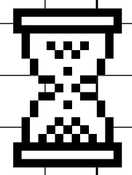
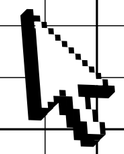
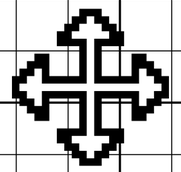
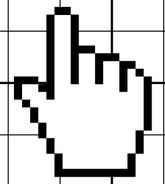
01. 기본 알고리즘
02. 기본 자료구조

알고리즘

1 김율리아 2 정재홍

발표일: 2022년 3월 5일 (토)

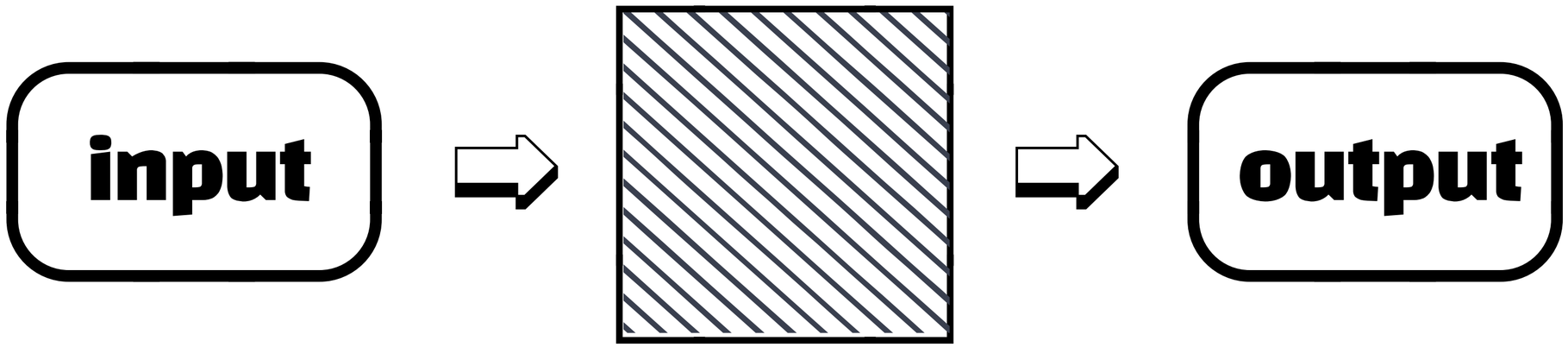
목차

01

기본 알고리즘

컴퓨팅

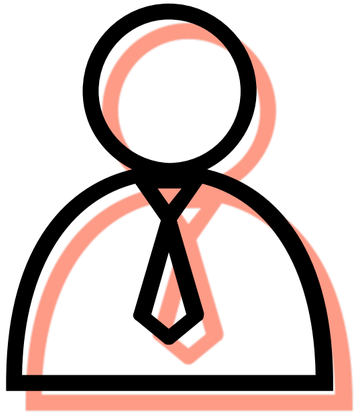
입력을 받아 그 입력을 처리한 후 출력하는 과정



Algorithms?

입력값을 출력값의 형태로 바꾸기 위해
어떤 명령들이 수행되어야 하는지에 대한
규칙들의 **순서적 나열**

홍길동 찾기



1. 전화번호부를 집어 든다.
2. 전화번호부의 중간을 편다.
3. 페이지를 본다.
4. 만약 홍길동이 페이지에 있으면,
   홍길동에게 전화한다.
5. 그렇지 않고 만약 홍길동이 앞 페이지에 있으면,
   앞 페이지의 절반을 편다.
   3번째 줄부터 다시 실행한다.
6. 그렇지 않고 만약 홍길동이 뒷 페이지에 있으면,
   뒷 페이지의 절반을 편다.
   3번째 줄부터 다시 실행한다.
7. 그러지 않으면,
   그만둔다.

1

함수

1. 전화번호부를 집어 든다.
2. 전화번호부의 중간을 편다.
3. 페이지를 본다.
4. 만약 홍길동이 페이지에 있으면,
   홍길동에게 전화한다.
5. 그렇지 않고 만약 홍길동이 앞 페이지에 있으면,
   앞 페이지의 절반을 편다.
   3번째 줄부터 다시 실행한다.
6. 그렇지 않고 만약 홍길동이 뒷 페이지에 있으면,
   뒷 페이지의 절반을 편다.
   3번째 줄부터 다시 실행한다.
7. 그러지 않으면,
   그만둔다.

2

조건

1. 전화번호부를 집어 든다.
2. 전화번호부의 중간을 편다.
3. 페이지를 본다.
4. 만약 홍길동이 페이지에 있으면,
   홍길동에게 전화한다.
5. 그렇지 않고 만약 홍길동이 앞 페이지에 있으면,
   앞 페이지의 절반을 편다.
   3번째 줄부터 다시 실행한다.
6. 그렇지 않고 만약 홍길동이 뒷 페이지에 있으면,
   뒷 페이지의 절반을 편다.
   3번째 줄부터 다시 실행한다.
7. 그러지 않으면,
   그만둔다.

3

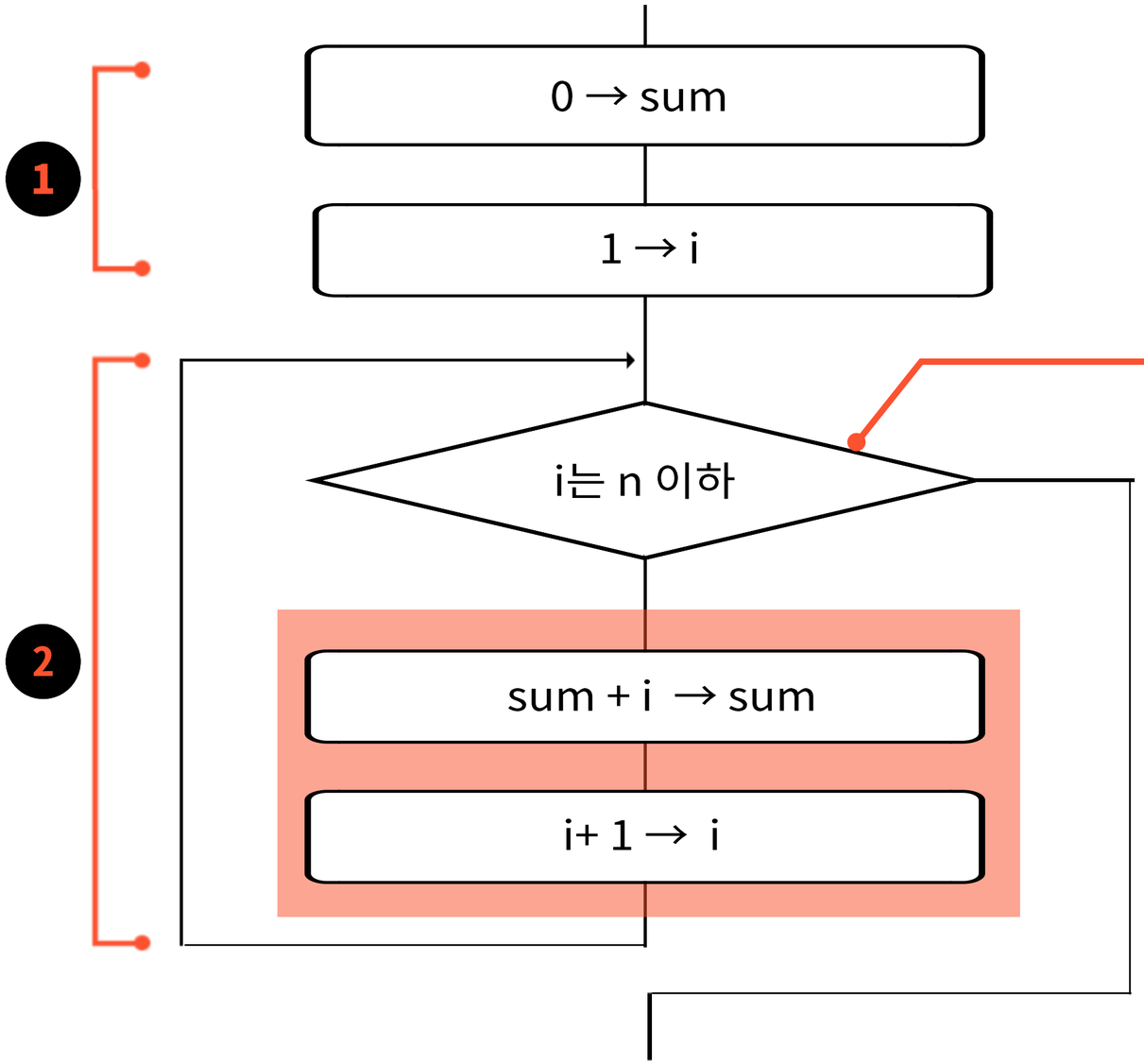
boolean

1. 전화번호부를 집어 든다.
2. 전화번호부의 중간을 편다.
3. 페이지를 본다.
4. 만약 홍길동이 페이지에 있으면,
   홍길동에게 전화한다..
5. 그렇지 않고 만약 홍길동이 앞 페이지에 있으면,
   앞 페이지의 절반을 편다.
   3번째 줄부터 다시 실행한다.
6. 그렇지 않고 만약 홍길동이 뒷 페이지에 있으면,
   뒷 페이지의 절반을 편다.
   3번째 줄부터 다시 실행한다.
7. 그러지 않으면,
   그만둔다.

4

반복(Loop)

1. 전화번호부를 집어 든다.
2. 전화번호부의 중간을 편다.
3. 페이지를 본다.
4. 만약 홍길동이 페이지에 있으면,
   홍길동에게 전화한다.
5. 그렇지 않고 만약 홍길동이 앞 페이지에 있으면,
   앞 페이지의 절반을 편다.
   3번째 줄부터 다시 실행한다.
6. 그렇지 않고 만약 홍길동이 뒷 페이지에 있으면,
   뒷 페이지의 절반을 편다.
   3번째 줄부터 다시 실행한다.
7. 그러지 않으면,
   그만둔다.



루프 본문

: 반복의 대상이 되는 '명령문'

i 와 sum 값 변함

1 sum : 합을 저장하는 변수 // 0

i : 반복을 제어하기 위한 변수 //1

2 변수 i의 값이 n이하인 동안
i 의 값이 1씩 증가하면서
루프 본문을 n회 반복실행

for문

```
for ( 초기치; 조건문; 증감치 ) {
    <수행할 문장>;
    <수행할 문장>;
}
```

```
for(int i=0;i<5;i++) {
 System.out.print(i + " ");
}
```

==========

0 1 2 3 4

while문

```
while ( 조건문 ) {
    <수행할 문장>;
    <수행할 문장>;
}
```

```
int i = 0;
while(i < 5) {
 System.out.print(i + " ");
 i++;
}
```

==========

0 1 2 3 4

do-while문

do-while문은 무조건 한번 이상은 실행

```
do {
    <수행할 문장>;
}while( 조건문 )
```

```
int i = 0;
do {
    System.out.print(i + " ");
    i++;
 } while (i<5);

======================
0 1 2 3 4
```

foreach문

가변적인 배열 or 리스트의 크기를

일일이 구할 필요가 없음

```
int [] arr = {0,1,2,3,4};
for(int a : arr) {
 System.out.print(a + " ");
}

======================
0 1 2 3 4
```

continue/break문

break : 만나는 즉시 반복문 **전체 탈출**

continue : 만나면 해당 반복 **부분 탈출** 후 다음 반복 실행

```
for(int i=0;i<10;i++) {
   if(i==5) {
   break;
   }
   System.out.print(i+ " ");
}
System.out.println("반복문 종료");

for(int i=0;i<6;i++) {
        if(i==3) {
         continue;
         }
         System.out.print(i + " ");
        }


======================
0 1 2 3 4 반복문 종료
0 1 2 4 5
```

시간복잡도
(Time Complexity)

알고리즘을 수행하기 위해
프로세스가 수행해야하는
연산을 수치화 한 것

(명령어의 실행시간은
프로그래밍 언어마다 다르므로)

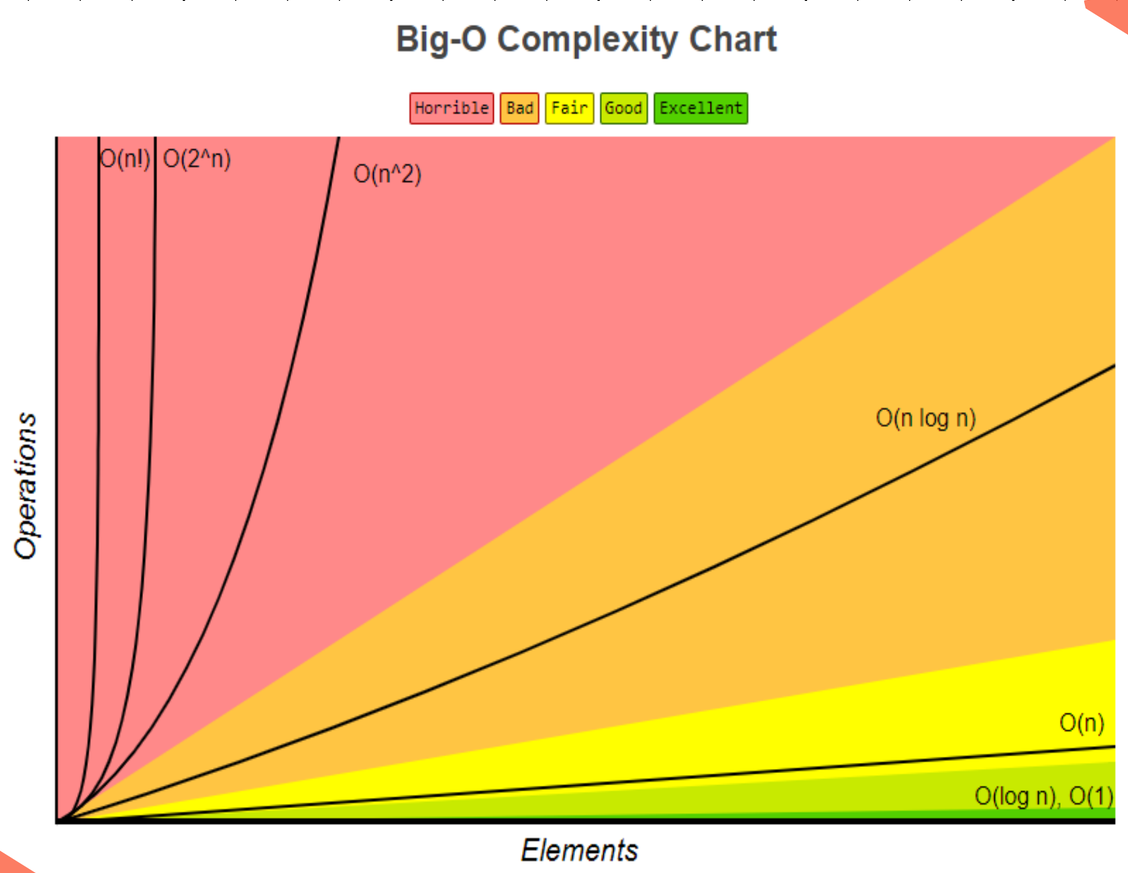
점금적 표기법(Asymptotic notation)

(점근적: 가장 큰 영향을 주는 항만 계산한다

상수와 계수들을 제거해
알고리즘의 실행시간에서 중요한
성장률(입력값의 크기에 따른 함수의 증가량)에 집중

최상의 경우 : **오메가 표기법** (Big-Ω Notation)
평균의 경우 : **세타 표기법** (Big-θ Notation)
최악의 경우 : **빅오 표기법** (Big-O Notation)

출처 : https://blog.chulgil.me/algorithm/



출처 : https://www.bigocheatsheet.com/

Big O 표기법

- **'O' 는 on the order of의 약자**

  '~만큼의 정도로 커지는'

- **O(n) : n만큼 커진다**

  N이 늘어날수록 선형적으로 증가

- **알고리즘이 최악일때의 경우를 판단**

  평균과 가까운 성능으로 예측하기 쉽기 때문

출처 : https://blog.chulgil.me/algorithm/

시간복잡도 순서

O(1)	해시 함수(hash function)
O(log n)	이진 탐색(binary search)
O(n)	순차 탐색(sequential search)
O(nlogn)	퀵 정렬, 병합 정렬
O(n^2)	거품 정렬, 삽입 정렬, 선택 정렬

O(1) < O(log n) < O(n) < O(nlog n) < O(n^2)

출처 : https://bit.ly/3vzWlKI

배열(array)

- 같은 자료형의 변수로 이루어진 구성요소가 모인 것

선언방식

(자료형이 int이고 길이가 n인 배열)

```
int [] a = new int [n];
int a [] = new int [n];
int a [] = {1,2,3,4,n,...};
```

index

컴퓨터는 0부터 읽으므로 배열도 0번부터

```
int a [] = {1,2,3};
a[0] = 1;
a[1] = 2;
a[2] = 3;
```

배열의 총 원소 개수 = 배열의 길이 – 1
= (a.length – 1)

clone 메서드

배열의 복제

```
public class test123 {

    public static void main(String[] args) {
        int [] a = {1,2,3,4};
        int [] b = a.clone();

        for(int i=0;i<b.length;i++) {
            System.out.println(b[i]);
        }
    }

}
```

fill()

배열 채우기

```
1 int[] intArr = new int[5];
2 Arrays.fill(intArr, 1); // 채울 배열, 채울 값
3
4 System.out.println(Arrays.toString(intArr)); // [1, 1, 1, 1, 1]
```

asList()

배열을 List로 변환

- asList로 만들어진 List는

  1) 원소 추가(add)불가 및

  2) 값 변경 시 원본 배열 값도 변경됨

- 원소 추가 및 원소 배열 유지를 위해선

  new List 사용

```java
String[] strArr1 = {"aa", "bb", "cc"};
List<String> list1 = Arrays.asList(strArr1);

list1.set(0, "hi");
list1.add("dd"); // java.lang.UnsupportedOperationException

System.out.println(Arrays.toString(strArr1)); // [hi, bb, cc]
System.out.println(list1.toString()); // [hi, bb, cc]


String[] strArr2 = {"aa", "bb", "cc"};
List<String> list2 = new ArrayList<String>(Arrays.asList(strArr2));

list2.set(0, "hi");
list2.add("dd");

System.out.println(Arrays.toString(strArr2)); // [aa, bb, cc]
System.out.println(list2.toString()); // [hi, bb, cc, dd]
```

toString(), deepToString()

배열의 element 출력

- 1차원 배열일 경우 toString 사용,
  다차원 배열일 경우 deepToString 사용

- 다차원배열에 toString 사용 시,
  에러는 나지 않지만 주소값 출력

```
1  int[] intStr1 = {1, 2, 3};
2  System.out.println(Arrays.toString(intStr1)); // [1, 2, 3]
3
4  int[][] intStr2 = {{1, 2, 3}, {4, 5, 6}};
5  System.out.println(Arrays.toString(intStr2)); // [[I@15db9742, [I@6d06d69c]
6  System.out.println(Arrays.deepToString(intStr2)); // [[1, 2, 3], [4, 5, 6]]
```

equals(), deepEquals()

배열 비교

- 1차원 배열일 경우 equals

- 다차원 배열일 경우 deepEquals

```
1  int[] intStr1 = {1, 2, 3};
2  int[] compStr1 = new int[]{1, 2, 3};
3  System.out.println(Arrays.equals(intStr1, compStr1)); // true
4
5  int[][] intStr2 = {{1, 2, 3}, {4, 5, 6}};
6  int[][] compStr2 = new int[][]{{1, 2, 3}, {4, 5, 6}};
7  System.out.println(Arrays.equals(intStr2, compStr2)); // false
8  System.out.println(Arrays.deepEquals(intStr2, compStr2)); // ture
```

copyOf(), copyOfRange()

배열 복사

- 원본 배열보다

  큰 길이의 배열로 복사할 경우

  값이 없는 값에

  1) int의 경우 0

  2) String의 경우 null로 채워짐

```
Arrays.copyOf([] original, int newLength)
original = 원본 배열, newLength = 새 배열 길이
Arrays.copyOfRange([] original, int form, int to)
original = 원본 배열, from = 시작 인덱스, to = 끝 인덱스(해당 인덱스 전까지만 포함)
```

```
int[] intArr1 = {1, 2, 3};
int[] intArr2 = Arrays.copyOf(intArr1, 2);
int[] intArr3 = Arrays.copyOf(intArr1, 5);
int[] intArr4 = Arrays.copyOfRange(intArr1, 0, 3);
int[] intArr5 = Arrays.copyOfRange(intArr1, 0, 5);

System.out.println(Arrays.toString(intArr2)); // [1, 2]
System.out.println(Arrays.toString(intArr3)); // [1, 2, 3, 0, 0]
System.out.println(Arrays.toString(intArr4)); // [1, 2, 3]
System.out.println(Arrays.toString(intArr5)); // [1, 2, 3, 0, 0]
```

sort()

배열 정렬

```java
int[] intArr = {3, 2, 5, 1, 4};
Arrays.sort(intArr); // 오름차순
System.out.println(Arrays.toString(intArr)); // [1, 2, 3, 4, 5]

Integer[] integerArr = {3, 2, 5, 1, 4};
Arrays.sort(integerArr, Collections.reverseOrder()); // 내림차순
System.out.println(Arrays.toString(integerArr)); // [5, 4, 3, 2, 1]
Arrays.sort(integerArr, Comparator.reverseOrder()); // 내림차순
System.out.println(Arrays.toString(integerArr)); // [5, 4, 3, 2, 1]

String[] strArr = {"c", "d", "b", "a", "e"};
Arrays.sort(strArr, Collections.reverseOrder()); // 내림차순
System.out.println(Arrays.toString(strArr)); // [e, d, c, b, a]
Arrays.sort(strArr, Comparator.reverseOrder()); // 내림차순
System.out.println(Arrays.toString(strArr)); // [e, d, c, b, a]
```

```java

String[] stirngArr = {"HI", "HELLO", "BYE"};
Arrays.sort(stirngArr, new Comparator<String>() {
    @Override
    public int compare(String o1, String o2) {
        return o1.length() - o2.length(); // 글자 수로 정렬
    }
});
System.out.println(Arrays.toString(stirngArr)); // [HI, BYE, HELLO]
```

binarySearch()

배열 검색

- 사용 전 배열이 정렬(sort)되어 있어야 함

```
1  int[] intArr = {3, 2, 1, 5, 4};
2  Arrays.sort(intArr); // 검색 사용 전 정렬
3  int idx = Arrays.binarySearch(intArr, 3);
4  System.out.println(idx); // 2
```

클래스(class)

필드(field)

- 객체의 상태
- 클래스에 포함된 변수(variable)를 의미
- 멤버(member)로 속성 표현

메소드(method)

- 객체의 행동
- 특정 작업을 수행하기 위한 명령문의 집합 (기능을 표현)

클래스 선언

```
class XYZ{
        int x;
        long y;
        double z;
}
```

XYZ a; // XYZ형의 클래스 형 변수 a선언
a = new XYZ(); //XYZ형의 클래스 인스턴스(실체) 생성

XYZ a = new XYZ(); //변수와 인스턴스 생성을 한꺼번에 선언

method

```
접근제어자 반환타입 메소드이름(매개변수목록) { // 선언부
    // 구현부
}
```

[접근 제어자]

: 해당 메소드에 접근할 수 있는 범위

[반환 타입(return type]

: 메소드가 모든 작업을 마치고 반환하는 데이터의 타입

[메소드 이름]

: 메소드를 호출하기 위한 이름

[매개변수 목록(parameters)]

: 메소드 호출 시에 전달되는 인수의 값을 저장할 변수

이상으로 발표를 마칩니다

그거 아시나요?

★★★★★

감사합니다!

THANK YOU!