2.이진탐색, 투 포인터 누적합

23 안해성

배울 내용

- 이진탐색
  좌표 압축
- 매개변수 탐색
- 투포인터
  슬라이딩 윈도우
- 누적합

2-1. 이진 탐색

Binary search

수열에서 숫자를 찾는 알고리즘

- 길이가 **N**인 오름차순 정렬된 수열 **A**와 정수 **x**가 주어진다.
  A 수열안에 **x**가 있는지 판단하는 프로그램을 작성하라.

- 입력:
  N 이 주어지고 다음 줄에 정수 **N** 개가 주어진다.
  그리고 마지막 줄에 **x**가 주어진다

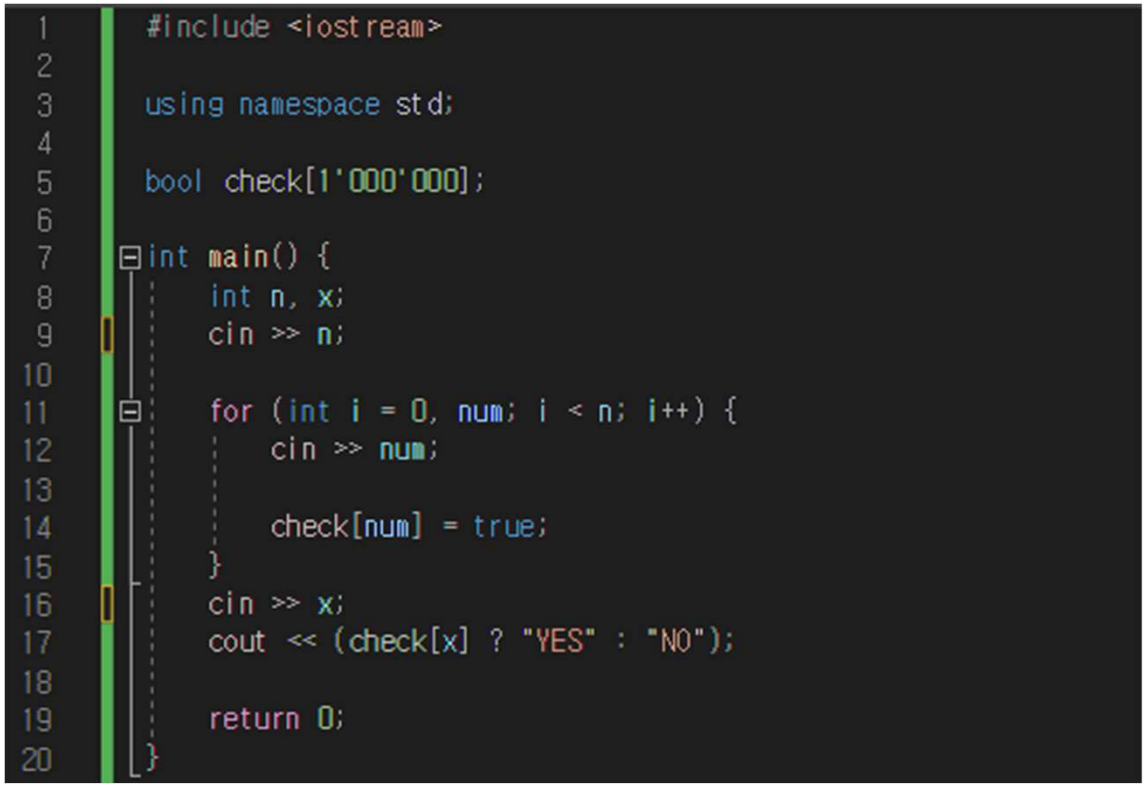
- 출력:
  x가 있으면 YES, **x**가 없으면 NO를 출력하라.

O(1)에 해결하기

- 만약 입력 받는 정수의 범위가 적당히 작다고 가정한다면,

  10이 입력으로 들어오면 배열의 10번째 인덱스에 true를 넣어주는 방법으로 해결 가능.

- 시간 복잡도 :
  check[x] -> O(1)
  따라서 O(1)



```cpp
#include <iostream>

using namespace std;

bool check[1'000'000];

int main() {
    int n, x;
    cin >> n;

    for (int i = 0, num; i < n; i++) {
        cin >> num;

        check[num] = true;
    }
    cin >> x;
    cout << (check[x] ? "YES" : "NO");

    return 0;
}
```

O(n)에 해결하기

- 입력 받은 값들을 배열에 전부 넣고,
  배열의 요소를 전부 순회하여 x 찾기.

- 시간 복잡도 :
  20번째 줄의 반복문 -> O(n)
  따라서 O(n)

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    int n, x;
    cin >> n;

    vector<int> arr;

    for (int i = 0, num; i < n; i++) {
        cin >> num;

        arr.push_back(num);
    }

    cin >> x;
    bool exist = false;
    for (const int& element : arr) {
        exist = element == x;

        if (exist) break;
    }

    cout << (exist ? "YES" : "NO");

    return 0;
}
```

O(log n)에 해결하기

- O(1)알고리즘처럼 숫자를 제한하지 않으면서, O(n) 보다 빠르게 구현 할 수 없을까?

- **“이진 탐색”**을 사용하면 가능하다!

이진 탐색

- **"정렬된"** 값들에서 O(log n)만에 원하는 값을 찾아내는 알고리즘.
- 10억개의 입력이 들어온다고 해도 30번의 연산에 값을 찾을 수 있을 정도로 빠름.

- 알고리즘 시뮬레이션 :
  다음 배열에서 6을 찾는 경우.

이진 탐색

- 중앙 값을 기준으로,

  찾는 값이 더 크면 왼쪽부터 중앙까지 배열 버리기,

  찾는 값이 더 작으면 중앙부터 오른쪽까지 배열 버리기.

- 시간 복잡도:
  binary_search() -> O($\log_2 N$)
  따라서 O(log N)

```
6   vector<int> arr;
7
8   bool binary_search(int lt, int rt, int x) {
9       int mid;
10
11      while (lt <= rt) {
12          mid = (lt + rt) / 2;
13
14          if (arr[mid] < x) {
15              lt = mid + 1;
16          }
17          else if (arr[mid] > x) {
18              rt = mid - 1;
19          }
20          else {
21              return true;
22          }
23      }
24
25      return false;
26  }
```

std::lower_bound();

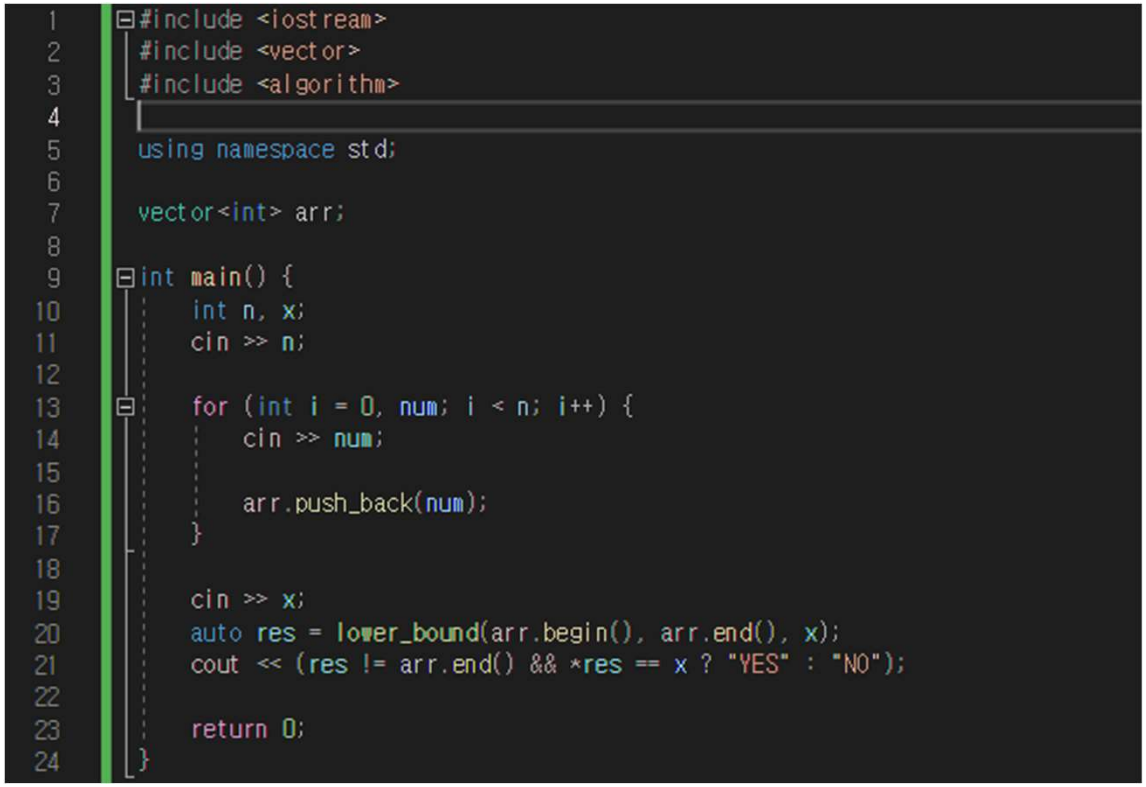
- <algorithm>에 있는 이진탐색 알고리즘.

- std::lower_bound(
  배열의 첫번째 요소 포인터,
  배열의 마지막 다음 요소 포인터,
  찾는 값
)

- 값의 포인터를 반환.
  있으면 찾는 값의 포인터,
  없으면 찾는 값보다 큰 값 중에 최소값의 포인터 반환.

std::lower_bound()

- 결과가 배열의 마지막 다음 요소의 포인터 일 수 있는 점 주의!



```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

vector<int> arr;

int main() {
    int n, x;
    cin >> n;

    for (int i = 0, num; i < n; i++) {
        cin >> num;

        arr.push_back(num);
    }

    cin >> x;
    auto res = lower_bound(arr.begin(), arr.end(), x);
    cout << (res != arr.end() && *res == x ? "YES" : "NO");

    return 0;
}
```

10815.숫자 카드

- 문제 요약:
  길이가 n인 수열이 주어지고 정수 m개가 주어지는데, 그 정수가 수열에 존재하는지 판단하는 문제.

- 풀이 :
  O(n) 방식으로 숫자 존재 판단하면, 500,000 X 500,000 > 1e9 이기 때문에 TLE.

  따라서 이진 탐색을 사용한다면?

10815 이진탐색 사용 풀이

- 이진 탐색은 정렬된 수열에서만 사용가능, 하지만 주어진 수열은 정렬되어 있지 않다.

  따라서 std::sort로 정렬 후, 각 정수를 이진 탐색으로 검색하면 된다.

- 시간 복잡도:
  std::sort() -> O(n log n)
  std::lower_bound() -> O(log n)
  따라서 O(n long n) + m*O(log n),
  대충 m이 더 크다고 한다면, O(m log n)

10815 코드

```cpp

using namespace std;

int arr[500500];

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;

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

    sort(arr, arr + n);

    int tc;
    cin >> tc;

    while (tc--) {
        int num;
        cin >> num;

        auto res = lower_bound(arr, arr + n, num);
        cout << ( res != arr + n && *res == num )<< ' ';
    }

    return 0;
}
```

2-2.매개 변수 탐색

Parametric search

매개 변수 탐색

- 이진 탐색 활용한 알고리즘. 이진 탐색이랑 똑같음.

- 배열에서 탐색을 진행하는 것이 아닌, 어떤 구간에서 이진탐색을 진행.

  문제에서 구간이 **[minLimit, maxLimit]**이고
  어떤 값 **x**가 참이라고 가정했을 때,
  [minLimit, x]에서 전부 참이고, **(x, maxLimit]**에서 거짓인
  문제에서 쓸 수 있음.

매개변수 탐색 의사코드

- 어떤 구간에서 조건을 만족하는 최대값 찾기

- valid(int x) :
  x가 참이면 return true, 아니면 return false.

```
int main() {
    int lt = minLimit, rt = maxLimit;
    int mid, ans;

    while (lt <= rt) {
        mid = (lt + rt) / 2;

        if (valid(mid)) {
            ans = mid;
            lt = mid + 1;
        }
        else {
            rt = mid - 1;
        }
    }

    return 0;
}
```

1654.랜선 자르기

- 문제 요약
  길이가 다양한 랜선 K개가 있음. 랜선들을 잘라서 어떤 길이 a의 랜선 N개 이상을 만들어야 하는 문제.

- 풀이:
  랜선 길이를 1로 만들 때, 2로 만들 때... 이런 방식으로 탐색하면 2^32 – 1까지 탐색해야 하므로 TLE

  매개변수 탐색이 가능할까?

1654 매개변수 탐색 확인하기

- 모든 랜선을 x 길이씩 자른다고 가정하자.
  그러면 각 랜선의 길이를 A_i라고 할 때,
  랜선의 개수 res는 $\sum_{i=1}^{k} \lfloor A_i / x \rfloor$가 된다.

- x와 res의 관계는 반비례라는 것을 알 수 있다.

  따라서 어떤 x의 res가 n 이상이라면 [minLimit, x]에서 x의 res는 전부 n이상이다.

- 시간 복잡도
  O(log (2^31 -1) * k)

1654 코드

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

vector<int> lines;

bool valid(int len, int cnt) {
    int res = 0;
    for (const int line : lines) {
        res += line / len;
    }

    return res >= cnt;
```

```cpp
int main() {
    int n, m;

    cin >> n >> m;

    for (int i = 0, len; i < n; i++) {
        cin >> len;
        lines.push_back(len);
    }

    unsigned int lt = 1, rt = (1 << 31) - 1;
    int mid, ans = 0;

    while (lt <= rt) {
        mid = (lt + rt) / 2;

        if (valid(mid, m)) {
            ans = mid;
            lt = mid + 1;
        }
        else {
            rt = mid - 1;
        }
    }

    cout << ans;

    return 0;
```

+ 좌표 압축

- 좌표 압축이란?
  길이가 N인 임의의 정수의 수열이 주어졌을 때, 각 수를 크기 순서에 맞게 0부터 N-1까지 번호를 매겨주는 기법.

- E.g.)
  23 7983 -238 3049 13 43 ->
  2 5 0 4 1 3

- 유용하지만 이진 탐색과 정렬로 쉽게 할 수 있다.

좌표 압축 코드

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;


int main() {
    int n;
    cin >> n;

    vector<int> arr, temp;
    for (int i = 0, num; i < n; i++) {
        cin >> num;
        arr.push_back(num);
        temp.push_back(num);
    }

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

    vector<int> compress;
    for (int i = 0; i < n; i++) {
        compress.push_back(lower_bound(temp.begin(), temp.end(), arr[i]) - temp.begin());
    }

    return 0;
}

```

2-3. 투 포인터

Two pointer

합이 x가 되게 하는 두 요소

- 길이가 N인 수열 A와 정수 x가 주어진다. 수열 A에서 합이 x가 되도록 하는 요소 2개를 찾는 프로그램을 작성하라.

- 입력 :
  N과 x가 주어지고 다음줄에 A의 요소 N개가 주어진다.

- 출력 :
  합이 x가 되는 A의 요소 2개를 출력하라. 그러한 요소가 없으면 NONE을 출력하라

O(n^2) 풀이

- For문 2개 써서 모든 경우의 수를 확인하면 된다.

  20줄의 for문 -> O((n-1)*(n-2)/2)
  따라서 O(n^2)

- 훨씬 빠르게 할 수는 없을까?

```cpp
int main() {
    int n, x;
    cin >> n >> x;

    vector<int> arr;
    for (int i = 0, num; i < n; i++) {
        cin >> num;
        arr.push_back(num);
    }

    int i, k;
    bool res = true;
    for (i = 0; res && i < n; i++) {
        for (k = i + 1; k < n; k++) {
            if (arr[i] + arr[k] == x) {
                res = false;
                break;
            }
        }
    }

    if (res) {
        cout << "NONE";
    }
    else {
        cout << arr[i-1] << ' ' << arr[k];
    }
    return 0;
}
```

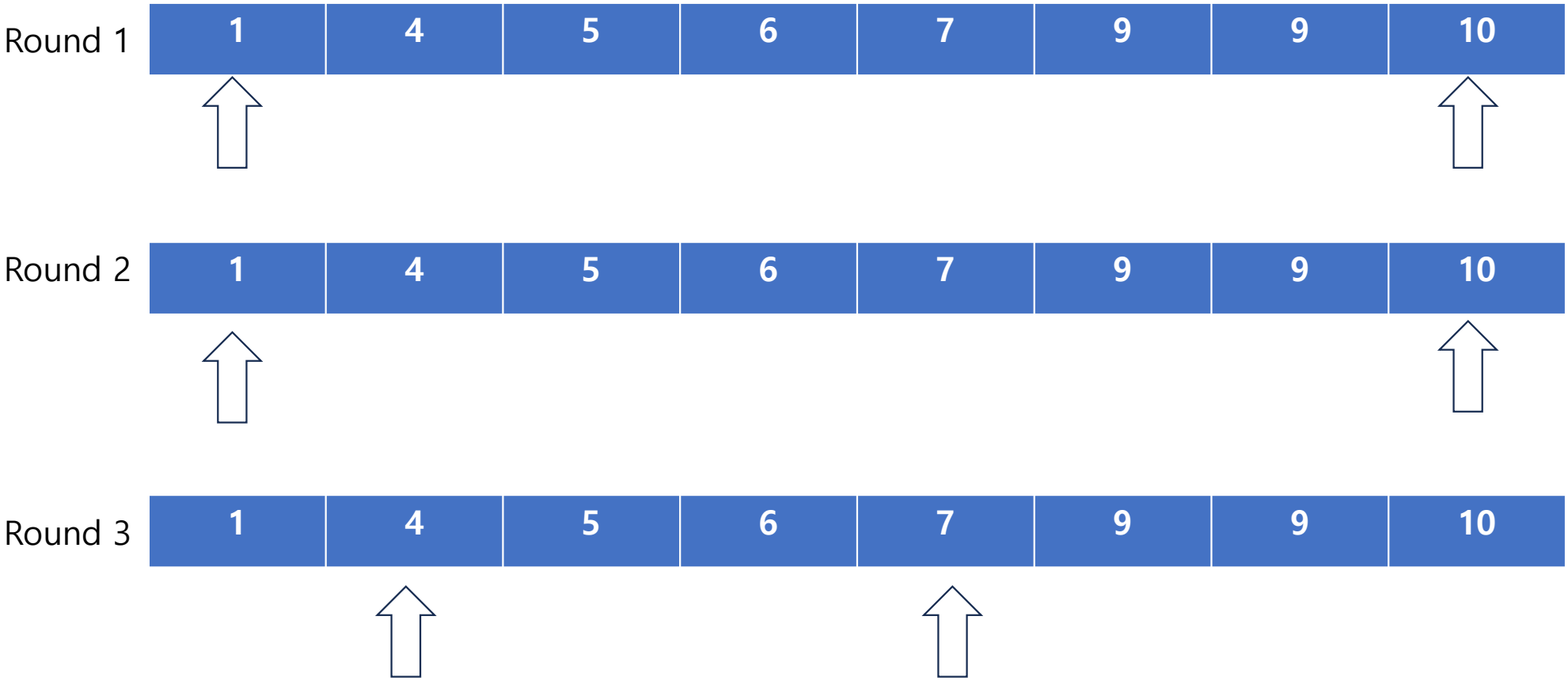
O(n log n) 풀이

- 가장 먼저, 수열 A를 정렬한다. 그리고 정렬된 수열의 맨 처음, 맨 마지막을 가리키는 포인터를 2개 만든다.

  왼쪽 포인터는 한 라운드가 지날 때마다 오른쪽으로,
  오른쪽 포인터는 두 포인터가 가리키는 요소의 합이 x보다 작아질 때까지 왼쪽으로 이동한다.

- 증명은 직관적으로 가능하다.

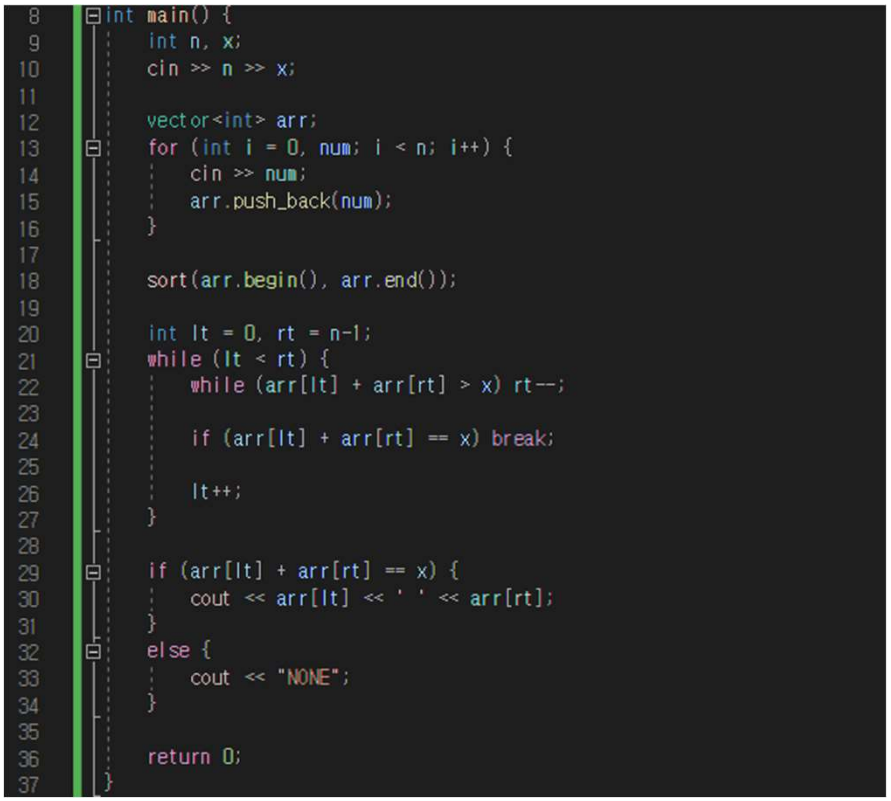
O(n log n) 풀이 시뮬레이션



O(n log n) 풀이 코드

- 시간 복잡도 :
  std::sort -> O(n log n)
  포인터 움직이기 -> O(n)
  따라서 O(n) + O(n log n) -> O(n log n)

- 다른 방식으로도 O(n log n) 가능할까?



```cpp
int main() {
    int n, x;
    cin >> n >> x;

    vector<int> arr;
    for (int i = 0, num; i < n; i++) {
        cin >> num;
        arr.push_back(num);
    }

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

    int lt = 0, rt = n-1;
    while (lt < rt) {
        while (arr[lt] + arr[rt] > x) rt--;

        if (arr[lt] + arr[rt] == x) break;

        lt++;
    }

    if (arr[lt] + arr[rt] == x) {
        cout << arr[lt] << ' ' << arr[rt];
    }
    else {
        cout << "NONE";
    }

    return 0;
}
```

투 포인터

- 방금 문제와 같이 포인터 2개를 적절히 움직여서 문제를 푸는 기법.

  하나로 정해진 알고리즘이 아니라, 여러가지로 사용할 수 있는 기법이다.

- 3273.두 수의 합 : 방금 푼 문제 업그레이드 버전
  2018.수들의 합 5 : 포인터를 다른 형식으로 처리
  10025.게으른 백곰 : 슬라이딩 윈도우

2-4.누적 합

Prefix sum

구간 합을 여러 개 구하는 문제

- 길이가 N인 수열 A이 주어지고 t개의 쿼리가 들어온다.
  쿼리의 종류는 하나로 정수 a, b (a<=b)가 주어졌을 때, 수열 A[a]~A[b]까지의 합을 구하는 쿼리다.

- 입력 :
  N과 다음 줄에 수열 A의 요소 N개가 주어진다. 그 다음 줄에 t가 주어지고 다음 t개의 줄에는 쿼리 a, b가 주어진다.

- 출력 :
  각 쿼리의 답을 출력하라

O(Nt) 풀이

- 쿼리 하나하나 직접 반복문을 돌리면서 구하는 방법이 있다.

- 시간 복잡도 :
  t개의 쿼리마다 최악의 경우인 [0, n-1] 쿼리가 주어진다면,
  O(f(t))는 O(Nt)가 된다.

```cpp
int main() {
    int n;
    vector<int> arr;

    cin >> n;

    for (int i = 0, num; i < n; i++) {
        cin >> num;

        arr.push_back(num);
    }

    int tc;
    cin >> tc;

    while (tc--) {
        int a, b;
        cin >> a >> b;

        int sum = 0;
        for (int i = a; i <= b; i++) {
            sum += arr[i];
        }

        cout << sum << '₩n';
    }

    return 0;
}
```

미리 구간의 합을 구하면 어떨까?

- 문제의 t가 아무리 커진다고 하더라도, 구간 합을 O(1)만에 해결 가능.

- 실제로 모든 경우의 구간 합을 구하면 매우 큰 시간과, 메모리가 필요하다.

  그렇다면 어떻게 모든 구간의 합을 구할 수 있을까?

누적 합

- 길이가 N인 어떤 배열 arr이 있다.
  그 배열의 원소는 arr[i] = $\sum_{k=0}^{i} arr[k]$ 인 특징이 있다.

- 이런 경우에 0, b인 쿼리는 쉽게 처리 할 수 있다.

  a, b인 쿼리는 어떻게 처리 할 수 있을까?

O(N)+O(t) 풀이

- 가장 먼저 입력된 배열의 누적합을 구하고 쿼리를 누적 합을 이용해 O(1)만에 해결한다.

- 시간 복잡도 :
  전처리 시간 -> O(n)
  쿼리 처리 시간 -> t*O(1)
  따라서 O(N) + O(t)

```cpp
int main() {
    int n;
    vector<int> arr;

    cin >> n;

    arr.push_back(0);
    for (int i = 0, num; i < n; i++) {
        cin >> num;

        arr.push_back(num);
    }

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

    int tc;
    cin >> tc;

    while (tc--) {
        int a, b;
        cin >> a >> b;

        a++, b++;

        cout << arr[b] - arr[a-1] << '\n';
    }

    return 0;
}
```

과제

쉬움 :
2470(투포인터, 이진 탐색)
27931(정렬, 투포인터)
11660(2차원 누적합)
10025.게으른 백곰
어려움 :
27942(2차원 누적합 활용)