



# ALOHA

## #2주차

Binary Search

Parametric Search

LIS

# #CH.1

Binary Search



[illegible]



## Q. 프로그램으로 이 작업을 한다면?

---

- Linear Search(선형탐색)을 이용한 방법
- 앞에서부터 하나하나 탐색
- 전체 데이터 크기가 N이라면 최대 N번 탐색
- 시간 복잡도  $O(N)$



**더 좋은 방법이 없을까요?**

# Binary Search(이진 탐색)

EX) N = 10

	Left				Mid		Right			
ARR	1	2	3	4	5	6	7	8	9	10

**Left** : 찾고자 하는 값이 속한 구간의 왼쪽 끝 (시작은 1)

**Right** : 찾고자 하는 값이 속한 구간의 오른쪽 끝 (시작은 n)

**Mid** : 구간의 가운데  $== (Left + Right) / 2$

**Value** : 찾고자 하는 값

\* Left, Right, Mid 모두 값이 아닌 인덱스이다!

조건 ) 배열이 **정렬**되어 있어야함. (예시에선 오름차순)

# Binary Search(이진 탐색)

Value 가 Arr[mid]보다 **크다** => Value가 **Mid + 1 ~ Right** 구간에 있음

Value 가 Arr[mid]보다 **작다** => Value가 **Left ~ Mid - 1** 구간에 있음

Value 가 Arr[mid]와 **같다** => 찾았으므로 **끝낸다**

조건 ) 배열이 **정렬**되어 있어야함. (예시에선 오름차순)

# Binary Search(이진 탐색)

식으로 나타내면 아래와 같아요

$\text{Value} > \text{Arr}[\text{mid}] \Rightarrow \text{Left} = \text{mid} + 1$

$\text{Value} < \text{Arr}[\text{mid}] \Rightarrow \text{Right} = \text{mid} - 1$

$\text{Value} == \text{Arr}[\text{mid}] \Rightarrow \text{return mid}$

조건 ) 배열이 **정렬**되어 있어야함. (예시에선 오름차순)



# Binary Search(이진 탐색)

1. Left, Right 값 설정
2. Mid 값 구하기
3. Arr[mid] 와 Value를 비교해서 Left, Right값 변경
4. Value == Arr[mid] 일때까지 반복!

Q. Value가 없는 경우 무한히 반복하는 거 아니가요?

A. Value를 찾지 못할 경우에 Left > Right 가 됩니다.

따라서 Left <= Right 인 동안만 반복해주면 해결할 수 있어요.

조건 ) 배열이 정렬되어 있어야함. (예시에선 오름차순)

## 연습문제



#1920  
수 찾기

Linear Search를  
이용하면 TLE를 받아요

```

19 int main(void)
20 {
21     std::cin.tie(0);
22     std::ios_base::sync_with_stdio(false);
23
24     int N, x;
25
26     std::cin >> N;
27     for (int i = 0; i < N; i++)
28     {
29         std::cin >> x;
30         v.push_back(x);
31     }
32
33     std::sort(v.begin(), v.end());
34
35     int M;
36     std::cin >> M;
37     for (int i = 0; i < M; i++)
38     {
39         std::cin >> x;
40         std::cout << binarySearch(0, N - 1, x) << "\n";
41     }
42
43     return 0;
44 }

```

이분탐색은 정렬부터!

## #1920 수 찾기

Main 함수에선 주어지는 입력을 받고 **이분탐색**을 통해 문제를 해결할게요.

선형탐색으로 문제를 푼다면 M번의 탐색마다 각각 N번의 탐색을 하기 때문에  $O(NM)$ 이라는 시간 초과나기에 좋은 시간 복잡도가 나와요.

binarySearch()함수의 구현은 다음 슬라이드에서 확인할게요.

## 1. 반복문을 통한 구현

```
7 int binarySearch(int left, int right, int value)
8 {
9     while (left <= right)
10    {
11        int mid = (left + right) / 2;
12        if (value > v[mid]) left = mid + 1;
13        else if (value < v[mid]) right = mid - 1;
14        else return 1;
15    }
16    return 0;
17 }
```

## 2. 재귀함수를 통한 구현

```
7 int binarySearch(int left, int right, int value)
8 {
9     int mid = (left + right) / 2;
10    if (left > right)
11        return 0;
12    if (value > v[mid])
13        return binarySearch(mid + 1, right, value);
14    else if (value < v[mid])
15        return binarySearch(left, mid - 1, value);
16    else
17        return 1;
18 }
```

# #1920 수 찾기

binarySearch() 함수의 구현은 두가지 방법 중에 편한걸로 하시면 돼요.

구현에 필요한 개념은 아래에서 벗어나지 않으니  
까 이해하는게 어렵지 않을 거예요.

Value 가 Arr[mid]보다 **크다** => Value가 **Mid + 1 ~ Right** 구간에 있음

Value 가 Arr[mid]보다 **작다** => Value가 **Left ~ Mid - 1** 구간에 있음

Value 가 Arr[mid]와 **같다** => 찾았으므로 **끝낸다**

과제 (어렵지 않아요~)



#10816  
숫자 카드 2

이분탐색

과제 (어렵지 않아요~)



#3020  
개똥벌레

약간의 응용

과제 (어렵지 않아요~)



#2792  
보석 상자

질투심을 최소로!

# #CH.2

Parametric Search







# Parametric Search

---

1. 답이 가능한 범위를 둔다.
2. 이진 탐색을 통해 범위를 좁히며 답을 결정한다.

연습 문제를 풀면서 이해하는게 더 수월해요!

## 연습문제



#1654  
랜선 자르기

Parametric  
Search 의 정석

# #1654 랜선 자르기

풀이 - 브루트 포스

1. 랜선의 길이(L)를 1씩 증가시킨다.
2. 가지고 있는 조각을 L로 나눈 몫의 합(만들 수 있는 길이 L 랜선의 개수)을 구함.
3. N개의 조각을 만들 수 없는 경우 발생
4. 그때의 L-1이 정답

- 시간 복잡도 :  $O(KL)$
- L의 최댓값이  $2^{31} - 1$  이기 때문에 시간초과

# #1654 랜선 자르기

풀이 – Parametric Search

- L 길이의 조각을 N개 이상 만들 수 있다면 1~L-1 길이의 조각도 N개 이상 만들 수 있음!



- 이진 탐색으로 L의 범위를 좁혀 가면서 YES 에 해당하는 L의 최댓값을 찾자!

# #1654 랜선 자르기

풀이 - Parametric Search

1. **이진탐색**으로 L의 값을 결정
2. L의 길이로 최대 몇 조각을 만들 수 있는지 계산
3. N개 이상의 조각을 만들 수 있다면  $\Rightarrow \text{Left} = \text{Mid} + 1$
4. N개의 조각을 만들 수 없다면  $\Rightarrow \text{Right} = \text{Mid} - 1$

반복

한번의 탐색마다  $O(K)$ 로 최대 몇 조각을 만들 수 있는지 계산하기 때문에  
시간 복잡도 :  $O(K \lg L)$  (  $L = 2147483647$  )

```
34 | int main(void)
35 | {
36 |     std::cin.tie(0);
37 |     std::ios_base::sync_with_stdio(false);
38 |
39 |     std::cin >> K >> N;
40 |
41 |     int x;
42 |     for (int i = 0; i < K; i++)
43 |     {
44 |         std::cin >> x;
45 |         v.push_back(x);
46 |     }
47 |
48 |     std::cout << parametricSearch(1, 2147483647) << "\n";
49 |
50 |     return 0;
51 | }
```

## #1654 랜선 자르기

Main 함수는 간단해요.

입력을 받고 parametricSearch()함수를 호출해서 정답을 출력합니다.

```

7  ✓ long long parametricSearch(long long left, long long right)
8  {
9      long long res = 0;
10  ✓ while (left <= right)
11  {
12      long long mid = (left + right) / 2;
13      long long cnt = 0;
14  ✓ for (int i = 0; i < K; i++)
15      {
16          cnt += v[i] / mid;
17      }
18  ✓ if (cnt >= N)
19      {
20  ✓     if (res < mid)
21      {
22          res = mid;
23      }
24      left = mid + 1;
25  }
26  ✓ else
27  {
28      right = mid - 1;
29  }
30  }
31  return res;
32  }

```

overflow 주의!

## #1654 랜선 자르기

mid 값을 구한 뒤, **최대 몇 조각**을 만들 수 있는지 cnt 변수로 세 준 다음 그 값에 따라 left와 right를 조절해줘요.

만약 N개 이상의 조각을 만들 수 있다면 애는 답이 될 수 있는 '후보' 이고 이 '후보' 중에 가장 큰 값을 리턴 해주면 돼요.

풀이 - Parametric Search

1. **이진탐색**으로 L의 값을 결정
2. L의 길이로 최대 몇 조각을 만들 수 있는지 계산
3. N개 이상의 조각을 만들 수 있다면 => **Left = Mid + 1**
4. N개의 조각을 만들 수 없다면 => **Right = Mid - 1**

반복

과제 (어렵지 않아요~)



#2512  
예산

연습문제와 다르지  
않아요



과제 (어렵지 않아요~)



#1072  
게임

범위를 조절하는  
기준을 잘 알아야겠죠?

과제 (어렵지 않아요~)



#2805  
나무 자르기

높이의 최댓값

과제 (어렵지 않아요~)



#2613  
숫자구슬

각 그룹의 합의 최댓값  
을 최소로

# #CH.3

LIS 에 대해 알아보자!



# LIS란?

---

Longest Increasing Subsequence

Longest 최장

Increasing 증가

Subsequence 부분 수열

# LIS

---

우리는 이미  $O(N^2)$  풀이에 대해 배웠다...

하지만..

## 연습문제





# #12015

가장 긴 증가하는  
부분 수열 2

DP인가?  
TLE를 받아보자

# LIS

24734386	ingyu1008	  12015	시간 초과
----------	-----------	---	-------



# LIS

---

첫째 줄에 수열 A의 크기  $N$  ( $1 \leq N \leq 1,000,000$ )이 주어진다.

**$\Rightarrow O(N^2)$  풀이로 구현하는 경우,  $(1,000,000)^2 = 1,000,000,000,000$**

## lower\_bound(first, last, value)

---

정렬 되어있는 배열에서 구간 **[first, last)** 에서 처음으로 **value 이상인 수**가 나타나는 위치를 **Binary Search(이진 탐색)**로 찾아주는 함수입니다.

표준 헤더 `<algorithm>` 에 들어있습니다.

시간 복잡도 :  **$O(\lg N)$**

# lower\_bound(first, last, value)

예시)

arr[i]	0	1	2	3	4
value	5	13	30	54	56

`lower_bound(arr, arr + 5, 6) = 1`

`lower_bound(arr, arr + 5, 100) = 5` (해당하는 위치가 없으므로 5 반환)

# LIS

---

1. 전체 문제 : 길이  $N$ 의 수열에서 LIS의 최대 길이를 구하는 것
2. 부분 문제 :  $K < N$ 인 어떤  $K$ 에 대해  $arr[1] \sim arr[k]$ 까지의 LIS의 최대 길이를 구하는 것.
3. DP Table :  $DP[i]$  = 길이  $i$ 의 LIS중, 마지막 수가 가장 작은 LIS의 마지막 수

설명이 복잡하니 예시를 통해 이해해주세요!

# LIS

`arr = {2, 6, 9, 4, 5, 7, 1}`

idx	0	1	2	3	4	5	6
arr[idx]	2	6	9	4	5	7	1

idx	0	1	2	3	4	5	6
dp[idx]							

# LIS

arr = {2, 6, 9, 4, 5, 7, 1}



idx	0	1	2	3	4	5	6
arr[idx]	2	6	9	4	5	7	1



idx	0	1	2	3	4	5	6
dp[idx]							

# LIS

arr = {2, 6, 9, 4, 5, 7, 1}



idx	0	1	2	3	4	5	6
arr[idx]	2	6	9	4	5	7	1



idx	0	1	2	3	4	5	6
dp[idx]	2						

# LIS

arr = {2, 6, 9, 4, 5, 7, 1}



idx	0	1	2	3	4	5	6
arr[idx]	2	6	9	4	5	7	1



idx	0	1	2	3	4	5	6
dp[idx]	2						



# LIS

arr = {2, 6, 9, 4, 5, 7, 1}



idx	0	1	2	3	4	5	6
arr[idx]	2	6	9	4	5	7	1



idx	0	1	2	3	4	5	6
dp[idx]	2	6					

# LIS

arr = {2, 6, 9, 4, 5, 7, 1}



idx	0	1	2	3	4	5	6
arr[idx]	2	6	9	4	5	7	1



idx	0	1	2	3	4	5	6
dp[idx]	2	6					

# LIS

arr = {2, 6, 9, 4, 5, 7, 1}



idx	0	1	2	3	4	5	6
arr[idx]	2	6	9	4	5	7	1



idx	0	1	2	3	4	5	6
dp[idx]	2	6	9				

# LIS

arr = {2, 6, 9, 4, 5, 7, 1}



idx	0	1	2	3	4	5	6
arr[idx]	2	6	9	4	5	7	1



idx	0	1	2	3	4	5	6
dp[idx]	2	6	9				

# LIS

arr = {2, 6, 9, 4, 5, 7, 1}



idx	0	1	2	3	4	5	6
arr[idx]	2	6	9	4	5	7	1



idx	0	1	2	3	4	5	6
dp[idx]	2	4	9				

# LIS

arr = {2, 6, 9, 4, 5, 7, 1}



idx	0	1	2	3	4	5	6
arr[idx]	2	6	9	4	5	7	1



idx	0	1	2	3	4	5	6
dp[idx]	2	4	9				

# LIS

arr = {2, 6, 9, 4, 5, 7, 1}



idx	0	1	2	3	4	5	6
arr[idx]	2	6	9	4	5	7	1



idx	0	1	2	3	4	5	6
dp[idx]	2	4	5				

# LIS

arr = {2, 6, 9, 4, 5, 7, 1}



idx	0	1	2	3	4	5	6
arr[idx]	2	6	9	4	5	7	1



idx	0	1	2	3	4	5	6
dp[idx]	2	4	5				



# LIS

arr = {2, 6, 9, 4, 5, 7, 1}



idx	0	1	2	3	4	5	6
arr[idx]	2	6	9	4	5	7	1



idx	0	1	2	3	4	5	6
dp[idx]	2	4	5	7			

# LIS

arr = {2, 6, 9, 4, 5, 7, 1}



idx	0	1	2	3	4	5	6
arr[idx]	2	6	9	4	5	7	1



idx	0	1	2	3	4	5	6
dp[idx]	2	4	5	7			

# LIS

arr = {2, 6, 9, 4, 5, 7, 1}



idx	0	1	2	3	4	5	6
arr[idx]	2	6	9	4	5	7	1



idx	0	1	2	3	4	5	6
dp[idx]	1	4	5	7			

# LIS

arr = {2, 6, 9, 4, 5, 7, 1}

idx	0	1	2	3	4	5	6
dp[idx]	1	4	5	7			

dp배열의 길이 = LIS의 길이!

BUT 수를 덮어쓰면서 dp배열이 만들어 지기때문에 **dp배열의 원소가 LIS의 원소는 아님!**

```

std::vector<int> LIS;
for (int i = 0; i < N; i++)
{
    int idx = std::lower_bound(LIS.begin(), LIS.end(), v[i]) - LIS.begin();
    if (idx == LIS.size())
    {
        LIS.push_back(v[i]);
    }
    else
    {
        LIS[idx] = v[i];
    }
}

std::cout << LIS.size() << "\n";

```

LIS의 마지막 값보다 현재 값이 큰 경우

LIS 배열의 길이가 답

## #12015 가장 긴 증가하는 부분 수열2

LIS 배열의 마지막 값보다 현재 값이 더 큰 경우 LIS 배열 뒤에 push\_back 해줍니다.

그렇지 않은 경우엔 벡터에서  $v[i]$  이상의 값이 처음으로 나타나는 위치에 있는 값을  $v[i]$ 로 대체해줍니다.

# 두 LIS 알고리즘의 비교

	$O(N^2)$	$O(N \lg N)$
구현 방법	바깥쪽 for문으로 전체를 순회하면서 $O(N)$ , 안쪽으로 현재 원소보다 작은 걸 찾으면서 $O(N)$	배열 전체를 순회하면서 $O(N)$ , lower_bound로 현재 원소보다 크거나 같은 수를 찾는데 $O(\lg N)$
DP[i]의 정리	arr[i]를 마지막으로 하는 LIS의 길이	길이가 i인 LIS 중 마지막 수가 가장 작은 LIS의 마지막 수
DP Table의 크기 (M)	$N == M$	$M == (\text{LIS 길이}) \leq N$
알 수 있는 것	LIS와 그 길이	LIS의 길이 only

과제



#3745  
오름세

길이만 구하면 됩니다

과제



#2352  
반도체 설계

조금 생각이  
필요한 문제



과제



#1365  
꼬인 전깃줄

어렵지 않아요~



다음 시간에 만나요~