07. Binary Seach & Divide and Conquer

이화여자대학교 황채원

이분 탐색이란

정렬된 데이터에서 원하는 값을 찾아내는 알고리즘

대상 데이터의 중앙값과 찾고자 하는 값(target)을 비교하여

데이터의 크기를 절반씩 줄여나가는 방식입니다

이분 탐색이란

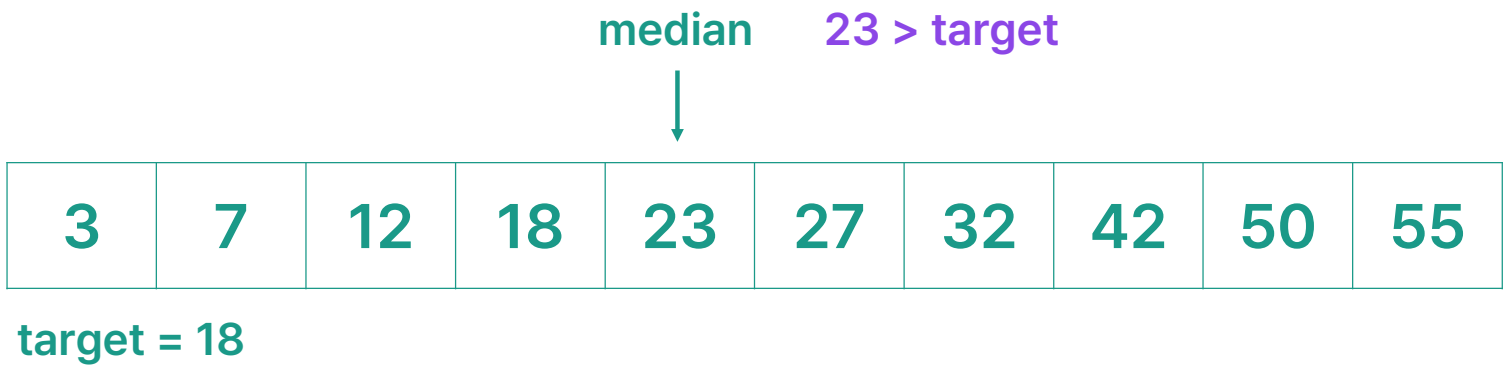
이분탐색의 과정(오름차순 기준)

1. 현재 데이터의 중앙값을 선택한다
2. 중앙값 > target : 왼쪽 절반 배열 선택
3. 중앙값 < target : 오른쪽 절반 배열 선택
4. 위의 과정을 반복하다가 중앙값==target이면 탐색 종료

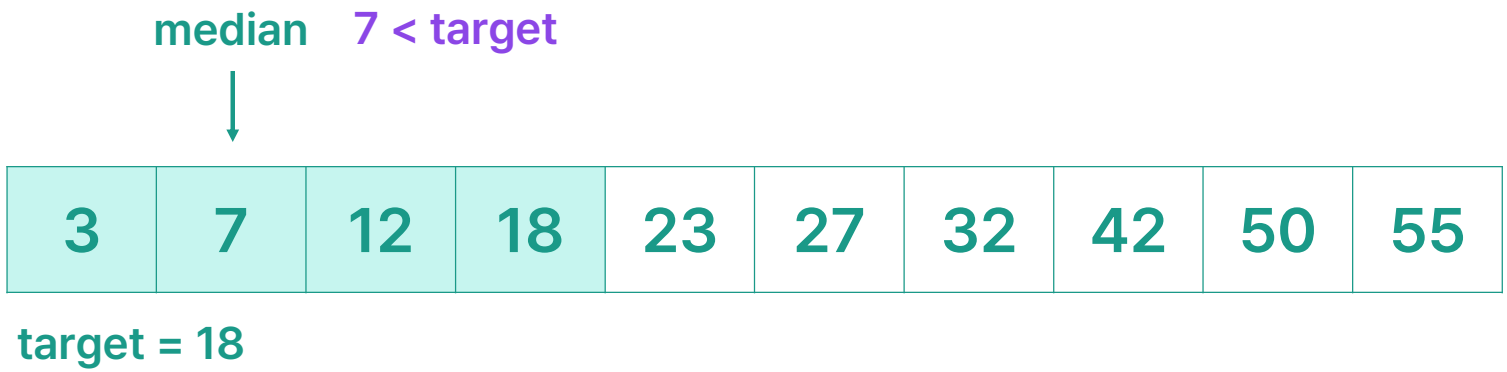
이분 탐색이란

target = 18

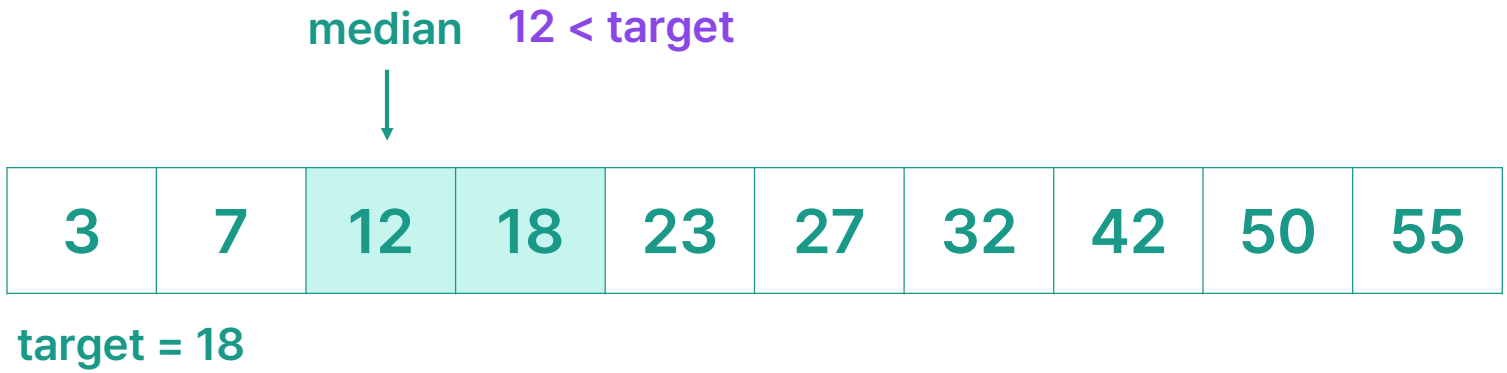
이분 탐색이란



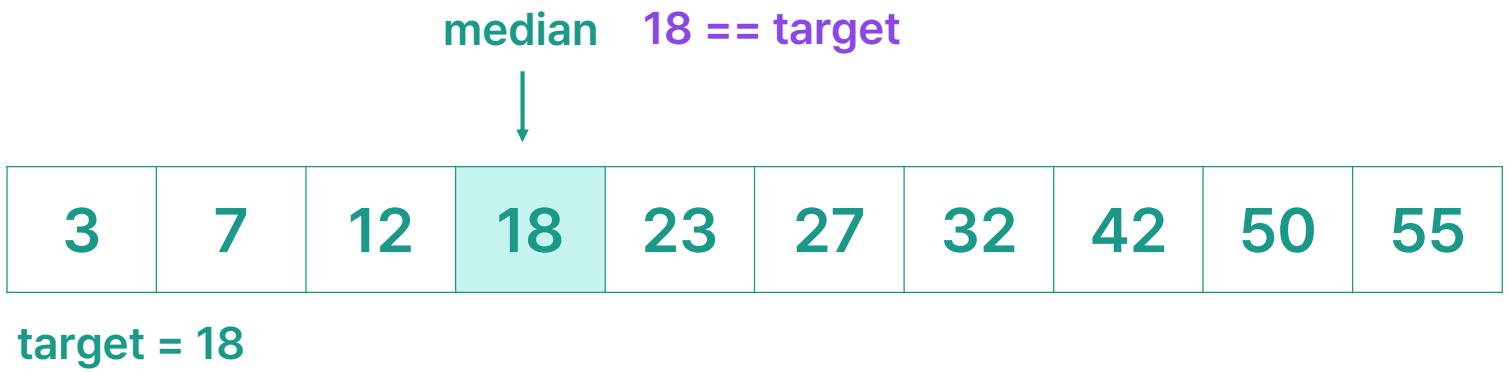
이분 탐색이란



이분 탐색이란



이분 탐색이란



이분 탐색이란

이분탐색의 특징

1. 배열의 크기가 매 반복마다 절반씩 줄어듭니다
2. 반복문, 재귀로 구현 가능합니다
3. 시간복잡도 O(logn)
4. 이분탐색을 적용하기 위해서는 **배열이 반드시 정렬**되어 있어야 합니다

반복적 이분탐색

```python
def binary_search(array, target):
    start = 0
    end = len(array) - 1  # 배열의 길이를 사용하여 end 초기화

    while start <= end:
        mid = (start + end) // 2  # 정수 나눗셈으로 중앙 인덱스 계산
        if array[mid] == target:
            return mid  # 타겟을 찾았을 경우 해당 인덱스 반환
        elif array[mid] < target:
            start = mid + 1  # 타겟이 중앙 값보다 크면, 시작점을 조정
        else:
            end = mid - 1  # 타겟이 중앙 값보다 작으면, 끝점을 조정

    return -1  # 타겟을 찾지 못했을 경우 -1 반환
```

재귀적 이분탐색

```python
def binary_search_recursive(array, target, start, end):
    if start > end:
        return -1  # Target not found
    mid = (start + end) // 2  # 정수 나눗셈으로 중앙 인덱스 계산
    if array[mid] == target:
        return mid  # Found the target
    elif array[mid] < target:
        return binary_search_recursive(array, target, mid + 1, end)
    else:
        return binary_search_recursive(array, target, start, mid - 1)
```

BOJ 1920 | 수 찾기 | Silver 4

수 찾기 성공

☆

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	128 MB	247404	76283	50640	29.965%

문제

N개의 정수 A[1], A[2], ..., A[N]이 주어져 있을 때, 이 안에 X라는 정수가 존재하는지 알아내는 프로그램을 작성하시오.

입력

첫째 줄에 자연수 N(1 ≤ N ≤ 100,000)이 주어진다. 다음 줄에는 N개의 정수 A[1], A[2], ..., A[N]이 주어진다. 다음 줄에는 M(1 ≤ M ≤ 100,000)이 주어진다. 다음 줄에는 M개의 수들이 주어지는데, 이 수들이 A안에 존재하는지 알아내면 된다. 모든 정수의 범위는 -2^{31} 보다 크거나 같고 2^{31}보다 작다.

출력

M개의 줄에 답을 출력한다. 존재하면 1을, 존재하지 않으면 0을 출력한다.

BOJ 1920 | 수 찾기 | Silver 4

입력:

- N: 정수 배열 A의 크기 (1 ≤ N ≤ 100,000).
- A[1], A[2], ..., A[N]: 배열 A에 속한 N개의 정수.
- M: 쿼리의 수, 즉 확인해야 하는 정수의 개수 (1 ≤ M ≤ 100,000).
- 이어지는 줄에 M개의 정수가 주어지며, 각 정수가 배열 A 내에 존재하는지 확인.

조건: 모든 정수의 범위는 -2^31 이상 2^31 미만.

출력: 각 쿼리에 대해 배열 A 내에 해당 정수가 존재하면 1, 존재하지 않으면 0을 출력.

BOJ 1920 | 수 찾기 | Silver 4

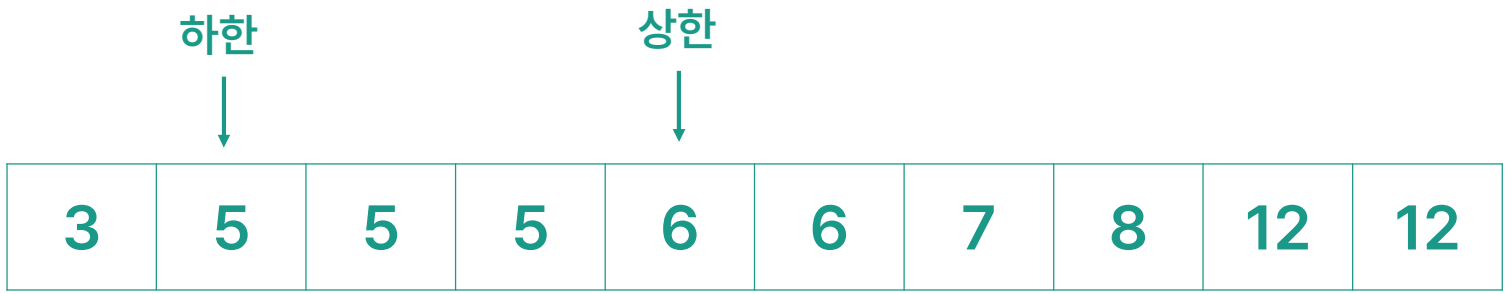
```cpp
int main() {

    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n, m;
    cin >> n;
    vector<int> array(n);
    for (int i = 0; i < n; i++) {
        cin >> array[i];
    }

    sort(array.begin(), array.end());

    cin >> m;
    for (int i = 0; i < m; i++) {
        int query;
        cin >> query;
        cout << binary_search(array, query) << '\n';
    }

    return 0;
}
```

```cpp
bool binary_search(const vector<int>& sortedArray, int target) {
    int low = 0;
    int high = sortedArray.size() - 1;

    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (sortedArray[mid] == target) {
            return true;
        } else if (sortedArray[mid] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }

    return false;
}
```

상한과 하한

하한 탐색(Lower Bound) : 주어진 값 이상이 처음 나타나는 위치를 찾습니다

상한 탐색(Upper Bound) : 주어진 값 초과가 처음 나타나는 위치를 찾습니다



BOJ 10816 | 숫자 카드 2 | Silver 4

숫자 카드 2 성공

☆

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	137535	53073	37924	37.351%

문제

숫자 카드는 정수 하나가 적혀져 있는 카드이다. 상근이는 숫자 카드 N개를 가지고 있다. 정수 M개가 주어졌을 때, 이 수가 적혀있는 숫자 카드를 상근이가 몇 개 가지고 있는지 구하는 프로그램을 작성하시오.

입력

첫째 줄에 상근이가 가지고 있는 숫자 카드의 개수 N(1 ≤ N ≤ 500,000)이 주어진다. 둘째 줄에는 숫자 카드에 적혀있는 정수가 주어진다. 숫자 카드에 적혀있는 수는 -10,000,000보다 크거나 같고, 10,000,000보다 작거나 같다.

셋째 줄에는 M(1 ≤ M ≤ 500,000)이 주어진다. 넷째 줄에는 상근이가 몇 개 가지고 있는 숫자 카드인지 구해야 할 M개의 정수가 주어지며, 이 수는 공백으로 구분되어져 있다. 이 수도 -10,000,000보다 크거나 같고, 10,000,000보다 작거나 같다.

출력

첫째 줄에 입력으로 주어진 M개의 수에 대해서, 각 수가 적힌 숫자 카드를 상근이가 몇 개 가지고 있는지를 공백으로 구분해 출력한다.

BOJ 10816 | 숫자 카드 2 | Silver 4

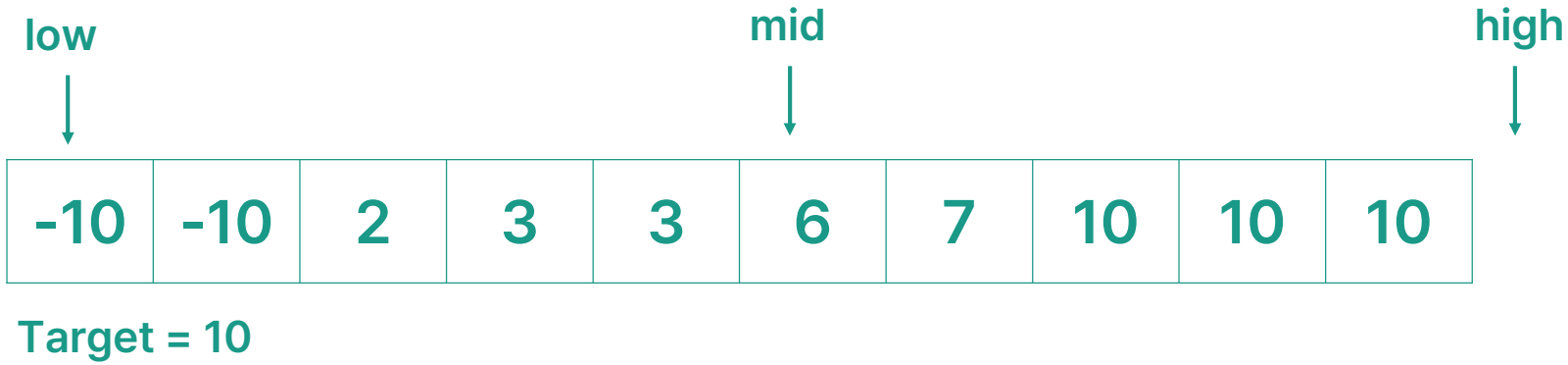
입력:

- N: 상근이가 가지고 있는 숫자 카드의 개수 (1 ≤ N ≤ 500,000).
- N개의 숫자 카드에 적힌 정수들 (범위: -10,000,000 이상 10,000,000 이하).
- M: 구해야 할 정수의 개수 (1 ≤ M ≤ 500,000).
- M개의 정수들 (범위: -10,000,000 이상 10,000,000 이하), 상근이가 몇 개 가지고 있는지 구하기

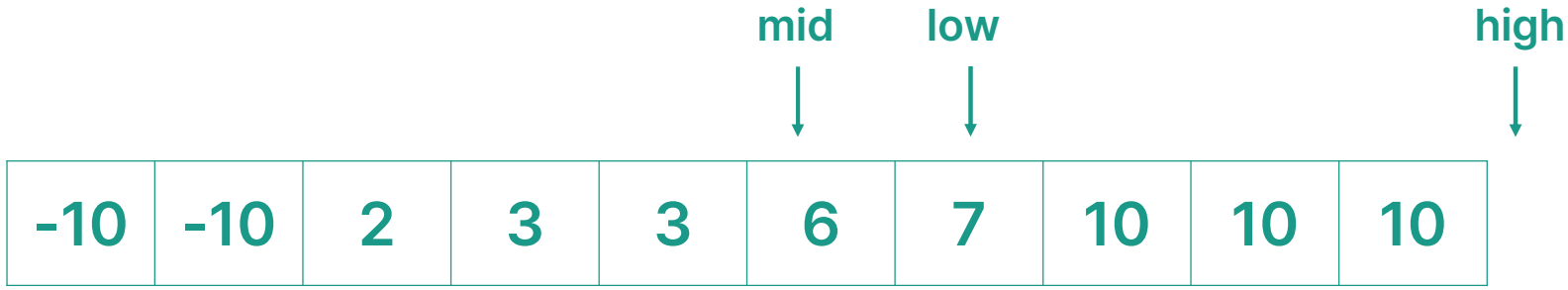
출력:

주어진 M개의 정수에 대해, 각 정수가 상근이의 숫자 카드 중 몇 개 있는지를 공백으로 구분하여 출력.

BOJ 10816 | 숫자 카드 2 | Silver 4



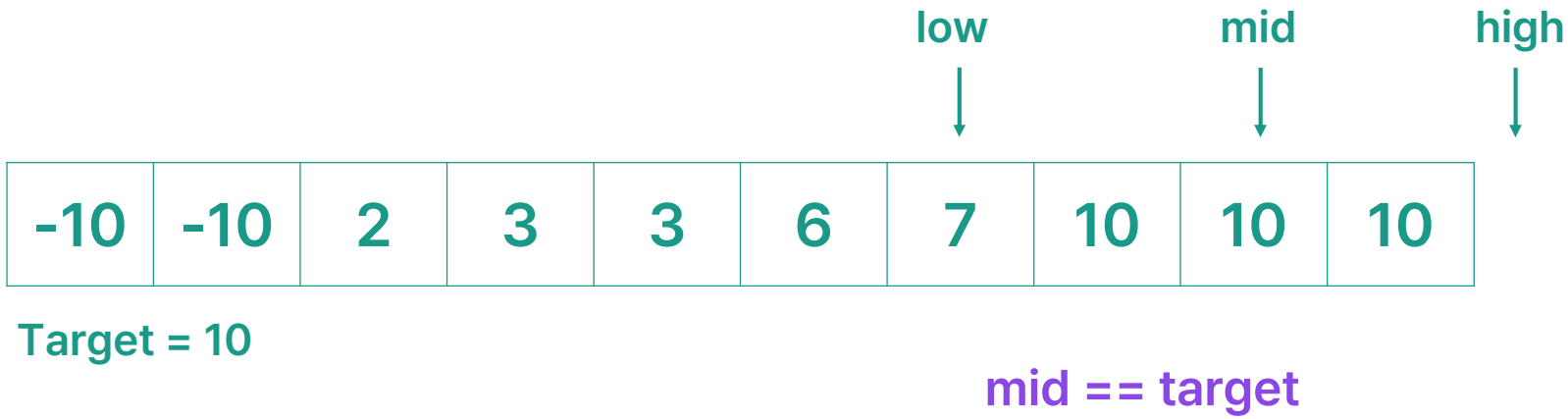
BOJ 10816 | 숫자 카드 2 | Silver 4



Target = 10

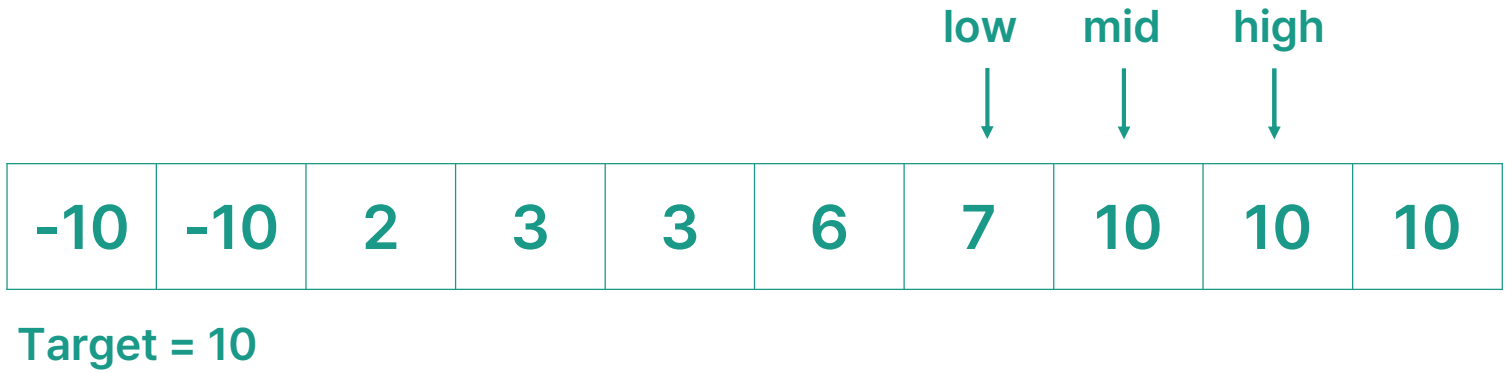
mid < target

BOJ 10816 | 숫자 카드 2 | Silver 4

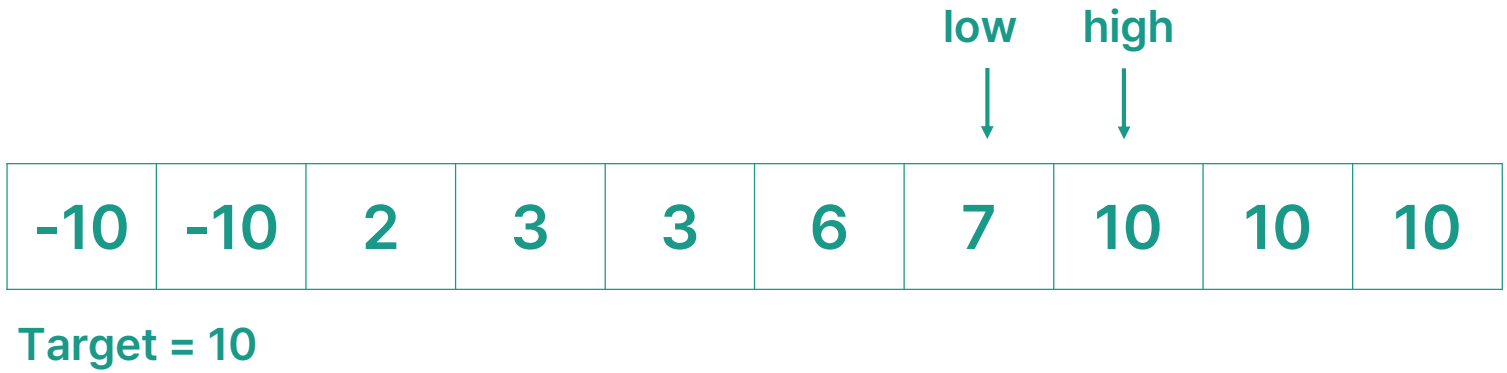


mid == target이면 종료? 왼쪽을 더 살펴봐야 하지 않을까?

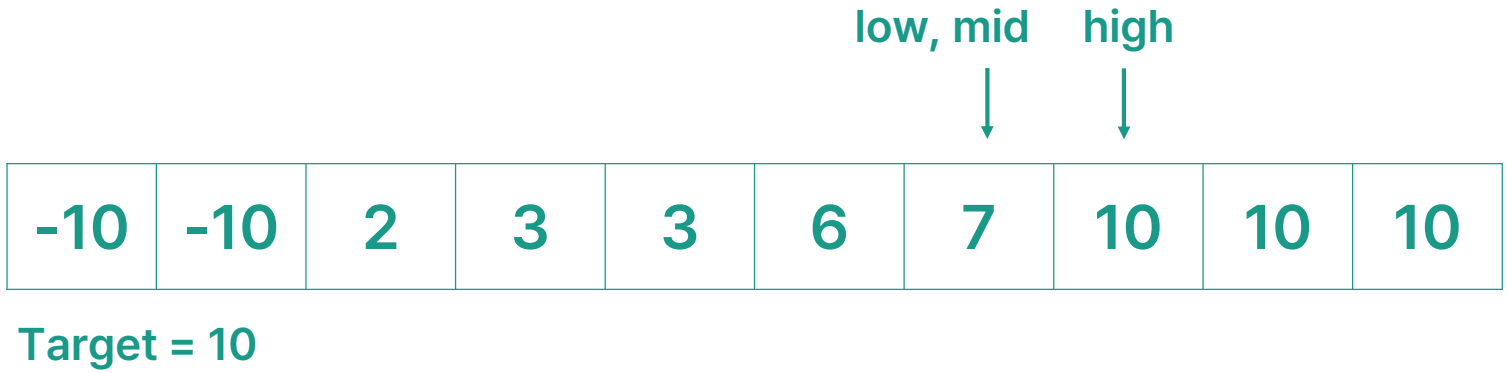
BOJ 10816 | 숫자 카드 2 | Silver 4



BOJ 10816 | 숫자 카드 2 | Silver 4



BOJ 10816 | 숫자 카드 2 | Silver 4



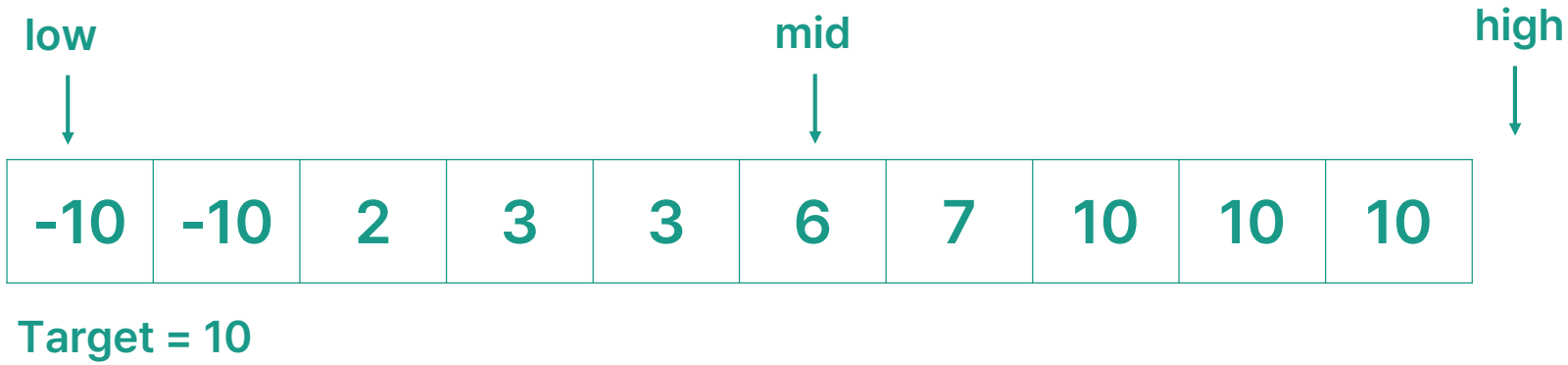
BOJ 10816 | 숫자 카드 2 | Silver 4

low = mid + 1

return low

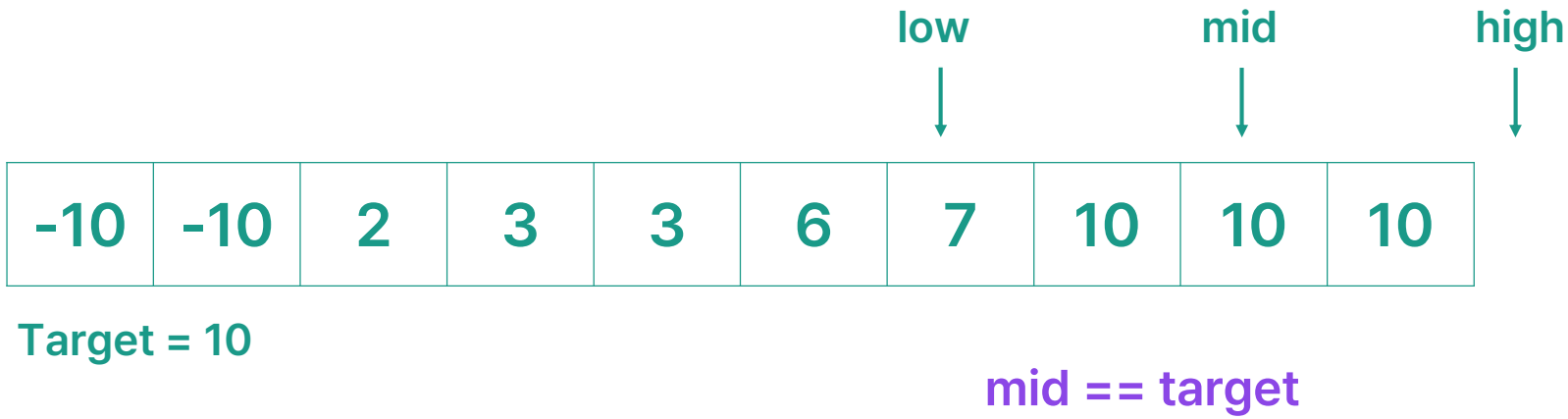
Target = 10

BOJ 10816 | 숫자 카드 2 | Silver 4



BOJ 10816 | 숫자 카드 2 | Silver 4

BOJ 10816 | 숫자 카드 2 | Silver 4



mid == target이면 종료? 이번엔 오른쪽을 더 살펴봐야합니다.

BOJ 10816 | 숫자 카드 2 | Silver 4

BOJ 10816 | 숫자 카드 2 | Silver 4

Target = 10

low==high==10

low < high 루프 조건이 만족되지 않아 탐색 종료

BOJ 10816 | 숫자 카드 2 | Silver 4


```cpp
int lower_bound(const vector<int>& array, int target) {
    int low = 0, high = array.size();
    while (low < high) {
        int mid = low + (high - low) / 2;
        if (array[mid] >= target) {
            high = mid;
        } else {
            low = mid + 1;
        }
    }
    return low;
}

int upper_bound(const vector<int>& array, int target) {
    int low = 0, high = array.size();
    while (low < high) {
        int mid = low + (high - low) / 2;
        if (array[mid] <= target) {
            low = mid + 1;
        } else {
            high = mid;
        }
    }
    return low;
}
```

```cpp
int count_cards(const vector<int>& cards, int number) {
    int lower = lower_bound(cards, number);
    int upper = upper_bound(cards, number);
    return upper - lower;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n, m;
    cin >> n;
    vector<int> cards(n);
    for (int i = 0; i < n; ++i) {
        cin >> cards[i];
    }
    sort(cards.begin(), cards.end());

    cin >> m;
    vector<int> queries(m);
    for (int i = 0; i < m; ++i) {
        cin >> queries[i];
    }

    for (int number : queries) {
        cout << count_cards(cards, number) << " ";
    }

    return 0;
}
```

파라메트릭 서치

최적화 문제(최대값/최소값 찾기)를 결정 문제(예/아니오)로 바꾸어 해결해보자!

BOJ 1654 | 랜선 자르기 | Silver 2

랜선 자르기 성공

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	128 MB	209015	49362	33386	21.300%

문제

집에서 시간을 보내던 오영식은 박성원의 부름을 받고 급히 달려왔다. 박성원이 캠프 때 쓸 N개의 랜선을 만들어야 하는데 너무 바빠서 영식이에게 도움을 청했다.

이미 오영식은 자체적으로 K개의 랜선을 가지고 있다. 그러나 K개의 랜선은 길이가 제각각이다. 박성원은 랜선을 모두 N개의 같은 길이의 랜선으로 만들고 싶었기 때문에 K개의 랜선을 잘라서 만들어야 한다. 예를 들어 300cm 짜리 랜선에서 140cm 짜리 랜선을 두 개 잘라내면 20cm는 버려야 한다. (이미 자른 랜선은 붙일 수 없다.)

편의를 위해 랜선을 자르거나 만들 때 손실되는 길이는 없다고 가정하며, 기존의 K개의 랜선으로 N개의 랜선을 만들 수 없는 경우는 없다고 가정하자. 그리고 자를 때는 항상 센티미터 단위로 정수길이만큼 자른다고 가정하자. N개보다 많이 만드는 것도 N개를 만드는 것에 포함된다. 이때 만들 수 있는 최대 랜선의 길이를 구하는 프로그램을 작성하시오.

입력

첫째 줄에는 오영식이 이미 가지고 있는 랜선의 개수 K, 그리고 필요한 랜선의 개수 N이 입력된다. K는 1이상 10,000이하의 정수이고, N은 1이상 1,000,000이하의 정수이다. 그리고 항상 K ≤ N 이다. 그 후 K줄에 걸쳐 이미 가지고 있는 각 랜선의 길이가 센티미터 단위의 정수로 입력된다. 랜선의 길이는 2^{31}-1보다 작거나 같은 자연수이다.

출력

첫째 줄에 N개를 만들 수 있는 랜선의 최대 길이를 센티미터 단위의 정수로 출력한다.

BOJ 1654 | 랜선 자르기 | Silver 2

입력:

- K: 이미 가지고 있는 랜선의 개수 (1 ≤ K ≤ 10,000)
- N: 필요한 랜선의 개수 (1 ≤ N ≤ 1,000,000), K ≤ N.
- 다음 K줄에 걸쳐, 가지고 있는 각 랜선의 길이가 정수로 입력 (길이 ≤ 2^31-1)

출력:

만들 수 있는 랜선의 최대 길이를 센티미터 단위의 정수로 출력

BOJ 1654 | 랜선 자르기 | Silver 2

가능한 최대 길이를 찾으라 (최대값 찾기)

이 길이로 N개 이상의 랜선을 만들 수 있는가? (Yes/No)

BOJ 1654 | 랜선 자르기 | Silver 2

BOJ 1654 | 랜선 자르기 | Silver 2


```cpp
bool can_make_cables(const vector<long long>& cableLengths, int target, long long length) {
    long long count = 0;
    for (long long cable : cableLengths) {
        count += cable / length;
        if (count >= target) {
            return true;
        }
    }
    return count >= target;
}
```

BOJ 1654 | 랜선 자르기 | Silver 2

```cpp
long long binary_search_max_length(const vector<long long>& cableLengths, int target) {
    long long low = 1;
    long long high = *max_element(cableLengths.begin(), cableLengths.end());
    long long maxLength = 0;

    while (low <= high) {
        long long mid = low + (high - low) / 2;
        if (can_make_cables(cableLengths, target, mid)) {
            maxLength = mid;
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }

    return maxLength;
}
```

BOJ 1654 | 랜선 자르기 | Silver 2

```cpp
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int k, n;
    cin >> k >> n;
    vector<long long> cableLengths(k);
    for (int i = 0; i < k; ++i) {
        cin >> cableLengths[i];
    }

    long long maxLength = binary_search_max_length(cableLengths, n);
    cout << maxLength << endl;

    return 0;
}
```

분할정복

상위 문제를 하위 문제로 나누어 해결하는 Top-down approach 입니다.

- 분할(Divide): 원래 문제를 작은 하위 문제로 분할합니다.
- 정복(Conquer): 분할된 작은 문제들을 재귀적으로 해결합니다.
- 결합(Combine): 정복된 문제들의 해를 결합하여 원래 문제의 해를 얻습니다.

분할정복 vs 동적 프로그래밍

분할 정복(Divide and Conquer)

중복 하위 문제: 하위 문제들은 서로 독립적입니다. 각 하위 문제는 한 번씩만 해결됩니다.

적용 예시: 병합 정렬, 퀵 정렬, 이진 탐색

동적 프로그래밍(Dynamic Programming)

중복 하위 문제: 중복되는 하위 문제들의 결과를 저장하고 재사용함으로써 계산량을 크게 줄일 수 있습니다.

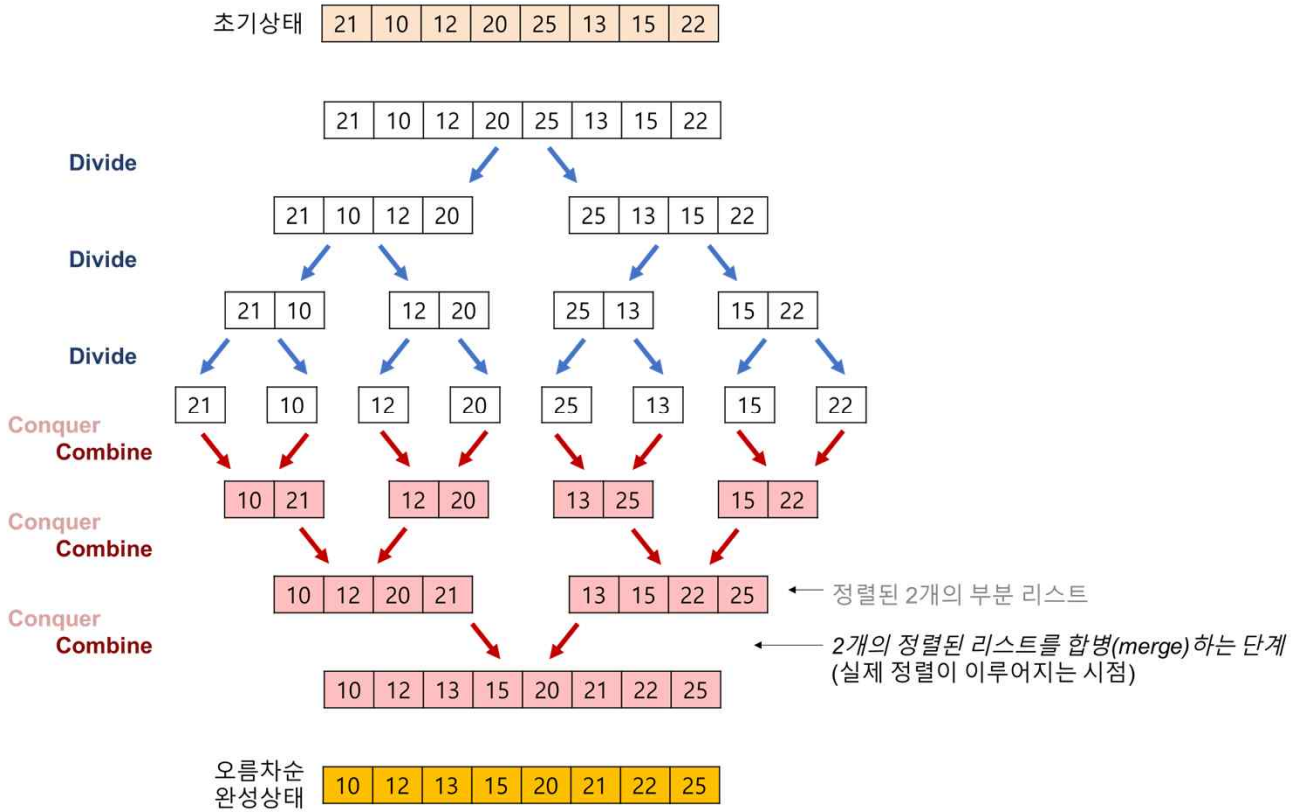
적용 예시: 피보나치 수열, 배낭 문제, 최장 공통 부분 수열(Longest Common Subsequence)

병합정렬

전체 집합을 작은 크기의 부분 집합으로 나눠서 각각을 정렬한 뒤,

정렬된 부분 집합을 합치는 정렬 알고리즘입니다.

병합정렬



https://gmlwjd9405.github.io/2018/05/08/algorithm-merge-sort.html

병합정렬

병합정렬의 특징

- 안정적인 정렬 방법: 동일한 값을 가진 원소의 상대적인 순서가 변경되지 않습니다.
- 평균적으로 O(n log n)의 시간 복잡도를 가집니다. 여기서 n은 리스트의 길이입니다.
- 상대적으로 간단한 정렬 알고리즘(예: 삽입 정렬)에 비해 구현이 복잡할 수 있습니다.
- 배열을 사용할 경우, 정렬 과정에서 추가적인 메모리 공간이 필요합니다(병합 과정에서 임시 배열이 필요).

퀵정렬

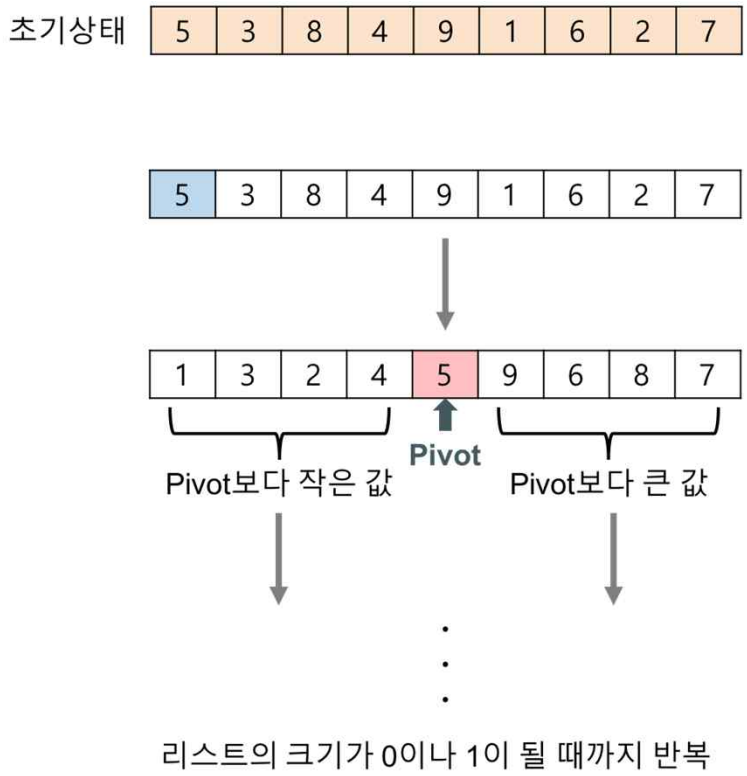
기준값(pivot)을 선정해 해당 값보다 작은 데이터와 큰 데이터로 분류하는 것을 반복하여

정렬하는 알고리즘입니다.

퀵정렬

퀵정렬의 특징

- 피벗의 선택이 시간복잡도에 많은 영향을 줍니다.
- 평균 시간 복잡도: O(nlogn), 여기서 n은 배열의 길이입니다.
- 최악 시간 복잡도: 피벗 선택을 잘못하면 O(n^2)
- 불안정 정렬: 동일한 값을 가진 요소의 상대적 순서가 정렬 과정에서 변경될 수 있습니다.



https://gmlwjd9405.github.io/2018/05/10/algorithm-quick-sort.html

BOJ 1992 | 쿼드트리 | Silver 1

쿼드트리

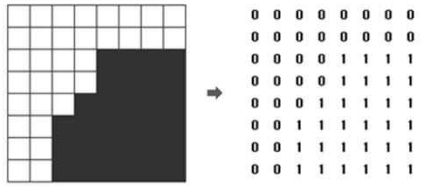
☆

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	128 MB	48107	29857	22960	62.232%

문제

흑백 영상을 압축하여 표현하는 데이터 구조로 쿼드 트리(Quad Tree)라는 방법이 있다. 흰 점을 나타내는 0과 검은 점을 나타내는 1로만 이루어진 영상(2차원 배열)에서 같은 숫자의 점들이 한 곳에 많이 몰려있으면, 쿼드 트리에서는 이를 압축하여 간단히 표현할 수 있다.

주어진 영상이 모두 0으로만 되어 있으면 압축 결과는 "0"이 되고, 모두 1로만 되어 있으면 압축 결과는 "1"이 된다. 만약 0과 1이 섞여 있으면 전체를 한 번에 나타내지를 못하고, 왼쪽 위, 오른쪽 위, 왼쪽 아래, 오른쪽 아래, 이렇게 4개의 영상으로 나누어 압축하게 되며, 이 4개의 영역을 압축한 결과를 차례대로 괄호 안에 묶어서 표현한다



위 그림에서 왼쪽의 영상은 오른쪽의 배열과 같이 숫자로 주어지며, 이 영상을 쿼드 트리 구조를 이용하여 압축하면 "`(0(0011)(0(0111)01)1)`"로 표현된다. N ×N 크기의 영상이 주어질 때, 이 영상을 압축한 결과를 출력하는 프로그램을 작성하시오.

BOJ 1992 | 쿼드트리 | Silver 1

입력

첫째 줄에는 영상의 크기를 나타내는 숫자 N 이 주어진다. N 은 언제나 2의 제곱수로 주어지며, 1 ≤ N ≤ 64의 범위를 가진다. 두 번째 줄부터는 길이 N의 문자열이 N개 들어온다. 각 문자열은 0 또는 1의 숫자로 이루어져 있으며, 영상의 각 점들을 나타낸다.

출력

영상을 압축한 결과를 출력한다.

예제 입력 1 복사

```
8
11110000
11110000
00011100
00011100
11110000
11110000
11110011
11110011
```

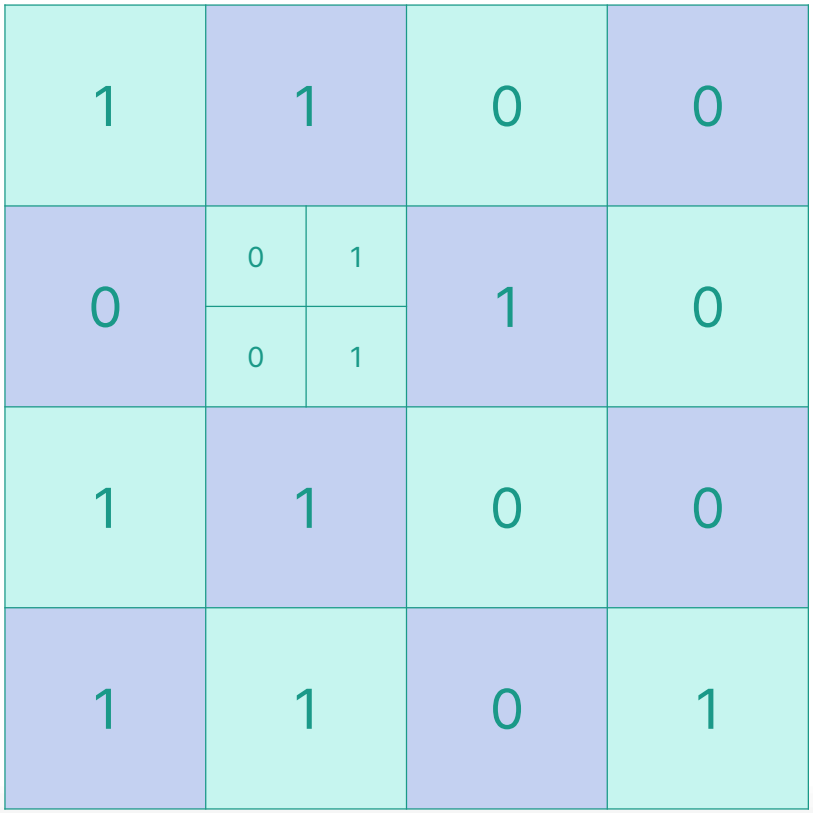
예제 출력 1 복사

```
((110(0101))(0010)1(0001))
```

BOJ 1992 | 쿼드트리 | Silver 1

1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
0	0	0	1	1	1	0	0
0	0	0	1	1	1	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	1	1
1	1	1	1	0	0	1	1

BOJ 1992 | 쿼드트리 | Silver 1



BOJ 1992 | 쿼드트리 | Silver 1


```cpp
#include <iostream>
#include <vector>
using namespace std;

vector<vector<char>> imageMatrix;

string quadTree(int y, int x, int size) {
    char init = imageMatrix[y][x];
    for (int i = y; i < y + size; ++i) {
        for (int j = x; j < x + size; ++j) {
            if (imageMatrix[i][j] != init) {
                int newSize = size / 2;
                return "("
                    + quadTree(y, x, newSize)
                    + quadTree(y, x + newSize, newSize)
                    + quadTree(y + newSize, x, newSize)
                    + quadTree(y + newSize, x + newSize, newSize)
                    + ")";
            }
        }
    }
    return string(1, init);
}
```

```cpp
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n;
    cin >> n;
    imageMatrix.resize(n, vector<char>(n));

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> imageMatrix[i][j];
        }
    }

    cout << quadTree(0, 0, n);

    return 0;
}
```

문제

필수 문제

- BOJ 2805 (나무자르기)
- BOJ 2343 (기타레슨)
- BOJ 17179 (케이크자르기)
- BOJ 1780 (종이의 개수)
- BOJ 1629 (곱셈)

심화 문제

- BOJ 2110 (공유기 설치)
- BOJ 1030 (프렉탈 평면)
- BOJ 1300 (k번째 수)
- BOJ 3079 (입국심사)