

ICPC Sinchon



06. Binary Search, Divide and Conquer

06. 이분 탐색, 분할 정복

2023 Summer Algorithm Camp

목차

1. 이분 탐색
2. 이분 탐색 STL
3. 파라메트릭 서치 (매개 변수 탐색, Parametric Search)
4. 분할 정복
 - 빠른 거듭제곱

2023 Summer Algorithm Camp

이분 탐색

- 선형 탐색
 - 데이터를 처음부터 끝까지 차례대로 순회하면서 자료를 탐색하는 방법
 - 자료의 수가 N 개라면 시간 복잡도는 $O(N)$
- 이분 탐색
 - 정렬된 자료에서만 사용할 수 있다.
 - 원하는 자료가 있는지 없는지 $O(\log N)$ 에 판별할 수 있다.
- 탐색을 여러 번 해야 할 때 효율적이다.

2023 Summer Algorithm Camp

이분 탐색

- 매번 탐색 범위를 절반씩 줄이면서 탐색하는 알고리즘

- 찾을 자료 : 8

1	4	5	7	8	11	14
↑			↑			↑

- 첫 번째 탐색

- $l=0, r=6$

- $mid = (0+6)/2 = 3$

- $arr[3] = 7 < 8$ 이므로 오른쪽으로 범위를 줄임

- $l = 4$ 로 변경

1	4	5	7	8	11	14
			↑	↑		↑

2023 Summer Algorithm Camp

이분 탐색

- 매번 탐색 범위를 절반씩 줄이면서 탐색하는 알고리즘

- 찾을 자료 : 8

1	4	5	7	8	11	14
				↑	↑	↑

- 두 번째 탐색

- $l=4, r=6$

- $mid = (4+6)/2 = 5$

- $arr[5] = 11 > 8$ 이므로 왼쪽으로 범위를 줄임

- $r = 4$ 로 변경

1	4	5	7	8	11	14
				↑	↑	


2023 Summer Algorithm Camp

이분 탐색

- 매번 탐색 범위를 절반씩 줄이면서 탐색하는 알고리즘

- 찾을 자료 : 8

1	4	5	7	8	11	14
---	---	---	---	---	----	----



- 세 번째 탐색
 - $l=4, r=4$
 - $mid = (4+4)/2 = 4$
 - $arr[4] = 8 = 8$ 이므로 원하는 자료를 찾음, 탐색을 종료한다.

질문?

2023 Summer Algorithm Camp

이분 탐색

- 매번 탐색 범위를 절반씩 줄이면서 탐색하는 알고리즘
- 찾으려고 하는 자료의 값을 x 라고 하자.
- 구체적으로 표현하면 처음 탐색 범위는 다음과 같이 정한다.
 - $l = 1, r = N$
 - N 은 자료의 길이
- $l \leq r$ 인 동안 다음 탐색 과정을 계속 반복한다.
 - $mid = \lfloor (l + r) / 2 \rfloor$
 - case work)
 - $arr[mid] < x$ 이면, $[l, mid]$ 에는 원하는 자료가 없다. (x 미만인 자료들만 있다.) 따라서, $l = mid + 1$ 로 변경한다.
 - $arr[mid] = x$ 이면, 원하는 자료를 찾은 것이다.
 - $arr[mid] > x$ 이면, $[mid, r]$ 에는 원하는 자료가 없다. (x 초과인 자료들만 있다.) 따라서, $r = mid - 1$ 로 변경한다.
- $l > r$ 이면, 원하는 자료를 찾지 못한 것이다.

2023 Summer Algorithm Camp

이분 탐색

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    vector<int> arr = {1, 4, 5, 6, 8, 11, 14};
    int x = 8; // 찾으려는 자료
    int l = 0, r = 6;
    while (l <= r) {
        int mid = (l + r) / 2;
        if (arr[mid] < x) l = mid + 1;
        else if (arr[mid] == x) {
            cout << mid; // 4
            return 0;
        } else r = mid - 1;
    }
    cout << "Not Found";
}
```

2023 Summer Algorithm Camp

이분 탐색

- 시간 복잡도 : $O(\log N)$
- 매번 탐색 범위가 절반씩 줄어든다.
- 탐색 범위가 1일 때부터 반대로 생각해 보면, 탐색 범위가 두 배씩 늘어난다고 생각할 수 있다.
 - $2 * 2 * \dots * 2 = 2^k = N$
 - $k = \log_2 N$
 - $O(\log N)$

질문?

2023 Summer Algorithm Camp

이분 탐색

- 정확히 x 인 자료를 찾는 것이 아니라, x 이상인 수가 처음으로 나오는 위치를 찾아보자.
 - C++의 algorithm 헤더에는 이 역할을 하는 함수로 `lower_bound`가 정의되어 있다.
 - 비슷하게 x 초과인 수가 처음 나오는 위치를 찾는 함수로는 `upper_bound`가 있다.
- $l < r$ 인 동안 다음 탐색 과정을 계속 반복한다.
 - $mid = \lfloor (l + r) / 2 \rfloor$
 - case work)
 - $arr[mid] < x$ 이면, $[l, mid]$ 에는 x 미만인 수만 있는 것이므로 $l = mid + 1$ 로 변경한다.
 - $arr[mid] \geq x$ 이면, $[l, mid]$ 으로 범위를 줄여도 된다. (mid 가 답의 후보가 되므로, $[mid + 1, r]$ 은 볼 필요가 없다.) 따라서, $r = mid$ 로 변경한다.
- 탐색 과정이 끝났을 때 $l = r$ 이며, l 이 x 이상인 수가 처음으로 나오는 위치가 된다.

2023 Summer Algorithm Camp

이분 탐색

- 9 이상인 자료가 처음으로 등장하는 위치

1	4	5	7	8	11	14
---	---	---	---	---	----	----

↑ ↑ ↑

- 첫 번째 탐색
 - $l=0, r=7$ (배열에서 가장 큰 자료보다 찾으려는 원소가 크면, 배열의 맨 뒤 인덱스 + 1을 탐색의 결과로 정의한다.)
 - $mid = (0+7)/2 = 3$
 - $arr[3] = 7 < 9$ 이므로 오른쪽으로 범위를 줄임
 - $l = 4$ 로 변경

1	4	5	7	8	11	14
---	---	---	---	---	----	----

 ↑ ↑ ↑

2023 Summer Algorithm Camp

이분 탐색

- 9 이상인 자료가 처음으로 등장하는 위치

1	4	5	7	8	11	14
---	---	---	---	---	----	----

↑ ↑ ↑

- 두 번째 탐색
 - $l=4, r=7$
 - $mid = (4+7)/2 = 5$
 - $arr[5] = 11 > 9$ 이므로 왼쪽으로 범위를 줄임
 - $r = 5$ 로 변경

1	4	5	7	8	11	14
---	---	---	---	---	----	----

↑ ↑

2023 Summer Algorithm Camp

이분 탐색

- 9 이상인 자료가 처음으로 등장하는 위치

1	4	5	7	8	11	14
---	---	---	---	---	----	----

 ↑ ↑

- 세 번째 탐색
 - $l=4, r=5$
 - $mid = (4+5)/2 = 4$
 - $arr[4] = 8 < 9$ 이므로 오른쪽으로 범위를 줄임
 - $l = 5$ 로 변경

1	4	5	7	8	11	14
---	---	---	---	---	----	----

 ↑

2023 Summer Algorithm Camp

이분 탐색

- 9 이상인 자료가 처음으로 등장하는 위치

1	4	5	7	8	11	14
---	---	---	---	---	----	----

↑

- $l = r = 5$ 로 탐색이 끝남.
- 답은 5

2023 Summer Algorithm Camp

이분 탐색 - lower_bound

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    vector<int> arr = {1, 4, 5, 6, 8, 11, 14};
    int x = 9; // 9 이상인 원소가 처음으로 등장하는 위치
    int l = 0, r = 7;
    while (l < r) {
        int mid = (l + r) / 2;
        if (arr[mid] < x) l = mid + 1;
        else r = mid;
    }
    cout << l; // 5
}
```

2023 Summer Algorithm Camp

이분 탐색 - upper_bound

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    vector<int> arr = {1, 4, 5, 6, 8, 8, 11, 14};
    int x = 8; // 8 초과인 원소가 처음으로 등장하는 위치
    int l = 0, r = 8;
    while (l < r) {
        int mid = (l + r) / 2;
        if (arr[mid] <= x) l = mid + 1; // <를 <=로 변경하면 됨
        else r = mid;
    }
    cout << l; // 6
}
```

질문?

이분 탐색 STL - binary_search

```
#include <algorithm>
```

```
bool std::binary_search(first, last, const T& value);
```

- 구간 [first, last)에 value가 있으면 true, 없으면 false를 반환한다.
- 구간 [first, last)는 반드시 **오름차순**으로 정렬되어 있어야 한다.
- 시간 복잡도
 - 임의 인덱스 접근(Random Access)이 되는 자료구조(std::vector 등)는 $O(\log N)$
 - 그렇지 않은 자료구조(std::set 등)는 $O(N)$

2023 Summer Algorithm Camp

이분 탐색 STL - binary_search

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    vector<int> arr = {1, 4, 5, 6, 8, 11, 14};
    cout << binary_search(arr.begin(), arr.end(), 6) << "\n"; // 1
    cout << binary_search(arr.begin(), arr.end(), 7) << "\n"; // 0

    cout << binary_search(arr.begin(), arr.begin() + 1, 4) << "\n"; // 0
    cout << binary_search(arr.begin(), arr.begin() + 2, 4) << "\n"; // 1
}
```

이분 탐색 STL - lower_bound

```
#include <algorithm>
```

```
iterator std::lower_bound(first, last, const T& value);
```

- 구간 $[first, last)$ 에 value보다 크거나 같은 원소 중 가장 앞에 있는 원소의 위치(iterator)를 반환한다.
 - 그러한 원소가 없다면, last를 반환한다.
- 구간 $[first, last)$ 는 반드시 **오름차순**으로 정렬되어 있어야 한다.
- 시간 복잡도
 - 임의 인덱스 접근(Random Access)이 되는 자료구조(std::vector 등)는 $O(\log N)$
 - 그렇지 않은 자료구조(std::set 등)는 $O(N)$

2023 Summer Algorithm Camp

이분 탐색 STL - lower_bound

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    vector<int> arr = {1, 4, 5, 6, 8, 11, 14};

    auto it = lower_bound(arr.begin(), arr.end(), 4);
    cout << *it << " " << it - arr.begin() << "\n"; // 4 1

    it = lower_bound(arr.begin(), arr.end(), 7);
    cout << *it << " " << it - arr.begin() << "\n"; // 8 4

    it = lower_bound(arr.begin(), arr.end(), 15);
    cout << (it == arr.end()) << " " << it - arr.begin() << "\n"; // 1 7
    // *it으로 값에 접근하려 하면 Runtime Error가 발생할 수 있음
}
```

이분 탐색 STL - upper_bound

```
#include <algorithm>
```

```
iterator std::upper_bound(first, last, const T& value);
```

- 구간 $[first, last)$ 에 value보다 큰 원소 중 가장 앞에 있는 원소의 위치(iterator)를 반환한다.
 - 그러한 원소가 없다면, last를 반환한다.
- 구간 $[first, last)$ 는 반드시 **오름차순**으로 정렬되어 있어야 한다.
- 시간 복잡도
 - 임의 인덱스 접근(Random Access)이 되는 자료구조(std::vector 등)는 $O(\log N)$
 - 그렇지 않은 자료구조(std::set 등)는 $O(N)$

2023 Summer Algorithm Camp

이분 탐색 STL - upper_bound

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    vector<int> arr = {1, 4, 5, 5, 5, 6, 8, 11, 14};

    auto it = upper_bound(arr.begin(), arr.end(), 2);
    cout << *it << " " << it - arr.begin() << "\n"; // 4 1

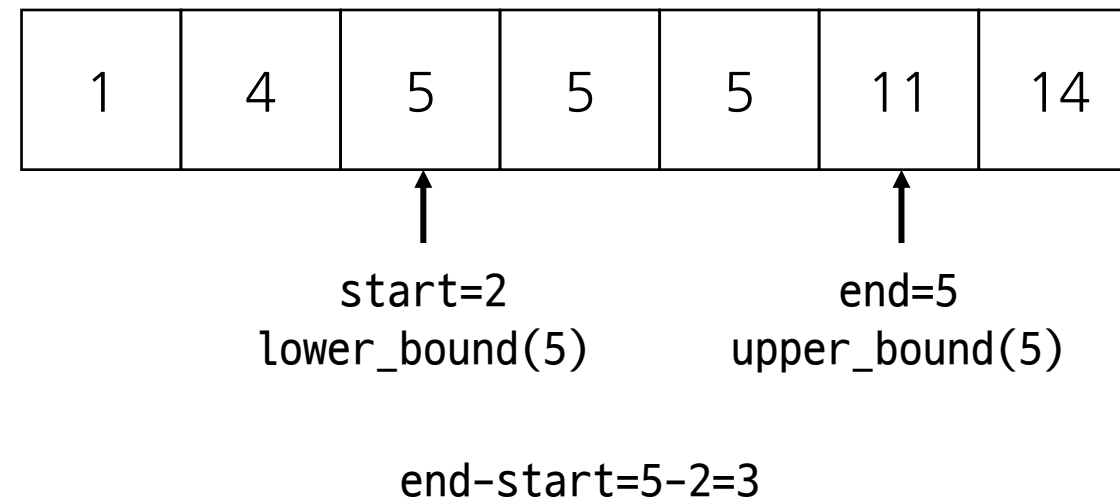
    it = upper_bound(arr.begin(), arr.end(), 5);
    cout << *it << " " << it - arr.begin() << "\n"; // 6 5
}
```

질문?

2023 Summer Algorithm Camp

이분 탐색 STL

- 정렬된 배열에서 특정한 수의 개수를 `upper_bound`와 `lower_bound`를 사용해서 구할 수 있다.
- 오름차순으로 정렬된 배열에서 같은 수끼리 서로 인접한다.
- 개수를 구하려는 수를 `x`라고 하자.
 - `x`에 대한 `lower_bound`를 하면 배열에서 `x`의 시작 위치(`start`)를 구할 수 있다.
 - `x`에 대한 `upper_bound`를 하면 배열에서 `x`의 마지막 위치에서 바로 다음 위치(`end`)를 구할 수 있다.
- $\text{end} - \text{start}$ 가 `x`의 개수가 된다.



2023 Summer Algorithm Camp

이분 탐색 STL

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    vector<int> arr = {1, 4, 5, 5, 5, 6, 8, 11, 14};

    cout << upper_bound(arr.begin(), arr.end(), 5) - lower_bound(arr.begin(), arr.end(), 5); // 3
}
```

질문?

2023 Summer Algorithm Camp

파라메트릭 서치 (Parametric Search, 매개 변수 탐색)

- 최적화 문제
 - 가능한 답 중에서 최솟값 또는 최댓값을 구하는 문제
- 결정 문제
 - 네(Yes/Y) 또는 아니오(No/N)로 답할 수 있는 문제
- 최적화 문제를 결정 문제로 바꾸어 푸는 기법
 - 특정 문제 상황(최적화 문제)에서 답을 직접 구하는 것이 아니라,
 - “이게 답이 될 수 있는가? [Y/N]”(결정 문제)를 반복 확인하여 최솟값 또는 최댓값을 구함.
- 결정 문제에 대한 답이 **“단조성이 없어도”** 파라메트릭 서치임.
- 그러나, 결정 문제에 대한 답이 다음과 같이 **단조성을 가지는 경우만 이분 탐색**으로 해결하는 방법을 다룬다.
 - YY...YYNN...NN
 - NN...NNYY...YY
 - 계속 Yes가 나오다가 그 이후로 꼭 No가 나오거나, 그 반대여야 한다.

2023 Summer Algorithm Camp

예시 문제

- [BOJ 15810](#) (풍선 공장)
- N명의 사람이 있고, 각각 풍선 하나를 만드는 데 A_1, A_2, \dots, A_N 분이 걸림
 - 풍선 하나 만드는 데 3분이 걸리는 사람은 3분, 6분, 9분, ...이 될 때마다 하나를 만듦.
- 풍선을 총 M개 만드는 데 걸리는 시간을 구하는 문제
- 최적화 문제
 - `optimization()` : 풍선 M개를 만드는 데 걸리는 최소 시간

2023 Summer Algorithm Camp

예시 문제

- 최적화 문제를 $O((M + N)\log N)$ 에 해결할 수 있는 풀이가 있으나, 현재까지 배운 내용으로는 최적화 문제를 해결할 수 없다.
 - 8회차 트리/그래프 때 다시 보도록 합시다.
- 파라메트릭 서치를 이용하여 접근해 보자.
- 최적화 문제
 - optimization() : 풍선 M개를 만드는 데 걸리는 최소 시간
- 결정 문제
 - decision(T) : 시간이 T분일 때 풍선을 M개 이상 만들 수 있는가?

2023 Summer Algorithm Camp

예시 문제

- 결정 함수의 매개 변수 T 가 증가함에 따라, $\text{decision}(T)$ 의 값은 아래와 같다.
 - $\text{NNNN}\cdots\text{NNN}\textcolor{red}{Y}\text{YY}\cdots\text{Y}$
 - 처음으로 등장하는 Y 의 위치는 **이분 탐색**으로 찾을 수 있다.
- 이분 탐색의 최대 범위 계산
 - 스템프의 최소화 : $N = 1, A_1 = 10^6$
 - 풍선의 최대화 : $M = 10^6$
 - 답은 10^{12} 로 이 경우가 최대
- $l = 1, r = 10^{12}$ 범위에 대하여 이분 탐색을 하면 된다.
 - $\text{decision}(\text{mid})$ 가 Y 면, $r = \text{mid}$
 - mid 는 답이 되기 때문에, mid 를 포함하면서 왼쪽으로 구간을 줄이면 됨
 - $\text{decision}(\text{mid})$ 가 N 이면, $l = \text{mid} + 1$
 - mid 는 답이 되지 않기 때문에, mid 는 포함하지 않으면서 오른쪽으로 구간을 줄이면 됨

2023 Summer Algorithm Camp

예시 문제

- decision(T) 함수 설계
 - 시간이 T분 있으면, i번째 스태프가 만들 수 있는 풍선의 수는 $\left\lfloor \frac{A_i}{T} \right\rfloor$ 이다.
 - 만들 수 있는 풍선의 개수가 M보다 크거나 같으면 Y, 그렇지 않으면 N이다.
- 시간 복잡도는 O(N)
- 전체 문제 시간 복잡도 계산
 - decision(T) 함수를 이분 탐색을 하는 과정마다 호출함.
 - 이분 탐색은 $\log_2(10^{12})$ 번
 - $O(\log(10^{12}) * N)$

2023 Summer Algorithm Camp

예시 문제

```
#include <bits/stdc++.h>

using namespace std;

int n, m;
vector<int> arr;

bool decision(long long T) {
    long long cnt = 0;
    for (int i : arr) cnt += T / i;
    return cnt >= m;
}

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

    cin >> n >> m;
    arr.resize(n);
    for (int &i : arr) cin >> i;
    long long l = 1, r = 1e12;
    while (l < r) {
        long long mid = (l + r) / 2;
        if (decision(mid)) r = mid;
        else l = mid + 1;
    }
    cout << l;
}
```

질문?

2023 Summer Algorithm Camp

파라메트릭 서치

- 답의 형태에 따른 이분 탐색 범위 지정
- $NNN \cdots NNNYYY \cdots YYY$ (방금 예시 문제와 동일함)
 - 처음으로 Y가 등장하는 위치 찾기 (최소화 문제)
 - $\text{decision}(\text{mid})$ 가 Y면, $r = \text{mid}$
 - mid는 답이 되기 때문에, mid를 포함하면서 왼쪽으로 구간을 줄이면 됨
 - $\text{decision}(\text{mid})$ 가 N이면, $l = \text{mid} + 1$
 - mid는 답이 되지 않기 때문에, mid는 포함하지 않으면서 오른쪽으로 구간을 줄이면 됨
- $YYY \cdots YYN \cdots NNN$
 - 마지막으로 Y가 등장하는 위치 찾기 (최대화 문제)
 - $\text{decision}(\text{mid})$ 가 Y면, $l = \text{mid}$
 - mid는 답이 되기 때문에, mid를 포함하면서 오른쪽으로 구간을 줄이면 됨
 - $\text{decision}(\text{mid})$ 가 N이면, $r = \text{mid} - 1$
 - mid는 답이 되지 않기 때문에, mid는 포함하지 않으면서 오른쪽으로 구간을 줄이면 됨
 - 이 경우, $\text{mid} = \left\lfloor \frac{l+r}{2} \right\rfloor$ 이어야 한다.
 - 구현할 때는 $\text{mid} = (l + r + 1) / 2$

2023 Summer Algorithm Camp

파라메트릭 서치

- 답의 형태에 따른 이분 탐색 범위 지정
- 외워서 짜기보다는 `decision(mid)`의 Y/N 값 여부에 따라서 생각하는 것이 편하다.
- `decision(mid)`의 Y/N 값 여부
 - Y면, mid가 답이 될 수 있으므로 mid를 포함하도록 범위를 줄임
 - N이면, mid가 답이 될 수 없으므로 mid를 포함하지 않도록 범위를 줄임

질문?

2023 Summer Algorithm Camp

분할 정복

- 큰 문제를 더 쉬운(작은) 문제로 **분할**하고
 - **정복** : 쪼개지지 않을 때까지(답을 직접 구할 수 있을 때까지) 계속 분할하여 직접 답을 계산한다.
- 부분 문제의 답을 **결합**하여 큰 문제의 답을 구하는 알고리즘

- 보통 재귀 함수로 구현하면 편리함
 - 분할 : 재귀 함수를 다시 호출
 - 정복 : 재귀 함수의 종료 조건 (base case)
 - 결합 : 재귀 함수의 결과값을 바탕으로 답을 계산

- 예시
 - 정렬 : 병합 정렬, 퀵 정렬
 - 탐색 : 이분 탐색

분할 정복 - 예시

- 병합 정렬
 - 분할 : 절반 크기인 두 배열로 나눈다.
 - 정복 : 배열의 크기가 0 또는 1이면 이미 정렬된 배열(문제를 해결한 상태)이다.
 - 결합 : 정렬된 두 배열을 잘 합쳐 정렬된 배열을 만든다.
- 퀵 정렬
 - 분할 : pivot을 기준으로 pivot 이하인 배열과 pivot 초과인 배열, 두 배열로 나눈다.
 - 정복 : 배열의 크기가 0 또는 1이면 이미 정렬된 배열(문제를 해결한 상태)이다.
 - 결합 : “[pivot 이하인 원소의 정렬된 배열] pivot [pivot 초과인 원소의 정렬된 배열]” 순서로 합치면 정렬된 배열이 된다.

분할 정복 - 예시

- 이분 탐색 (lower_bound)
 - 분할 : mid를 사용해 두 구간 $[l, mid]$, $[mid + 1, r]$ 중 어떤 구간에 답이 있는지 알아낸다.
 - 정복 : 구간 $[l, r]$ 의 크기가 1이면 답은 l이다.
 - 결합 : 하나의 구간에서 얻은 답은 전체 구간에서의 답과 같다.

2023 Summer Algorithm Camp

분할 정복 - 재귀 함수 (이분 탐색)

```
#include <bits/stdc++.h>

using namespace std;

int lower_bound(const vector<int> &arr, int value, int start, int end) {
    if (start == end) return start; // 정복 완료
    int mid = (start + end) / 2;
    if (arr[mid] >= value) return lower_bound(arr, value, start, mid); // find answer in [start, mid]
    else return lower_bound(arr, value, mid + 1, end); // find answer in [mid + 1, end]
}

int main() {
    vector<int> arr = {1, 4, 5, 6, 8, 11, 14};
    cout << lower_bound(arr, 7, 0, arr.size()) << "\n"; // 4
    cout << lower_bound(arr, 14, 0, arr.size()) << "\n"; // 6
    cout << lower_bound(arr, 15, 0, arr.size()); // 7
}
```

질문?

2023 Summer Algorithm Camp

분할 정복 응용 - 빠른 거듭제곱

- 정수 a , n 에 대하여 a^n 을 계산하는 방법
- 단순히 반복문으로 $a * a * \dots * a$ 를 하면 n 번 곱하여야 하므로 $O(n)$ 이 걸린다.
- 우선, 재귀 함수를 아래와 같이 정의해 보자.
- $F(a, n) = a^n$
 - base case) $n = 0$ 이면, 1을 반환한다.
 - general case) $n > 0$ 이면, $F(a, n - 1) * a$ 를 반환한다.
- 아직까지는 $O(n)$ 시간이 걸린다.

분할 정복 응용 - 빠른 거듭제곱

- 최적화할 수 있을까?
- n 의 홀짝성
 - n 이 짝수라면, 지수 법칙에 의해 $a^n = a^{n/2} * a^{n/2}$
 - n 이 홀수라면, 비슷하게 $a^n = a^{\lfloor n/2 \rfloor} * a^{\lfloor n/2 \rfloor} * a$
- $a^{\lfloor n/2 \rfloor}$ 을 구해두고 값을 제공하면 a^n 을 바로 구할 수 있다.
- 즉, 재귀 함수의 general case를 다음과 같이 바꿀 수 있다.
 - n 이 짝수, $F(n) = F(n / 2)^2$
 - n 이 홀수, $F(n) = F(n / 2)^2 * a$
- 시간 복잡도는 $O(\log N)$ 이다.
 - n 이 계속 절반씩 감소하기 때문.

2023 Summer Algorithm Camp

분할 정복 응용 - 빠른 거듭제곱

- 2^{10} 을 계산한다고 해보자.
- $n=10$
 - $2^5 * 2^5$
 - $= 1024$
- $n = 5$
 - $2^2 * 2^2 * 2$
 - $= 32$
- $n = 2$
 - $2^1 * 2^1$
 - $= 4$
- $n = 1$
 - $2^0 * 2^0 * 2$
 - $= 2$
- $n = 0$
 - return 1

2023 Summer Algorithm Camp

분할 정복 응용 - 빠른 거듭제곱

```
#include <bits/stdc++.h>

using namespace std;

int fast_pow(int a, int n) {
    if (!n) return 1;
    int ret = fast_pow(a, n / 2);
    ret *= ret;
    if (n % 2) ret *= a;
    return ret;
}

int main() {
    cout << fast_pow(2, 10); // 1024
}
```


질문?

2023 Summer Algorithm Camp

예시 문제

- [BOJ 1629](#) (곱셈)
- $A^B \pmod C$ 를 구하는 문제 ($1 \leq A, B, C \leq 2^{31} - 1$)
- 먼저 오버플로우를 고려해야 한다.
 - 우선 long long 자료형을 사용한다. ($2^{63} - 1 \approx 9 * 10^{18}$ 까지 저장할 수 있음)
 - 그래야 $a * a$ 를 했을 때 연산 결과를 온전하게 저장할 수 있다.
- 합동식의 성질에 따라 곱할 때마다 mod C를 해줘도 연산의 결과는 같고 long long 범위에서 오버플로우는 나지 않음.
- 방금 배운 빠른 거듭제곱을 그대로 구현하되 mod C를 해주면 해결할 수 있다.

2023 Summer Algorithm Camp

예시 문제

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

ll fast_pow(ll base, ll exp, ll mod) {
    if (!exp) return 1;
    ll ret = fast_pow(base, exp / 2, mod);
    ret *= ret;
    ret %= mod;
    if (exp % 2) ret *= base, ret %= mod;
    return ret;
}

int main() {
    ll a, b, c; cin >> a >> b >> c;
    cout << fast_pow(a, b, c);
}
```

질문?

2023 Summer Algorithm Camp

문제

- 필수 문제
- [BOJ 1920](#) (수 찾기)
- [BOJ 2417](#) (정수 제곱근)
- [BOJ 26258](#) (다중 일차 함수)
- [BOJ 15810](#) (풍선 공장)
- [BOJ 1654](#) (랜선 자르기)

- 연습/심화 문제
- [BOJ 2805](#) (나무 자르기)
- [BOJ 2110](#) (공유기 설치)
- [BOJ 1300](#) (K번째 수)
- [BOJ 11819](#) (The Shortest does not Mean the Simplest)
- [BOJ 4233](#) (가짜소수)