

25-2 이니로 알고리즘 멘토링

멘토 - 김수성

멘토링 진행 방식

기본 과제 풀이 C++, Python

알고리즘 설명

예제 문제 풀이 및 코드 설명

X요일 21 ~ 23시

과제

과제 5문제

다음 멘토링 시간까지 풀어오기

사용하고 싶은 언어 사용 (Java, C ...)

과제 제출

과제

📄 과제

과제

☰ 리스트 보기 +

📄 예시

📄 1주차

김수성

```
#include <iostream>
#include <stack>
using namespace std;

stack<int> st;
int main() {
    int n; cin >> n;
    for(int i = 1; i <= n; i++){
        string s; cin >> s;
        if(s == "push"){
            int v; cin >> v;
            st.push(v);
        }
        else if(s == "pop"){
            if(st.empty()) cout << -1 << "\n";
            else{
                st.pop();
                cout << st.top() << "\n";
            }
        }
    }
}
```


스택 10828

<https://www.acmicpc.net/problem/10828>

📄 김수성

과제 제출

문제 ▾ 문제집 대회 2 채점 현황 랭킹 게시판 그룹 더 보기 ▾ 🔍

그룹 이름	그룹장	멤버
25-1 이니로 알고리즘 멘토링	 kss418	6

메인 문제집 문제집 만들기 채점 현황 연습 연습 만들기 랭킹 게시판 글쓰기 파일 설정 관리

번호	만든 사람	제목	진행도
77987	 kss418	1주차 심화 과제	<div><div></div>2 / 2</div>
77986	 kss418	1주차 기본 과제	<div><div></div>5 / 5</div>

메인 문제집 문제집 만들기 채점 현황 연습 연습 만들기 랭킹 게시판 글쓰기 파일 설정 관리

연습 이름	시작	종료	상태	채점 현황	수정
1주차 과제	2025년 3월 17일 21:00	2025년 3월 24일 20:00	시작까지 3일 08:24:01	채점 현황	수정

주차 별 내용

1주차	3/17		6주차	4/28	
2주차	3/24		7주차	5/5	
3주차	3/31		8주차	5/12	
4주차	4/7		9주차	5/19	

풀다가 막힐 때..

디코, 카톡으로 질문하기 (과제가 아니어도 괜찮음)

풀이 검색해보고 문제 이해하기

더 좋은 강의 듣고 복습

kks227 블로그

<https://m.blog.naver.com/kks227/220769859177>

바킹독 유튜브 강의

<https://www.youtube.com/watch?v=Lc0lobH7ues&list=PLtqbFd2VIQv406D6l9HcD732hdrnYb6CY>

질문?

1주차 - 이분탐색 / 매개변수탐색

선형 탐색

정렬 되어 있는 배열 A에서 특정한 수 X를 찾는 방법

나이브하게 구현하면 배열 A를 다 돌아야 함

A의 길이가 N일 때 시간 복잡도 $O(N)$

1	4	6	11	14	16	22
0	1	2	3	4	5	6

이분 탐색

각 수에서 추가적인 정보를 얻을 수는 없을까?

배열 A는 정렬 되어 있으므로

인덱스 0 ~ 3 의 값들은 14 이하

인덱스 5 ~ 6 의 값들은 14 이상

1	4	6	11	14	16	22
0	1	2	3	4	5	6

이분 탐색

배열 A에서 16을 찾고 있다고 생각할 때
인덱스 4 이전의 값들은 항상 14 이하이므로
인덱스 5 ~ 6의 값만 고려하면 됨

1	4	6	11	14	16	22
0	1	2	3	4	5	6

이분 탐색

항상 가운데 수를 탐색하면
탐색 범위를 절반 씩 줄여 나갈 수 있음

배열 A에서 14를 찾는다고 해보자

1	4	6	11	14	16	22
0	1	2	3	4	5	6

이분 탐색

항상 가운데 수를 탐색하면
탐색 범위를 절반 씩 줄여 나갈 수 있음

배열 A에서 14를 찾는다고 해보자

1	4	6	11	14	16	22
0	1	2	3	4	5	6

이분 탐색

항상 가운데 수를 탐색하면
탐색 범위를 절반 씩 줄여 나갈 수 있음

배열 A에서 14를 찾는다고 해보자
 $A[3] = 11 < 14$

1	4	6	11	14	16	22
0	1	2	3	4	5	6

이분 탐색

항상 가운데 수를 탐색하면
탐색 범위를 절반 씩 줄여 나갈 수 있음

배열 A에서 14를 찾는다고 해보자
 $A[3] = 11 < 14$

1	4	6	11	14	16	22
0	1	2	3	4	5	6

이분 탐색

항상 가운데 수를 탐색하면
탐색 범위를 절반 씩 줄여 나갈 수 있음

배열 A에서 14를 찾는다고 해보자
 $A[5] = 16 > 14$

1	4	6	11	14	16	22
0	1	2	3	4	5	6

이분 탐색

항상 가운데 수를 탐색하면
탐색 범위를 절반 씩 줄여 나갈 수 있음

배열 A에서 14를 찾는다고 해보자
 $A[5] = 16 > 14$

1	4	6	11	14	16	22
0	1	2	3	4	5	6

이분 탐색

항상 가운데 수를 탐색하면
탐색 범위를 절반 씩 줄여 나갈 수 있음

배열 A에서 14를 찾는다고 해보자
 $A[4] = 14 == 14$

1	4	6	11	14	16	22
0	1	2	3	4	5	6

이분 탐색

항상 탐색 범위가 절반 씩 줄음
시간 복잡도는 $O(\log N)$

1	4	6	11	14	16	22
0	1	2	3	4	5	6

이분 탐색

탐색 범위의 시작점을 lo 끝점을 hi
lo 와 hi 의 중간을 mid 라고 두자

lo	mid				hi	
1	4	6	11	14	16	22
0	1	2	3	4	5	6

이분 탐색

mid의 값이 X의 값 초과이면 $hi = mid - 1$

$X = 4 < mid = 11$

lo	mid	hi
1	4	6
11	14	16
22		
0	1	2
3	4	5
6		

이분 탐색

mid의 값이 X의 값 초과이면 $hi = mid - 1$

$X = 4 < mid = 11$

lo hi mid

1	4	6	11	14	16	22
0	1	2	3	4	5	6

이분 탐색

mid의 값이 X의 값 초과이면 $hi = mid - 1$

$X = 4 < mid = 11$

lo mid hi

1	4	6	11	14	16	22
0	1	2	3	4	5	6

이분 탐색

mid의 값이 X의 값 미만이면 $lo = mid + 1$

$X = 16 > mid = 11$

lo		mid			hi	
1	4	6	11	14	16	22
0	1	2	3	4	5	6

이분 탐색

mid의 값이 X의 값 미만이면 $lo = mid + 1$

$X = 16 > mid = 11$

mid lo

hi

1	4	6	11	14	16	22
0	1	2	3	4	5	6

이분 탐색

mid의 값이 X의 값 미만이면 $lo = mid + 1$

$X = 16 > mid = 11$

lo mid hi

1	4	6	11	14	16	22
0	1	2	3	4	5	6

이분 탐색

이 과정을 hi 가 lo 보다 작아지거나
 $a[mid] == x$ 일 때 까지 반복

hi 가 lo 보다 작아질 때 까지 찾지 못하면
그 값은 배열에 없음

lo mid hi

1	4	6	11	14	16	22
0	1	2	3	4	5	6

이분 탐색

C++

Received Output:

1
4
-1

```
#include <iostream>
using namespace std;
int a[7] = { 1, 4, 6, 11, 14, 16, 22};

int binary_search(int x){
    int lo = 0, hi = 6; // A의 크기가 7이므로 hi = 6(7 - 1)
    int ret = -1; // A에 x가 존재하지 않을 경우 -1 반환

    while(lo <= hi){ // 시작점이 끝점보다 커지면 종료
        int mid = (hi + lo) / 2; // 중간값
        if(a[mid] == x){ // 현재 값이 탐색하고 있는 값이면
            ret = mid; break; // 정답은 mid, while문 종료
        }
        // 현재 값이 탐색하고 있는 값보다 크면
        if(a[mid] > x) hi = mid - 1; // mid 이상의 인덱스에는 x가 없음
        else lo = mid + 1; // 아니면 mid 이하의 인덱스에는 x가 없음
    }

    return ret;
}

int main(){
    cout << binary_search(4) << "\n"; // 1
    cout << binary_search(14) << "\n"; // 4
    cout << binary_search(15) << "\n"; // -1

    return 0;
}
```

이분 탐색

Python

Received Output:

1
4
-1

```
a = [1, 4, 6, 11, 14, 16, 22]

def binary_search(x):
    lo = 0
    hi = 6 # A의 크기가 7이므로 hi = 6(7 - 1)
    ret = -1 # A에 x가 존재하지 않을 경우 -1 반환

    while lo <= hi: # 시작점이 끝점보다 커지면 종료
        mid = (lo + hi) // 2 # 중간값
        if a[mid] == x: # 현재 값이 탐색하고 있는 값이면
            ret = mid # 정답은 mid
            break # while문 종료

        if a[mid] > x: # 현재 값이 탐색하고 있는 값보다 크면
            hi = mid - 1 # mid 이상의 인덱스에는 x가 없음
        else: # 현재 값이 탐색하고 있는 값보다 작으면
            lo = mid + 1 # mid 이하의 인덱스에는 x가 없음

    return ret

print(binary_search(4)) # 1
print(binary_search(14)) # 4
print(binary_search(15)) # -1
```

수 찾기 / 1920

백준 1920 / <https://www.acmicpc.net/problem/1920>

문제

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

입력

첫째 줄에 자연수 $N (1 \leq N \leq 100,000)$ 이 주어진다. 다음 줄에는 N개의 정수 $A[1], A[2], \dots, A[N]$ 이 주어진다. 다음 줄에는 $M (1 \leq M \leq 100,000)$ 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.

수 찾기 / 1920

배열 A의 크기 N이 주어지고 A값이 주어짐

$A = [4, 1, 5, 2, 3]$

자연수 M이 주어지고 M번에 대해서

X값이 주어지고 A에 X가 있으면 1 아니면 0 출력

예제 입력 1 [복사](#)

```
5
4 1 5 2 3
5
1 3 7 9 5
```

예제 출력 1 [복사](#)

```
1
1
0
0
1
```


수 찾기 / 1920

이분탐색을 사용하기 위해선 정렬을 해야함
각 언어의 기본 라이브러리 사용 / $O(N \log N)$

X가 M번 주어질 때 이분 탐색을 사용해서
값이 있으면 1 아니면 0 출력
이분 탐색을 M번 해야함 / $O(M \log N)$

예제 입력 1 [복사](#)

```
5
4 1 5 2 3
5
1 3 7 9 5
```

예제 출력 1 [복사](#)

```
1
1
0
0
1
```

수 찾기 / 1920

이분탐색을 사용하기 위해선 정렬을 해야함
각 언어의 기본 라이브러리 사용 / $O(N \log N)$

C++

<algorithm> 헤더 sort

배열을 정렬 할 때는
시작점이 s, 끝점이 e일 때
`sort(a + s, a + e + 1);`

```
#include <iostream>
#include <algorithm>
using namespace std;

int main(){
    int a[5] = {4, 1, 2, 5, 3};
    sort(a, a + 5);

    for(int i = 0; i < 5; i++) cout << a[i] << " ";
}
```

Received Output:

1 2 3 4 5

수 찾기 / 1920

C++

<algorithm> 헤더 sort

벡터를 정렬 할 때는

시작점이 s, 끝점이 e일 때

`sort(a.begin() + s , a.begin() + e + 1);`

전체를 다 정렬 할 때는

`sort(a.begin(), a.end());`

Received Output:

1 2 3 4 5

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

int main(){
    vector<int> a = {4, 1, 2, 5, 3};
    sort(a.begin(), a.end());

    for(int i = 0; i < 5; i++) cout << a[i] << " ";
}
```

수 찾기 / 1920

이분탐색을 사용하기 위해선 정렬을 해야함
각 언어의 기본 라이브러리 사용 / $O(N \log N)$

Python

List의 sort 함수 사용
`a.sort()`

```
a = [4, 1, 2, 5, 3]
a.sort()

for i in a:
    print(i, end = " ")
```

Received Output:
1 2 3 4 5

수 찾기 / 1920

C++

fastio를 사용하면 printf(), scanf()
사용이 불가능 하지만
입출력 속도가 증가 함

```
ios::sync_with_stdio(false);  
cin.tie(0); cout.tie(0);
```

Input:

```
5  
4 1 5 2 3  
5  
1 3 7 9 5
```

Received Output:

```
1  
1  
0  
0  
1
```

```
#include <iostream>  
#include <algorithm>  
using namespace std;  
const int MAX = 101010;  
int a[MAX];  
  
int binary_search(int x, int size){  
    int lo = 0, hi = size - 1; // A의 크기가 size이므로 hi = size - 1  
    int ret = -1; // A에 x가 존재하지 않을 경우 -1 반환  
  
    while(lo <= hi){ // 시작점이 끝점보다 커지면 종료  
        int mid = (hi + lo) / 2; // 중간값  
        if(a[mid] == x){ // 현재 값이 탐색하고 있는 값이면  
            ret = mid; break; // 정답은 mid, while문 종료  
        }  
        // 현재 값이 탐색하고 있는 값보다 크면  
        if(a[mid] > x) hi = mid - 1; // mid 이상의 인덱스에는 x가 없음  
        else lo = mid + 1; // 아니면 mid 이하의 인덱스에는 x가 없음  
    }  
  
    return ret;  
}  
  
int main(){  
    ios::sync_with_stdio(0); // fastio  
    cin.tie(0), cout.tie(0); // fastio  
  
    int n; cin >> n;  
    for(int i = 0; i < n; i++) cin >> a[i];  
    sort(a, a + n); // 이분탐색을 사용하기 위해선 정렬 해야함  
  
    int m; cin >> m;  
    while(m--){  
        int x; cin >> x;  
        int ret = binary_search(x, n); // 이분탐색 값  
        if(ret == -1) cout << 0 << "\n"; // -1 이면 A에 x가 없음  
        else cout << 1 << "\n"; // 아니면 A에 x가 있음  
    }  
  
    return 0;  
}
```

수 찾기 / 1920

Python

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

fastio를 사용하면
개행 문자까지 입력 받음
rstrip()으로 개행 문자를
제거해 줘야 함

Input:

```
5
4 1 5 2 3
5
1 3 7 9 5
```

Received Output:

```
1
1
0
0
1
```

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

def binary_search(x, size):
    lo = 0
    hi = size - 1 # A의 크기가 size 이므로 hi = size - 1
    ret = -1 # A에 x가 존재하지 않을 경우 -1 반환

    while lo <= hi: # 시작점이 끝점보다 커지면 종료
        mid = (lo + hi) // 2 # 중간값
        if a[mid] == x: # 현재 값이 탐색하고 있는 값이면
            ret = mid # 정답은 mid
            break # while문 종료

        if a[mid] > x: # 현재 값이 탐색하고 있는 값보다 크면
            hi = mid - 1 # mid 이상의 인덱스에는 x가 없음
        else: # 현재 값이 탐색하고 있는 값보다 작으면
            lo = mid + 1 # mid 이하의 인덱스에는 x가 없음

    return ret

n = int(input())
a = list(map(int, input().rstrip().split()))
a.sort() # 이분탐색을 사용하기 위해선 정렬 해야함

m = int(input())
x = list(map(int, input().rstrip().split()))

for i in range(m):
    ret = binary_search(x[i], n) # 이분탐색 값
    if ret == -1: # -1 이면 A에 x가 없음
        print(0)
    else: # 아니면 A에 x가 있음
        print(1)
```

질문?

매개 변수 탐색

매개 변수 탐색

최적화 문제를 결정 문제로 바꾸어
이분 탐색으로 문제를 해결 하는 것

결정 문제

Yes/No 로 답할 수 있는 문제

최적화 문제

최대값, 최소값을 찾는 문제

매개 변수 탐색

결정 문제

Yes/No 로 답할 수 있는 문제

최적화 문제

최대값, 최소값을 찾는 문제

결정 문제가 최적화 문제보다 항상 쉬움

최적화 문제를 해결 할 수 있으면

결정 문제를 항상 해결 할 수 있음

랜선 자르기 / 1654

백준 1654 / <https://www.acmicpc.net/problem/1654>

문제

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

이미 오영식은 자체적으로 K 개의 랜선을 가지고 있다. 그러나 K 개의 랜선은 길이가 제각각이다. 박성원은 랜선을 모두 N 개의 같은 길이의 랜선으로 만들고 싶었기 때문에 K 개의 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 K 개의 랜선으로 N 개의 랜선을 만들 수 없는 경우는 없다고 가정하자. 그리고 자를 때는 항상 센티미터 단위로 정수길이만큼 자른다고 가정하자. N 개보다 많이 만드는 것도 N 개를 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 N 이 입력된다. K 는 1이상 10,000이하의 정수이고, N 은 1이상 1,000,000이하의 정수이다. 그리고 항상 $K \leq N$ 이다. 그 후 K 줄에 걸쳐 이미 가지고 있는 각 랜선의 길이가 센티미터 단위의 정수로 입력된다. 랜선의 길이는 $2^{31}-1$ 보다 작거나 같은 자연수이다.

출력

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

랜선 자르기 / 1654

예제 입력 1 [복사](#)

```
4 11
802
743
457
539
```

예제 출력 1 [복사](#)

```
200
```

힌트

802cm 랜선에서 4개, 743cm 랜선에서 3개, 457cm 랜선에서 2개, 539cm 랜선에서 2개를 잘라내 모두 11개를 만들 수 있다.

랜선 자르기 / 1654

최적화 문제

최대값, 최소값을 찾는 문제

M개의 랜선을 만들 수 있는 랜선의 최대 길이

결정 문제

Yes/No 로 답할 수 있는 문제

랜선의 길이를 K로 잘랐을 때

M개 이상의 랜선을 만들 수 있는가

랜선 자르기 / 1654

결정 문제

랜선의 길이를 K 로 잘랐을 때
 M 개의 랜선을 만들 수 있는가

$K = 0, 1, 2 \dots$ 에 대해서 결정 문제를 해결
 $\text{Decision}(K) = \text{Yes}$ 인 K 의 최댓값을 찾으면 됨

모든 K 에 대해서 문제를 해결해야 함 \rightarrow 비효율적

랜선 자르기 / 1654

예제 입력 1 예제 출력 1

4 11

200

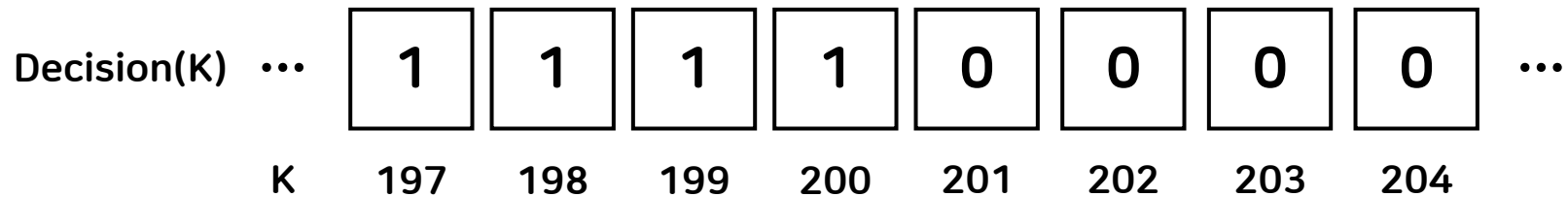
802

743

457

539

Decision(K)가 1인 K의 최댓값이 정답
결정 문제의 값은 항상 정렬 되어 있음 -> 이분 탐색



랜선 자르기 / 1654

이분 탐색

정답이 존재하는 구간의 $[lo, hi]$ 중간 지점 mid

Decision(mid)가 0이면 정답은 $[lo, mid - 1]$ 에 존재

Decision(mid)가 1이면 정답은 $[mid, hi]$ 에 존재

이분 탐색의 조건 -> 정렬

결정 문제의 값들이 정렬 되어 있지 않으면

매개 변수 탐색을 사용 할 수 없음

랜선 자르기 / 1654

C++

int를 사용하면 overflow
long long을 사용

$mid = (lo + hi + 1) / 2;$
올림 값 사용

Input:

4 11
802
743
457
539

Expected Output:

200

```
int main(){
    cin >> n >> m;
    for(int i = 1; i <= n; i++) cin >> a[i];
    cout << maximization(); // 최댓값 출력

    return 0;
}
```

```
#include <iostream>
using namespace std;
int n, m, a[1010101];

bool decision(long long cur){
    long long cnt = 0; // 만들 수 있는 랜선의 개수
    for(int i = 1; i <= n; i++){
        // 랜선의 길이를 cur 만큼 자를 때
        // a[i] / cur 만큼 랜선이 나옴
        cnt += a[i] / cur;
    }

    // 만들 수 있는 랜선의 개수가 m 이상이면 1 아니면 0 반환
    return cnt >= m;
}

int maximization(){
    // 정답의 범위는 1 ~ 2^31 - 1
    long long lo = 1, hi = (1 << 31) - 1;
    while(lo < hi){
        long long mid = (lo + hi + 1) / 2; // 중간값
        // 결정 문제의 답이 1 이면
        // 정답은 [mid, hi]에 존재
        if(decision(mid)) lo = mid;

        // 결정 문제의 답이 0 이면
        // 정답은 [lo, mid - 1]에 존재
        else hi = mid - 1;
    }

    return lo;
}
```


랜선 자르기 / 1654

Python

$\text{mid} = (\text{lo} + \text{hi} + 1) // 2;$
올림 값 사용

Input:
4 11
802
743
457
539

Expected Output:
200

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

n, m = list(map(int, input().rstrip().split()))
a = [int(input().rstrip()) for _ in range(n)]

def decision(cur):
    cnt = 0 # 만들 수 있는 랜선의 개수
    for i in a:
        # 랜선의 길이를 cur 만큼 자를 때
        # a[i] // cur 만큼 랜선이 나옴
        cnt += i // cur

    # 만들 수 있는 랜선의 개수가 m 이상이면 1 아니면 0 반환
    return cnt >= m

def maximization():
    # 정답의 범위는 1 ~ 2^31 - 1
    lo = 1
    hi = 2 ** 31 - 1
    while lo < hi:
        mid = (lo + hi + 1) // 2 # 중간값
        # 결정 문제의 답이 1 이면
        # 정답은 [mid, hi]에 존재
        if(decision(mid)):
            lo = mid

        # 결정 문제의 답이 0 이면
        # 정답은 [lo, mid - 1]에 존재
        else:
            hi = mid - 1
    return lo

print(maximization())
```

매개 변수 탐색

mid 값을 올림 하는 이유

lo와 hi의 차이가 1 일 때 올림을 하지 않으면

EX) $lo = 5, hi = 6, mid = (5 + 6) / 2 = 5$

decision(mid)가 1이면 $lo = mid$

항상 $lo = 5, hi = 6, mid = 5$ 로 갱신 됨

올림을 하면 $mid = lo = 6$ 으로 갱신되어서 무한루프 X

반대로 최솟값을 구할 때는 mid 값을 내림 해야함

$mid = (lo + hi) / 2$

매개 변수 탐색

C++

```
int maximization(){
    // 정답의 범위는 1 ~ N
    long long lo = 1, hi = N;
    while(lo < hi){
        long long mid = (lo + hi + 1) / 2; // 중간값
        // 결정 문제의 답이 1 이면
        // 정답은 [mid, hi]에 존재
        if(decision(mid)) lo = mid;

        // 결정 문제의 답이 0 이면
        // 정답은 [lo, mid - 1]에 존재
        else hi = mid - 1;
    }

    return lo;
}
```

```
int minimization(){
    // 정답의 범위는 1 ~ N
    int lo = 1, hi = N;
    while(lo < hi){
        int mid = (lo + hi) / 2; // 중간값
        // 결정 문제의 답이 1 이면
        // 정답은 [lo, mid]에 존재
        if(decision(mid)) hi = mid;

        // 결정 문제의 답이 0 이면
        // 정답은 [mid + 1, hi]에 존재
        else lo = mid + 1;
    }

    return lo;
}
```

Decision(K) 11111**1**000000

000000**1**11111

매개 변수 탐색

Python

```
def maximization():  
    # 정답의 범위는 1 ~ N  
    lo = 1  
    hi = N  
    while lo < hi:  
        mid = (lo + hi + 1) // 2 # 중간값  
        # 결정 문제의 답이 1 이면  
        # 정답은 [mid, hi]에 존재  
        if(decision(mid)):  
            lo = mid  
  
        # 결정 문제의 답이 0 이면  
        # 정답은 [lo, mid - 1]에 존재  
        else:  
            hi = mid - 1  
    return lo
```

```
def minimization():  
    # 정답의 범위는 1 ~ N  
    lo = 1  
    hi = N  
    while(lo < hi):  
        mid = (lo + hi) // 2 # 중간값  
        # 결정 문제의 답이 1 이면  
        # 정답은 [lo, mid]에 존재  
        if(decision(mid)):  
            hi = mid  
  
        # 결정 문제의 답이 0 이면  
        # 정답은 [mid + 1, hi]에 존재  
        else:  
            lo = mid + 1  
    return lo
```

Decision(K) 11111**1**000000

000000**1**11111

질문?

휴게소 세우기 / 1477

백준 1477 / <https://www.acmicpc.net/problem/1477>

문제

다솜이는 유료 고속도로를 가지고 있다. 다솜이는 현재 고속도로에 휴게소를 N 개 가지고 있는데, 휴게소의 위치는 고속도로의 시작으로부터 얼마나 떨어져 있는지로 주어진다. 다솜이는 지금 휴게소를 M 개 더 세우려고 한다.

다솜이는 이미 휴게소가 있는 곳에 휴게소를 또 세울 수 없고, 고속도로의 끝에도 휴게소를 세울 수 없다. 휴게소는 정수 위치에만 세울 수 있다.

다솜이는 이 고속도로를 이용할 때, 모든 휴게소를 방문한다. 다솜이는 휴게소를 M 개 더 지어서 휴게소가 없는 구간의 길이의 최댓값을 최소로 하려고 한다. (반드시 M 개를 모두 지어야 한다.)

예를 들어, 고속도로의 길이가 1000이고, 현재 휴게소가 {200, 701, 800}에 있고, 휴게소를 1개 더 세우려고 한다고 해보자.

일단, 지금 이 고속도로를 타고 달릴 때, 휴게소가 없는 구간의 최댓값은 200~701구간인 501이다. 하지만, 새로운 휴게소를 451구간에 짓게 되면, 최대가 251이 되어서 최소가 된다.

입력

첫째 줄에 현재 휴게소의 개수 N , 더 지으려고 하는 휴게소의 개수 M , 고속도로의 길이 L 이 주어진다. 둘째 줄에 현재 휴게소의 위치가 공백을 사이에 두고 주어진다. $N = 0$ 인 경우 둘째 줄은 빈 줄이다.

출력

첫째 줄에 M 개의 휴게소를 짓고 난 후에 휴게소가 없는 구간의 최댓값의 최솟값을 출력한다.

휴게소 세우기 / 1477

첫째 줄에 M개의 휴게소를 짓고 난 후에 휴게소가 없는 구간의 최댓값의 최솟값을 출력한다.

예제 입력 1 [복사](#)

```
6 7 800
622 411 201 555 755 82
```

예제 출력 1 [복사](#)

```
70
```

예제 입력 2 [복사](#)

```
3 1 1000
200 701 800
```

예제 출력 2 [복사](#)

```
251
```

예제 입력 3 [복사](#)

```
3 1 1000
300 701 800
```

예제 출력 3 [복사](#)

```
300
```

휴게소 세우기 / 1477

최적화 문제

M개의 휴게소를 추가로 지었을 때
휴게소 간의 거리의 최댓값의 최솟값

결정 문제

휴게소 세우기 / 1477

최적화 문제

M개의 휴게소를 추가로 지었을 때
휴게소 간의 거리의 최댓값의 최솟값

결정 문제

휴게소 간의 거리의 최댓값을 K로 만들 때
M개 이하의 휴게소가 추가로 필요한가

휴게소 세우기 / 1477

결정 문제

휴게소 간의 거리의 최댓값을 K 로 만들 때
 M 개 이하의 휴게소가 추가로 필요한가

M 개 초과인 휴게소가 필요하면 최댓값을 K 로 줄일 수 없음
 M 개 미만의 휴게소가 필요하면 남은 휴게소를
거리가 1인 휴게소로 배치하면 됨

휴게소 세우기 / 1477

결정 문제

휴게소 간의 거리의 최댓값을 K 로 만들 때
 M 개 이하의 휴게소가 추가로 필요한가

이분 탐색

K 가 감소하면 항상 필요한 휴게소 수는 증가함

K 가 증가하면 항상 필요한 휴게소 수는 감소함

Decision(K) -> 0000011111

매개 변수 탐색 가능

휴게소 세우기 / 1477

결정 문제

휴게소 간의 거리의 최댓값을 K 로 만들 때
 M 개 이하의 휴게소가 추가로 필요한가

고속도로의 길이가 L 이므로 결정 문제의 범위는 $1 \sim L$
일단 휴게소의 위치들을 정렬해보자

휴게소 세우기 / 1477

예제 입력 1

6 7 800

622 411 201 555 755 82 70

예제 출력 1

정렬

82 201 411 555 622 755

휴게소 세우기 / 1477

6 7 800

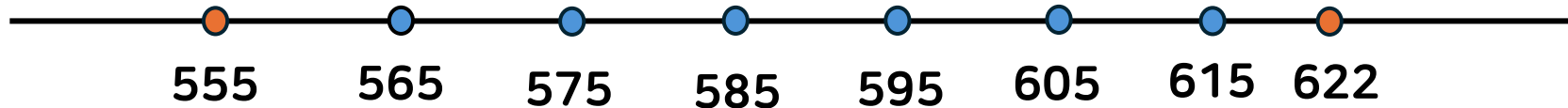
정답

82 201 411 555 622 755

70

휴게소 사이의 거리의 최댓값이 K일 때
각 휴게소 사이에는 (휴게소의 위치 차이 / K) 만큼
추가로 휴게소를 배치 해야 함

EX) 555 622 -> 차이 67, $K = 10 \rightarrow 6$



휴게소 세우기 / 1477

6 7 800

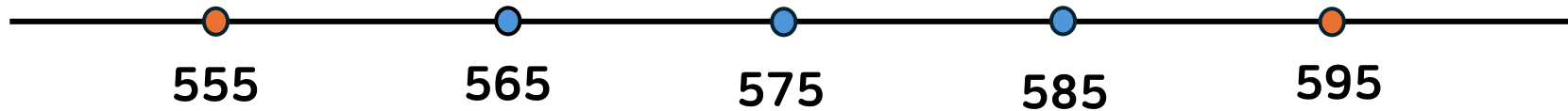
정답

82 201 411 555 622 755

70

휴게소 간의 거리 차이가 K의 배수일 때는 -1을 해줘야 함

EX) 555 595 -> 차이 40, $K = 10 \rightarrow 3$



휴게소 세우기 / 1477

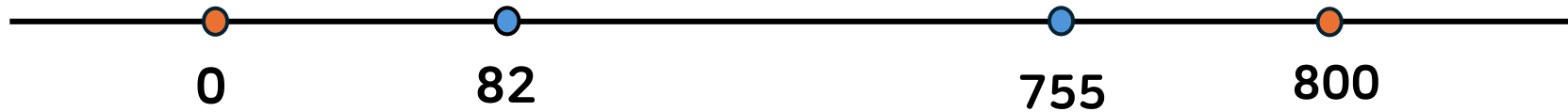
6 7 800

정답

82 201 411 555 622 755

70

시작점과 휴게소의 거리, 끝점과의 휴게소의 거리도
휴게소가 없는 구간이므로 시작점, 끝점도 휴게소로 지정



휴게소 세우기 / 1477

C++

Input:
6 7 800
622 411 201 555 755 82

Expected Output:

70

Received Output:

70

```
int main(){
    cin >> n >> m >> l;
    for(int i = 1; i <= n; i++) cin >> a[i];
    a[0] = 0; a[n + 1] = l; // 시작점 0, 끝점 l
    sort(a, a + n + 2); // 정렬

    cout << minimization();
    return 0;
}
```

```
#include <iostream>
#include <algorithm>
using namespace std;

int a[1010], n, m, l;

bool decision(int cur){
    // 휴게소 사이의 거리의 최댓값을 cur로 만들기 위해
    // 필요한 추가 휴게소의 개수
    int cnt = 0;
    for(int i = 1; i <= n + 1; i++){
        int diff = a[i] - a[i - 1]; // 휴게소의 거리
        cnt += diff / cur; // (차이 / 최댓값) 만큼 추가로 휴게소를 설치
        if(diff % cur == 0) cnt--; // 차이가 최댓값의 배수면 1을 빼줌
    }

    // 설치해야 하는 휴게소의 개수가 m 이하면 1
    // 아니면 0
    return cnt <= m;
}

int minimization(){
    // 정답의 범위는 1 ~ l
    int lo = 1, hi = l;
    while(lo < hi){
        int mid = (lo + hi) / 2; // 중간값
        // 결정 문제의 답이 1 이면
        // 정답은 [lo, mid]에 존재
        if(decision(mid)) hi = mid;

        // 결정 문제의 답이 0 이면
        // 정답은 [mid + 1, hi]에 존재
        else lo = mid + 1;
    }

    return lo;
}
```

휴게소 세우기 / 1477

Python

```
Input:
6 7 800
622 411 201 555 755 82

Expected Output:
70

Received Output: Set
70
```

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

n, m, l = list(map(int, input().rstrip().split()))
a = list(map(int, input().rstrip().split()))
a.append(0) # 시작점 0
a.append(l) # 끝점 l
a.sort() # 정렬
```

```
def decision(cur):
    # 휴게소 사이의 거리의 최댓값을 cur로 만들기 위해
    # 필요한 추가 휴게소의 개수
    cnt = 0
    for i in range(1, n + 2):
        diff = a[i] - a[i - 1] # 휴게소의 거리
        cnt += diff // cur # (차이 // 최댓값) 만큼 추가로 휴게소를 설치
        if diff % cur == 0:
            cnt -= 1 # 차이가 최댓값의 배수면 1을 빼줌

    # 설치해야 하는 휴게소의 개수가 m 이하면 1
    # 아니면 0
    return cnt <= m

def minimization():
    # 정답의 범위는 1 ~ l
    lo = 1
    hi = l
    while(lo < hi):
        mid = (lo + hi) // 2 # 중간값
        # 결정 문제의 답이 1 이면
        # 정답은 [lo, mid]에 존재
        if(decision(mid)):
            hi = mid

        # 결정 문제의 답이 0 이면
        # 정답은 [mid + 1, hi]에 존재
        else:
            lo = mid + 1
    return lo

print(minimization())
```

질문?

기본 과제

수 찾기 - <https://www.acmicpc.net/problem/1920>

랜선 자르기 - <https://www.acmicpc.net/problem/1654>

공유기 설치 - <https://www.acmicpc.net/problem/2110>

휴게소 세우기 - <https://www.acmicpc.net/problem/1477>

모자이크 - <https://www.acmicpc.net/problem/2539>

고생하셨습니다