래스를, 자바스크립트는 배열의 `push()`와 `pop()` 메서드를 활용하여 스택을 구현할 수 있습니다. 입력 처리 방식은 각 언어의 표준 입출력 방식에 맞춰 변경됩니다.

큐/덱 활용 - 카드 버리기

제일 위에 있는 카드를 버리고, 그 다음 카드를 맨 밑으로 옮기는 작업을 반복하여 마지막에 남는 카드를 찾는 문제입니다. 큐(Queue) 또는 덱(Deque)의 기본 연산을 활용하여 효율적으로 해결할 수 있습니다.

파이썬 코드

```
from collections import deque
import sys
input = sys.stdin.readline

n = int(input())
dq = deque(range(1, n+1)) # 1부터 n까지 큐에 담기

while len(dq) > 1:
    dq.popleft() # 제일 위 버림
    dq.append(dq.popleft()) # 다음 카드를 뒤로 옮김

print(dq[0])
```

자바 코드

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.util.Deque;
import java.util.ArrayDeque;

public class Main {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        int n = Integer.parseInt(br.readLine());

        Deque<Integer> dq = new ArrayDeque<>();
        for (int i = 1; i <= n; i++) {
            dq.addLast(i); // 큐에 1부터 n까지 담기
        }

        while (dq.size() > 1) {
            dq.removeFirst(); // 제일 위 버림
            dq.addLast(dq.removeFirst()); // 다음 카드를 뒤로 옮김
        }

        System.out.println(dq.getFirst());
    }
}
```

자바스크립트 코드

```
const readline = require('readline');
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

rl.on('line', (line) => {
    const n = parseInt(line);
    const dq = []; // 배열을 큐처럼 사용

    for (let i = 1; i <= n; i++) {
        dq.push(i); // 1부터 n까지 큐에 담기
    }

    let head = 0; // 큐의 앞을 가리키는 포인터
    while (dq.length - head > 1) {
        head++; // 제일 위 버림 (실제 제거는 아니지만, 논리적으로 버려짐)
        dq.push(dq[head]); // 다음 카드를 뒤로 옮김
        head++; // 옮긴 카드도 논리적으로 버려짐
    }

    console.log(dq[head]);
    rl.close();
});
```

세 언어 모두 큐(또는 덱)의 개념을 활용하여 문제의 로직을 구현합니다. 파이썬은 `collections.deque`를, 자바는 `java.util.Deque` 인터페이스의 `ArrayDeque` 구현체를 사용합니다. 자바스크립트는 배열을 사용하여 큐의 기능을 시뮬레이션하며, `head` 포인터를 사용하여 `shift()` 연산의 성능 저하를 피하는 방식을 적용했습니다.

출처

ChatGPT

감사합니다.

발표를 들어 주셔서 진심으로 감사합니다.

