분할정복 알고리즘

분할 정복

- 정의 : 문제를 나눌 수 없을 때까지 나누어서 각각을 풀면서 다시 합병하여 문제의 답을 얻는 알고리즘이다.

- 설계
    1) 분할: 해결할 문제를 여러 개의 작은 부분으로 나눈다.
    2) 정복: 나눈 문제를 각각 해결한다
    3) 통합: (필요하다면) 해결한 해답을 모은다.

- 문제를 제대로 분할하면 정복하기 쉽기 때문에 분할하는 방법이 중요하다.
- 재귀 알고리즘이 많이 사용된다.

분할정복 알고리즘 응용
1. 거듭제곱

1. 거듭제곱

- 기본 정의: n거듭제곱은 어떤 수를 n번 반복하여 곱한다.

$$C^n = C \times C \times \cdots \times C$$

- 시간 복잡도 : $O(n)$

Ex) C^8 $= C \times C \times C \times C \times C \times C \times C \times C$

$= (C \times C \times C \times C) \times (C \times C \times C \times C)$ $= C^4 \times C^4$

$= (C \times C) \times (C \times C) \times (C \times C) \times (C \times C)$ $= C^2 \times C^2 \times C^2 \times C^2$

$= ((C^2)^2)^2$

!! 8번 곱하는 것이 아니라 제곱을 구한 뒤 세 번 연산하면 같은 결과를 얻는다.

1. 거듭제곱 (분할 정복 이용)

➔ 시간 복잡도: $O(\log_2 n)$

$$C^n = \begin{cases} C^{n/2} \bullet C^{n/2} & n\text{은 짝수} \\ C^{(n-1)/2} \bullet C^{(n-1)/2} \bullet C & n\text{은 홀수} \end{cases}$$

- 지수가 짝수일때

-> 지수를 반을 나눠서 곱하고,

- 지수가 홀수일때

-> 지수에서 1을 빼고, 반으로 나누어 곱하고,
밑을 한번 더 곱하기

```java
import java.util.Scanner;

public class Exponentiation {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		long a = sc.nextInt(); // 밑
		long b = sc.nextInt(); // 지수

		System.out.println(pow(a, b));
		sc.close();
	}

	private static long pow(long a, long b) {
		// 지수가 0인 경우(종료 조건).
		if (b == 0) {
			return 1;
		}

		// 반으로 나눈 거듭제곱 계산.
		long res = pow(a, b / 2);

		// 지수가 짝수인 경우.
		if (b % 2 == 0) {
			return res * res;
		}
		// 지수가 홀수인 경우
		else {
			return res * res * a;
		}
	}
}
```

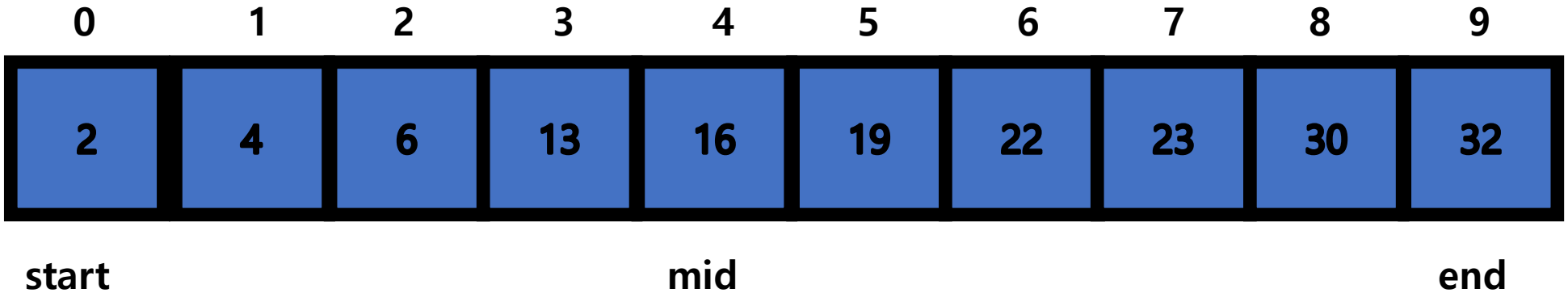
분할정복 알고리즘 응용
2. 이진탐색

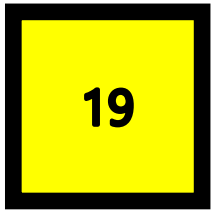
이분탐색

: **정렬되어 있는 배열** 에서 특정 데이터를 찾기 위해서 모든 데이터를 순차적으로 확인하는 대신 탐색 범위를 **절반으로 줄여** 가며 찾는 탐색 방법

- 과정
  1. 자료의 중앙에 있는 원소를 고른다
  2. 중앙 원소의 값과 목표 값을 비교
     - ➔ 일치하면 탐색 끝
     - ➔ 목표 값이 작으면 왼쪽 반에서 1~2 수행
     - ➔ 목표 값이 크면 오른쪽 반에서 1~2 수행

이분탐색 – 찾을 값이 있는 경우



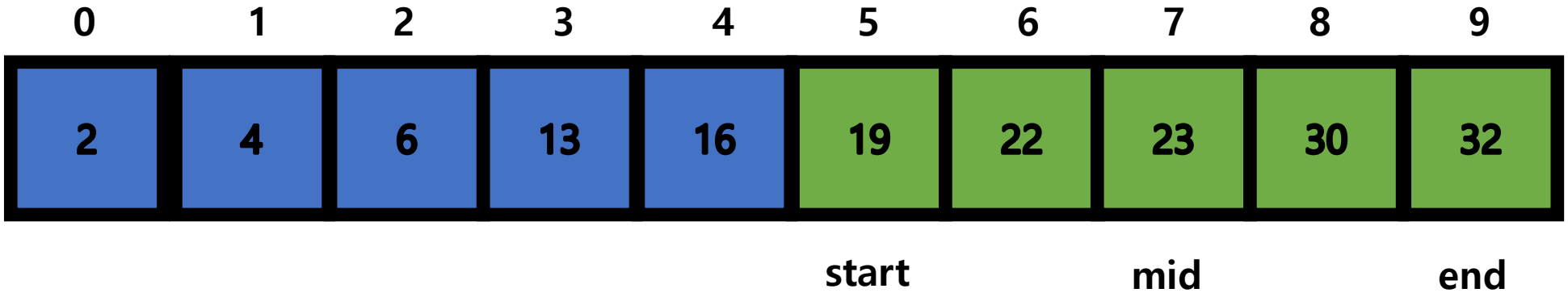


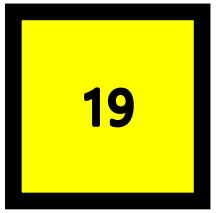
target

start = 0, end = 9, mid = 4
mid < target

-> start = mid + 1

이분탐색 – 찾을 값이 있는 경우



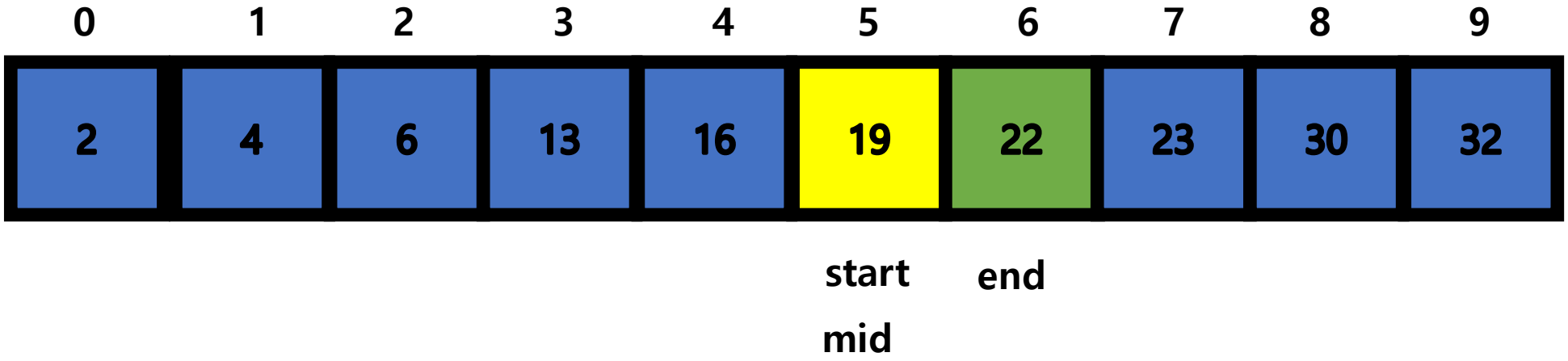


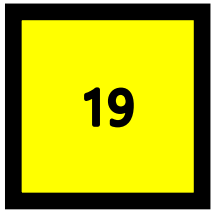
target

start = 5, end = 9, mid = 7
mid > target

-> end = mid - 1

이분탐색 – 찾을 값이 있는 경우



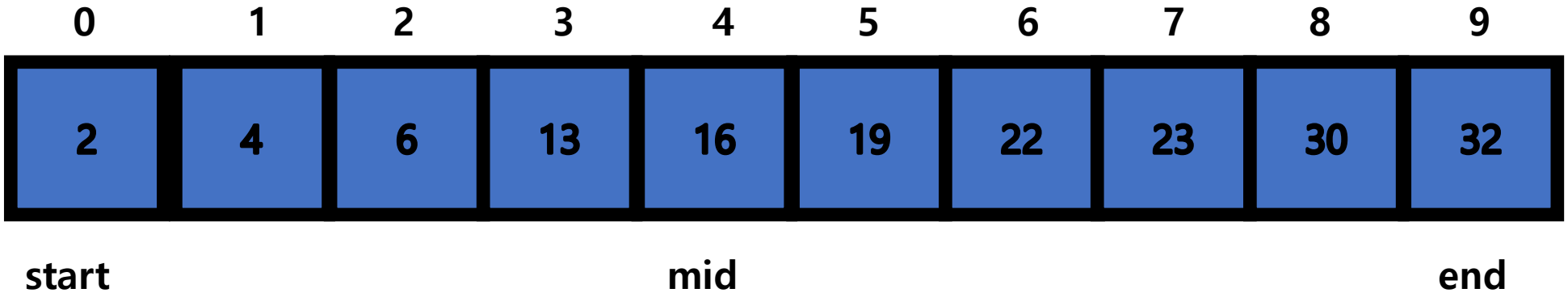


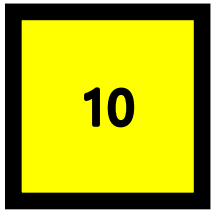
target

start = 5, end = 6, mid = 5

mid = target

이분탐색 – 찾을 값이 없는 경우



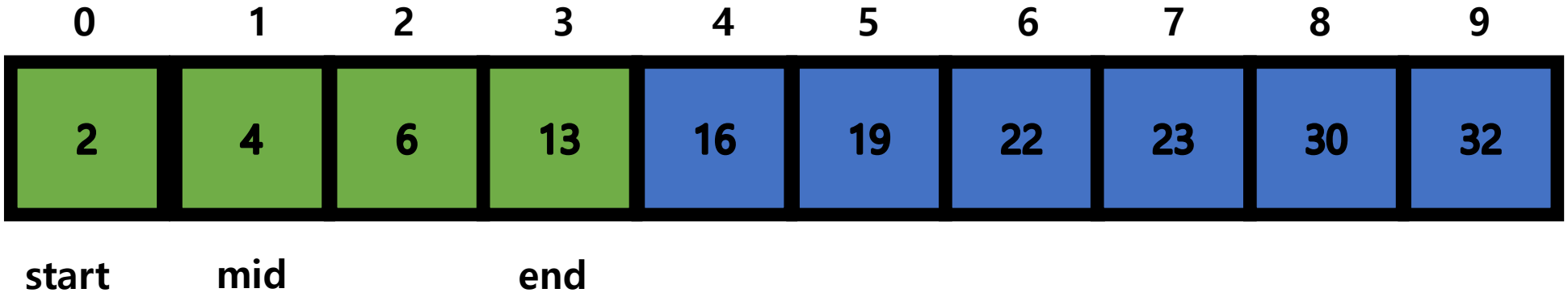


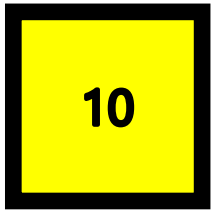
target

start = 0, end = 9, mid = 4
mid > target

-> end = mid - 1

이분탐색 – 찾을 값이 없는 경우

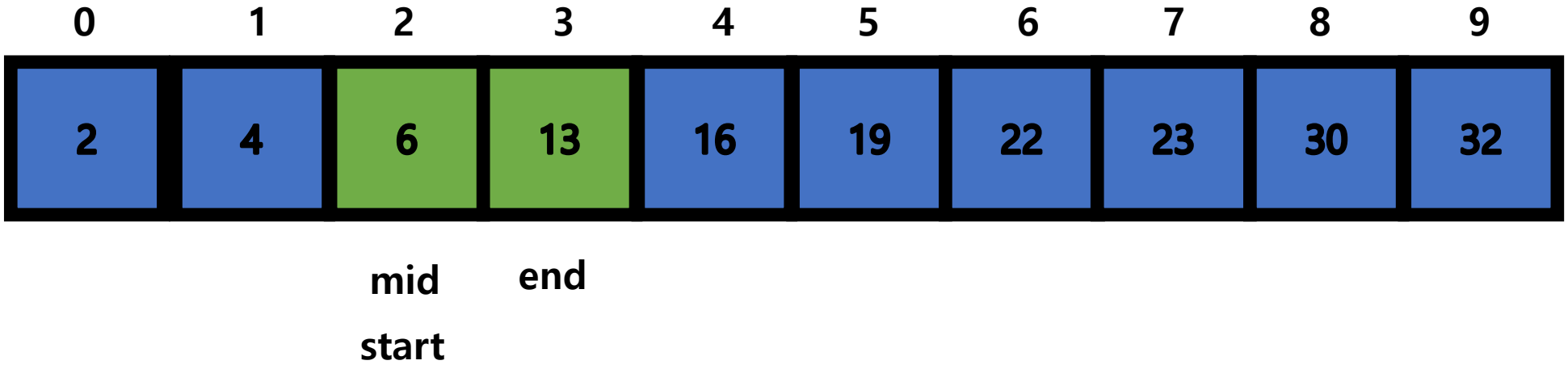


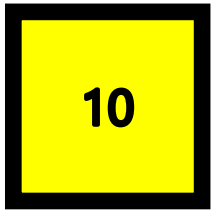


start = 0, end = 3, mid = 1
mid < target

-> start = mid + 1

이분탐색 – 찾을 값이 없는 경우



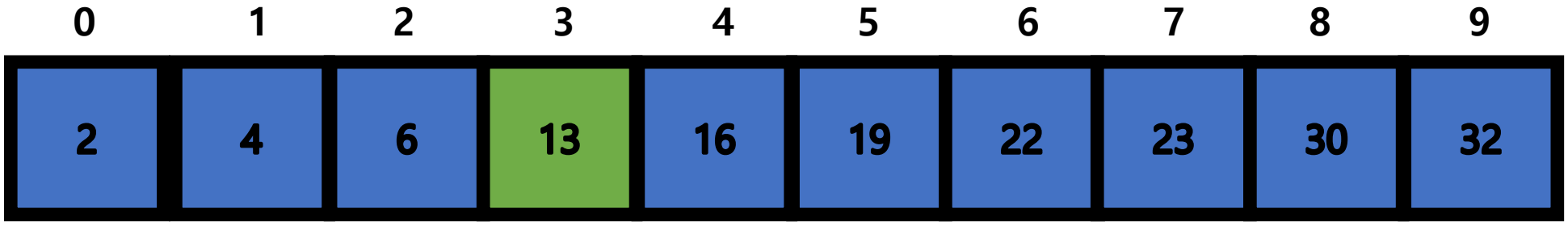


target

start = 2, end = 3, mid = 2
mid < target

-> start = mid + 1 = 3

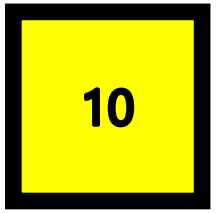
이분탐색 – 찾을 값이 없는 경우



end

start

mid

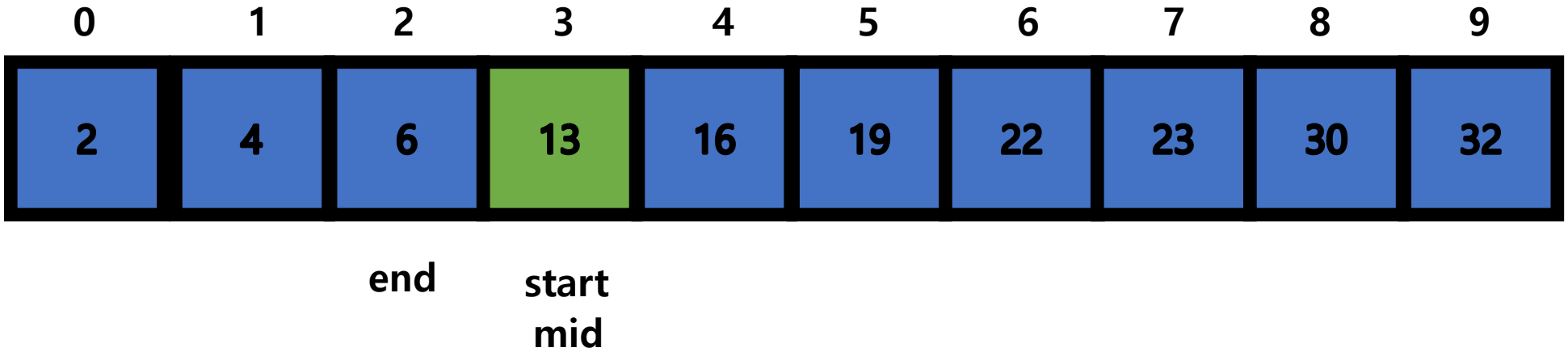


target

start = 3, end = 3, mid = 3
mid > target

-> end = mid - 1

이분탐색 – 찾을 값이 없는 경우



start > end : start 가 end보다 더 커짐 -> 코드 작성 시 조건이 된다.

이분탐색

1. start, end, mid 변수
2. 탐색할 배열, 찾을 목표 값(target)

- - 반복구조, 재귀 구조로 작성.

```java
public static int binarySearch(int[] arr, int target ) {

    int start = 0; // 탐색 범위의 왼쪽 끝 인덱스
    int end = arr.length - 1; // 탐색 범위의 오른쪽 끝 인덱스

    // start가 end보다 커지기 전까지 반복한다.
    while (start <= end) {

        int mid = (start + end) / 2; // 중간위치를 구한다.

        // target이 중간 위치의 값보다 작을 경우
        if (target < arr[mid]) {
            end = mid - 1;
        }
        // target이 중간 위치의 값보다 클 경우
        else if (target > arr[mid]) {
            start = mid + 1;
        }
        // target과 중간 위치의 값이 같을 경우
        else {
            return mid;
        }
    }

    // 찾고자 하는 값이 존재하지 않을 경우
    return -1;

}
```