Binary Search

시작하기 전에: 한번쯤은 해봤을 게임

- 업 다운 게임
- 1부터 100까지의 수 중에서 원하는 수를 기록
- 어떤 수를 기록했는지 맞추는 게임
- 틀렸을 때에는, 외친 수보다 기록한 수가 더 작은지, 더 큰지 말해 준다
- 정답을 낮은 시도로 맞춰야 한다

업 다운 게임

- 1부터 100까지의 수 중에서 고른다고 하면, 어떤 수를 고르는 게 최선의 방법일까?
- 10을 골라서 도박수를 노린다?
- 내가 좋아하는 수를 불러서 행운을 빈다?

업 다운 게임

- 1부터 100까지의 수 중에서 고른다고 하면, 어떤 수를 고르는 게 최선의 방법일까?
- 10을 골라서 도박수를 노린다?
- 내가 좋아하는 수를 불러서 행운을 빈다?
- 항상 구간의 절반을 후보에서 탈락시킬 수 있도록 가운데 수를 부른다
- 이 경우의 시간 복잡도는 어떻게 될까?

업 다운 게임

- 1부터 100까지의 수 중에서 고른다고 하면, 어떤 수를 고르는 게 최선의 방법일까?
- 10을 골라서 도박수를 노린다?
- 내가 좋아하는 수를 불러서 행운을 빈다?
- 항상 구간의 절반을 후보에서 탈락시킬 수 있도록 가운데 수를 부른다
- 이 경우의 시간 복잡도는 어떻게 될까?
- $O(logN)$

Binary Search

- 이분 탐색
- 정렬돼 있는 자료들에서 탐색 범위를 절반씩 줄여가면서 특정 원소를 찾는 기법
- C++ STL `lower_bound, upper_bound`
- lower_bound는 $A[i-1] < x \leq A[i]$인 i를 구해 준다
- upper_bound는 $A[i-1] \leq x < A[i]$인 i를 구해 준다

Binary Search

- 이분 탐색
- 정렬돼 있는 자료들에서 탐색 범위를 절반씩 줄여가면서 특정 원소를 찾는 기법
- Yes/No로 대답하는 결정 문제에서, 답이 두 구간으로 나뉘어질 때 사용할 수 있다
- 예) 정렬되어진 수열에서, 이 수는 **10 이상**인가? 처음으로 10 이상인 수는 어떤 수인가?

N	N	N	N	Y	Y	Y	Y	Y	Y
1	2	4	7	11	13	15	17	18	19

Binary Search

- 처음으로 10보다 같거나 큰 수를 찾으려면 정답이 바뀌는 순간 $(\mathrm{N} \rightarrow Y)$을 알아야 한다
- 즉, 처음으로 $\boldsymbol{10 \leq A[i]}$**인** $\boldsymbol{i}$를 찾아야 한다
- 처음 찾을 구간 내에 바뀌는 순간이 있으므로, 구간의 **처음과 끝의 답은 달라야 한다**
- 바뀌는 순간을 구간에 포함한 채로, 반씩 구간을 줄여나가면 빠르게 구할 수 있다

lo									hi
N	N	N	N	Y	Y	Y	Y	Y	Y
1	2	4	7	11	13	15	17	18	19

Binary Search

- 최초의 lo, hi에서, $A[lo], A[hi]$의 결정 문제에 대한 **답이 다르도록 구간을 설정**한다
- $lo + 1 < hi$인 동안, $mid = (lo + hi)/2$ 로 절반을 잡은 뒤 **버릴 구간을 찾는다**
- 바뀌는 구간을 찾아야 하므로, $A[mid]$와 $A[hi]$의 답이 같다면, hi를 mid로 만든다
- 반대의 경우도 마찬가지, $A[lo]$와 $A[hi]$의 답이 같다면, lo를 mid로 만든다

lo				mid					hi
N	N	N	N	Y	Y	Y	Y	Y	Y
1	2	4	7	11	13	15	17	18	19

Binary Search

- 반복문이 깨질 때까지, $A[lo]$와 $A[hi]$**의 답은 변하지 않는다**
- 바뀌는 구간을 찾아야 하므로, $A[mid]$와 $A[hi]$의 답이 같다면, hi를 mid로 만든다
- 반대의 경우도 마찬가지, $A[lo]$와 $A[hi]$의 답이 같다면, lo를 mid로 만든다
- 구간을 조정하는 과정에서, **조정하는 두 부분의 답은 서로 같다**

lo		mid		hi					
N	N	N	N	Y	Y	Y	Y	Y	Y
1	2	4	7	11	13	15	17	18	19

Binary Search

- $lo + 1 < hi$이므로, lo와 hi 사이에는 한 칸이 반드시 존재한다
- 반복문을 탈출했을 때에는 $lo < hi$이면서 $lo + 1 \geq hi$인 경우, 즉 $lo + 1 = hi$
- 즉 결정 문제의 답이 바뀌는 구간에 lo와 hi가 각각 존재
- lower bound

			lo	hi					
N	N	N	N	Y	Y	Y	Y	Y	Y
1	2	4	7	11	13	15	17	18	19

Binary Search

```
void lower_bound(int x) {
    int lo = -1, hi = N;
    while (lo + 1 < hi) {
        int mid = (lo + hi) / 2;
        if (arr[mid] < x) lo = mid;
        else hi = mid;
    }
    return hi;
}
```

```
int idx = lower_bound(v.begin(), v.end(), x) - v.begin();
```

			lo	hi					
N	N	N	N	Y	Y	Y	Y	Y	Y
1	2	4	7	11	13	15	17	18	19

수 찾기 BOJ 1920

- 두 개의 배열 A, B가 주어질 때, 각각의 B의 원소에 대해서 A에 존재하는지 확인하자
- Naïve : 주어진 대로 구현하기 $O(NM)$
- $N, M \leq 100\ 000$
- $A_i \leq 100\ 000$이라면? 체크 배열을 활용할 수 있음
- 문제에서는 $A_i < 2^{31}$

수 찾기 BOJ 1920

- 두 개의 배열 A, B가 주어질 때, 각각의 B의 원소에 대해서 A에 존재하는지 확인하자
- 배열 A를 정렬한 뒤, B의 각 원소에 대해서 이분 탐색을 통해 원소의 존재 여부를 탐색
- 이 경우의 시간복잡도는 $O(NlogN + MlogN)$
- A는 정렬되지 않았지만, 단순하게 풀이하는 방식이 정렬하는 방식보다 한참 오래 걸림
- 정렬한 뒤 빠르게 풀 수 있으므로 배열을 정렬

정수 제곱근 BOJ 2417

- 정수가 주어졌을 때, 해당 수의 정수 제곱근을 구하자
- n이 주어지면, $q^2 \geq n$인 음이 아닌 정수 q를 구하기

정수 제곱근 BOJ 2417

- 정수가 주어졌을 때, 해당 수의 정수 제곱근을 구하자
- n이 주어지면, $q^2 \geq n$인 음이 아닌 정수 q를 구하기
- 17의 정수 제곱근은 5
- 특정 구간부터 위 식이 성립하고, 위 식이 성립하는 첫 번째 구간을 구해야 함
- 범위가 매우 크므로, unsigned long long 등을 활용

최적화 문제와 결정 문제

- 결정 문제는 Yes / No로 대답할 수 있는 문제
- 최적화 문제는 답을 정수/실수 등으로 나타내는 문제 (최댓값, 최솟값 등)

- 어떤 문제가 더 풀기 어려울까?

최적화 문제와 결정 문제

- 최적화 문제를 결정 문제로 나타낸다면 빠르게 풀 수 있지 않을까?

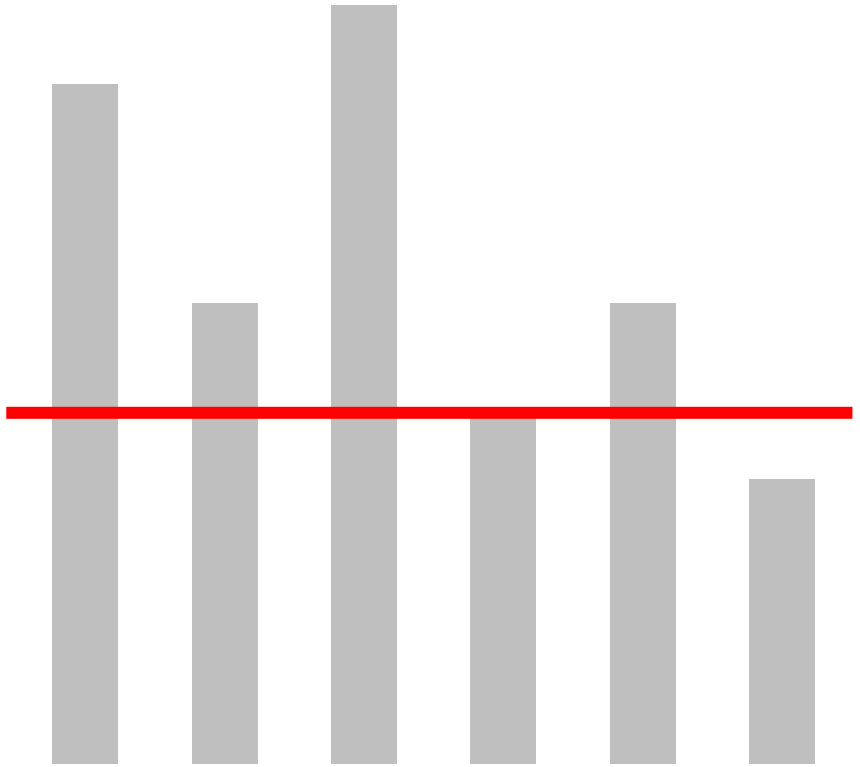
- 최적화 문제: 배열에서 최댓값을 구하자
- 결정 문제: 구간에서 x 이상의 값이 존재하는가?
- x를 1, 2, 3, ... 늘려가다 보면 결정 문제의 답이 No가 되는 순간이 온다
- 이를 이분 탐색으로 구할 수 있다

Parametric Search

- 매개변수 탐색
- 최적화 문제를 결정 문제로 환원해서 이분 탐색을 활용해 문제를 푸는 기법
- 결정 문제를 해결하는 데 얼마나 걸리는지 확인
- 이분탐색을 활용할 구간이 얼마나 큰지 확인
- 두 개의 시간복잡도를 곱해 최종 시간복잡도를 합산
- 정렬할 경우, 해당 시간복잡도도 계산

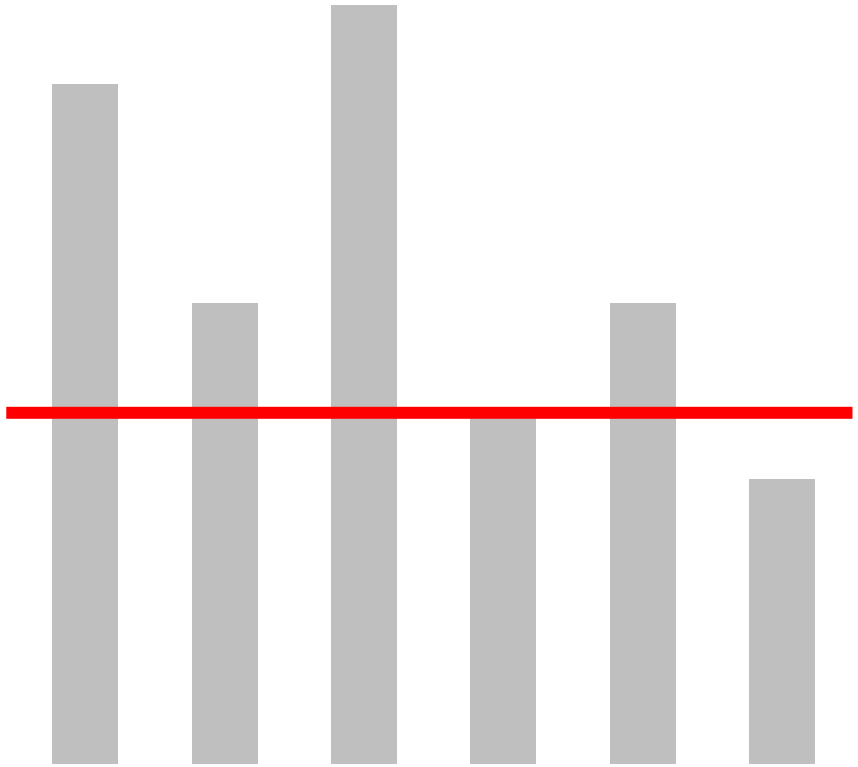
나무 자르기 BOJ 2805

- 톱을 특정 높이에 두고, 해당 높이보다 큰 부분을 다 잘라간다
- 잘라간 부분이 M 이상이어야 한다
- **톱의 높이의 최댓값**은?



나무 자르기 BOJ 2805

- 톱을 특정 높이에 두고, 해당 높이보다 큰 부분을 다 잘라간다
- 잘라간 부분이 M 이상이어야 한다
- 톱의 높이의 최댓값은?
- **톱의 높이가 x일 때, M 이상을 가져갈 수 있나?**
- 위 결정 문제를 해결하는 데 $O(N)$



나무 자르기 BOJ 2805

- **톱의 높이가 x일 때, M 이상을 가져갈 수 있나?**
- x가 정해졌다면?
- 모든 나무를 확인하면서 높이가 x 이상인 나무에 대해서 $h[i] - x$의 합을 계산
- 높이의 구간은 $[0, 2\ 000\ 000\ 000]$, 결정 문제에 대한 해답은 $YY..YNN$ 꼴
- 마지막으로 위치하는 Y의 위치를 찾아야 한다 (lo)

킥보드로 등교하기 BOJ 27977

- 건덕이의 위치는 0, 학교의 위치는 L, 킥보드를 최대 K번 충전할 수 있음
- 충전소의 위치가 주어질 때, 학교로 도착할 수 있는 **가장 작은 배터리 용량**을 구하자
- 최적화 문제

킥보드로 등교하기 BOJ 27977

- 건덕이의 위치는 0, 학교의 위치는 L, 킥보드를 최대 K번 충전할 수 있음
- 충전소의 위치가 주어질 때, 학교로 도착할 수 있는 **가장 작은 배터리 용량**을 구하자

- 결정 문제로 바꿔서 풀 수 있을까?
- 바꾼다면, 해당 결정 문제를 해결하는 데 걸리는 시간복잡도는 어떻게 될까?

킥보드로 등교하기 BOJ 27977

- 건덕이의 위치는 0, 학교의 위치는 L, 킥보드를 최대 K번 충전할 수 있음
- 충전소의 위치가 주어질 때, 학교로 도착할 수 있는 **가장 작은 배터리 용량**을 구하자

- x의 용량을 가진 킥보드로 최대 K번 충전해서 L의 위치에 도착할 수 있을까?
- $O(N)$, N은 충전소의 개수

킥보드로 등교하기 BOJ 27977

- x의 용량을 가진 킥보드로 최대 K번 충전해서 L의 위치에 도착할 수 있을까?
- 배터리의 개수를 기준으로 $1, 2, \ldots, L$까지 확인할 수 있음
- 결정 문제에 대한 해답은 $NN..NYYYY$ 형태
- 최초로 가능한 지점을 찾아야 한다, 이분 탐색에서의 hi 찾기

공유기 설치 BOJ 2110

- N개의 집의 좌표가 주어지고, C개의 공유기를 설치해야 한다
- 단, 한 집에는 공유기를 하나만 설치
- 가장 인접한 공유기 사이의 거리를 최대화 (공유기 사이의 거리의 최솟값을 최대화)
- 이 때의 가장 인접한 공유기 사이의 거리는?

공유기 설치 BOJ 2110

- 최적화 문제는 어려우니 결정 문제로 환원해 보자
- 가장 인접한 두 공유기 사이의 거리의 최댓값을 구하자
- 가장 인접한 두 공유기 사이의 거리의 최댓값이 x가 되도록 공유기를 배치할 수 있는가?
- 결정 문제의 답은 $x = 0, 1, \ldots$ 이 됨에 따라 어떻게 변화할까?

Y	Y	Y	Y	N	N	N	N	N	N
1	2	3	4	5	6	7	8	9	10

공유기 설치 BOJ 2110

- 최적화 문제는 어려우니 결정 문제로 환원해 보자
- 결정 문제의 답은 $x = 0, 1, \ldots$ 이 됨에 따라 어떻게 변화할까?
- 반복문을 탈출하고 난 뒤, lo와 hi 중 어떤 게 우리가 원하는 정답일까?
- 반복문이 시작할 때의 범위 $[lo, hi]$는 어디로 설정해야 할까?

lo hi

Y	Y	Y	Y	N	N	N	N	N	N
1	2	3	4	5	6	7	8	9	10

공유기 설치 BOJ 2110

- 결정 문제의 답은 그리디하게 구할 수 있다
- 항상 가장 왼쪽 집에 공유기를 설치한 뒤, 이전 공유기와의 거리가 x보다 작은 경우에는 공유기를 놓지 않는다
- 해당 전략의 시간복잡도는 $O(N)$
- 위의 전략으로 모든 공유기를 놓았다면, lo를 mid로 옮겨도 된다
- 모든 공유기를 놓지 못했다면, hi를 mid로 내려야 한다

공유기 설치 BOJ 2110

- 정답의 상한은 10^9, 집의 개수는 $200\,000$개
- 결정 문제로 환원했을 때, 문제를 푸는 데 걸리는 시간은 $O(N)$
- 정답 범위는 $[1, 10^9]$이고 결정 문제의 정답에 따라 답이 절반씩 줄어든다
- 주어진 집의 좌표가 정렬돼있지 않으므로 정렬할 필요도 있음
- $\log_2 10^9 \simeq 30$
- $O(NlogN + N \times log \max(x_i)\ ) = O(NlogN)$

Longest Increasing Subsequence

- 가장 긴 증가하는 부분 수열
- 수열 A가 주어질 때, 원소의 순서를 바꾸지 않고 몇 개의 원소를 제거할 수 있다
- 제거하고 남은 수열: 부분 수열
- 증가하는 부분 수열 중에서, 가장 긴 증가하는 부분수열의 길이 찾기

10	20	10	30	20	50

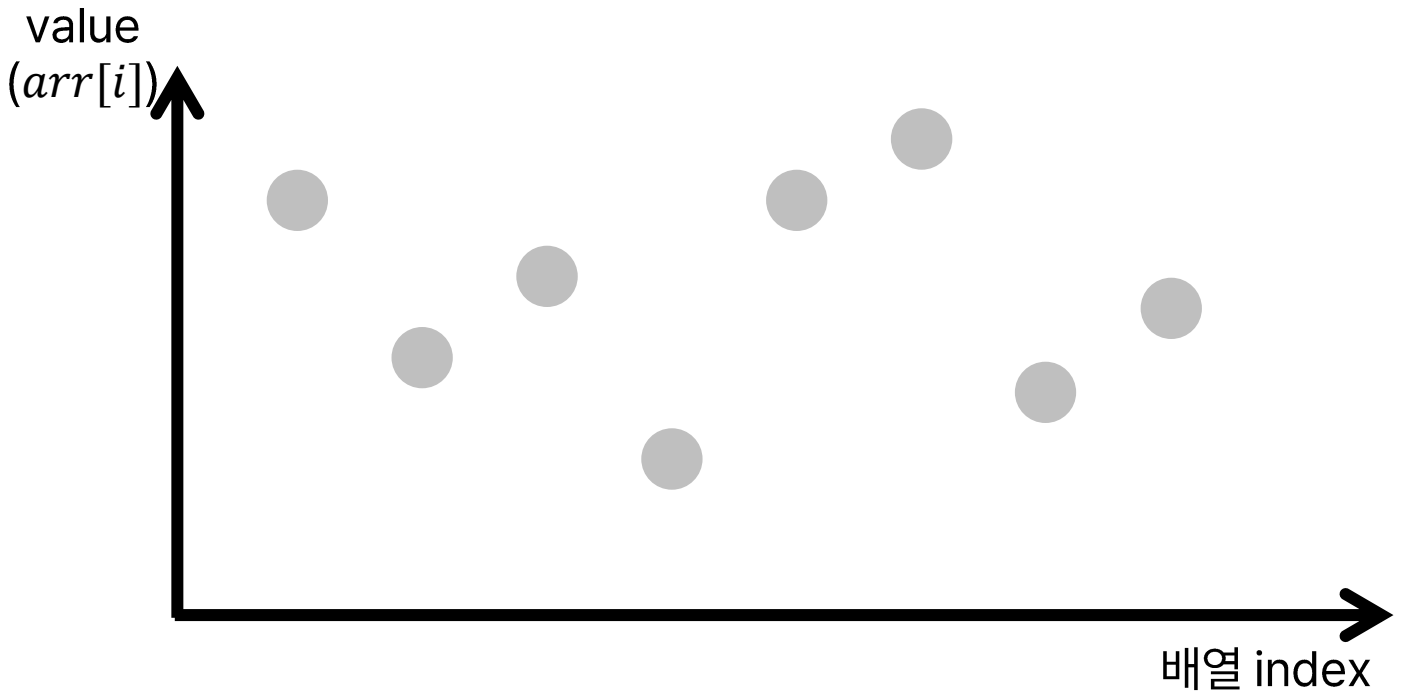
Longest Increasing Subsequence

- Naïve solution: 모든 부분 수열을 찾고, 해당 부분 수열이 증가하는지 확인한다
- 모든 원소에 대해서 넣거나, 넣지 않거나 두 가지 상태
- $O(2^N \times N)$

10	20	10	30	20	50

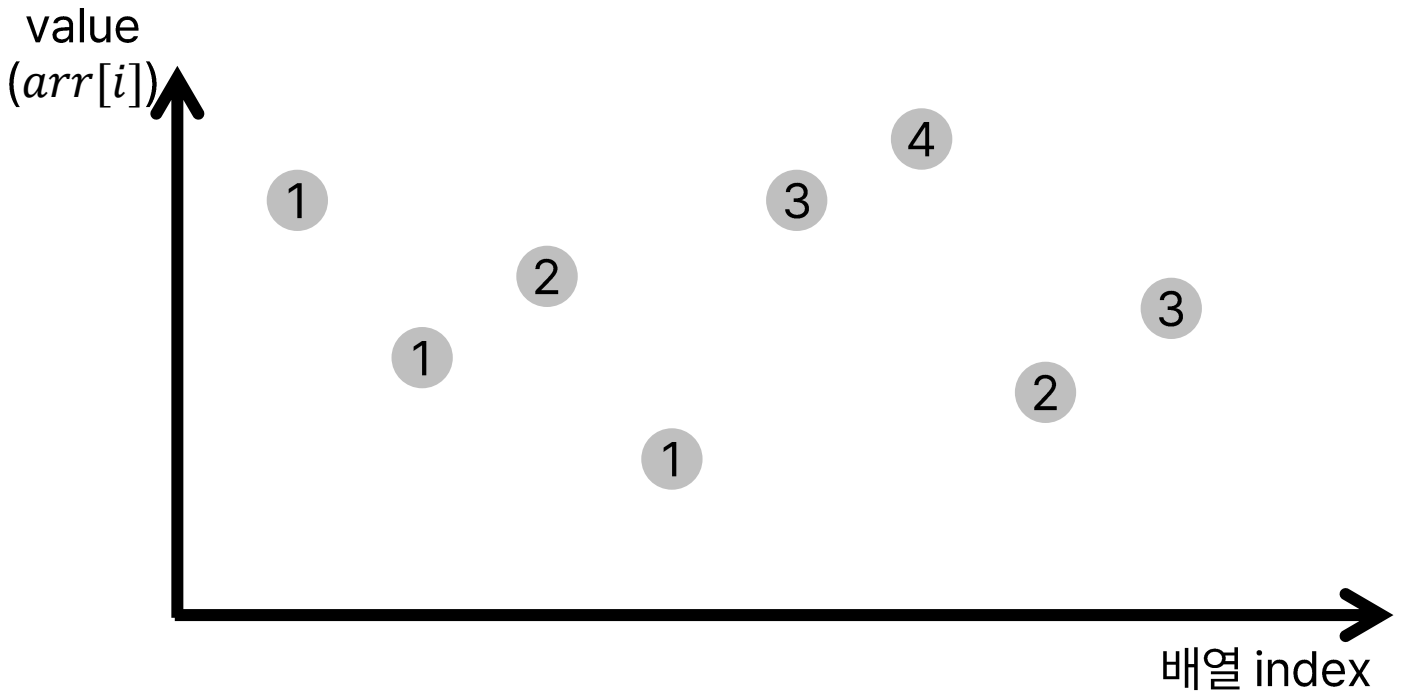
Longest Increasing Subsequence

- Better solution: $d[i]$ = i번째 인덱스의 값으로 끝나는 부분 수열의 최대 길이



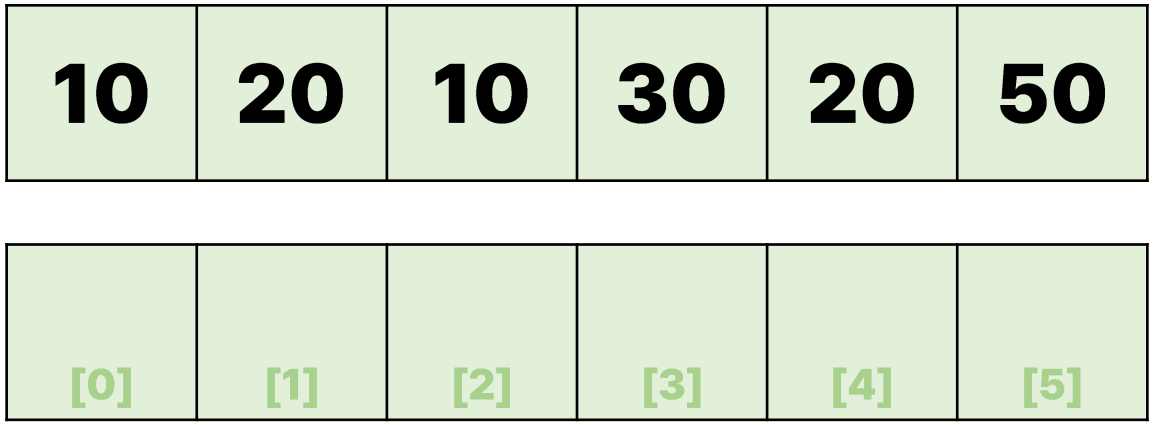
Longest Increasing Subsequence

- Better solution: $d[i]$ = i번째 인덱스의 값으로 끝나는 부분 수열의 최대 길이



Longest Increasing Subsequence

- Better solution: $d[i]$ = i번째 인덱스의 값으로 끝나는 부분 수열의 최대 길이
- 그래프에서 자신보다 앞에 있는 점을 모두 기억하고 있어야 함



Longest Increasing Subsequence

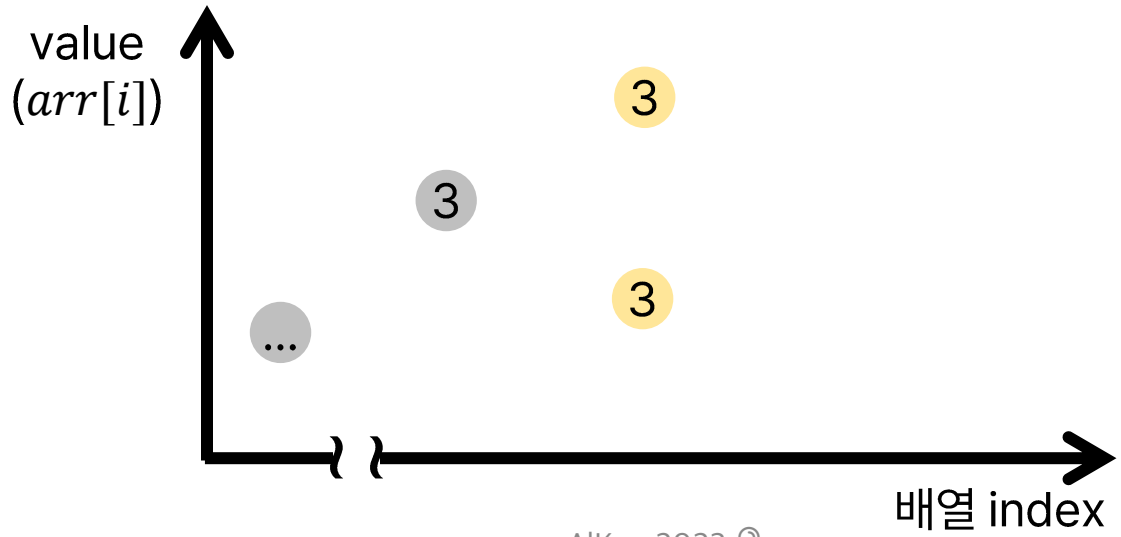
- Better solution: $d[i]$ = i번째 인덱스의 값으로 끝나는 부분 수열의 최대 길이
- $j < i$인 j에 대해서, $arr[j] < arr[i]$인 경우 $d[j] + 1$
- 해당 값의 최댓값을 저장하면 된다, $O(N^2)$

10	20	10	30	20	50

1	2	1	3	2	4

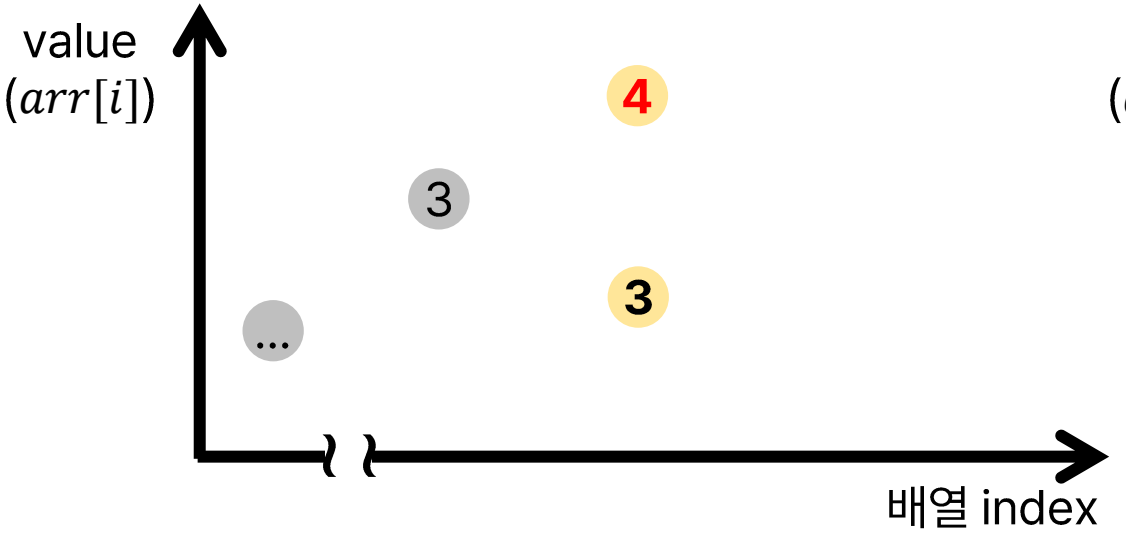
Longest Increasing Subsequence

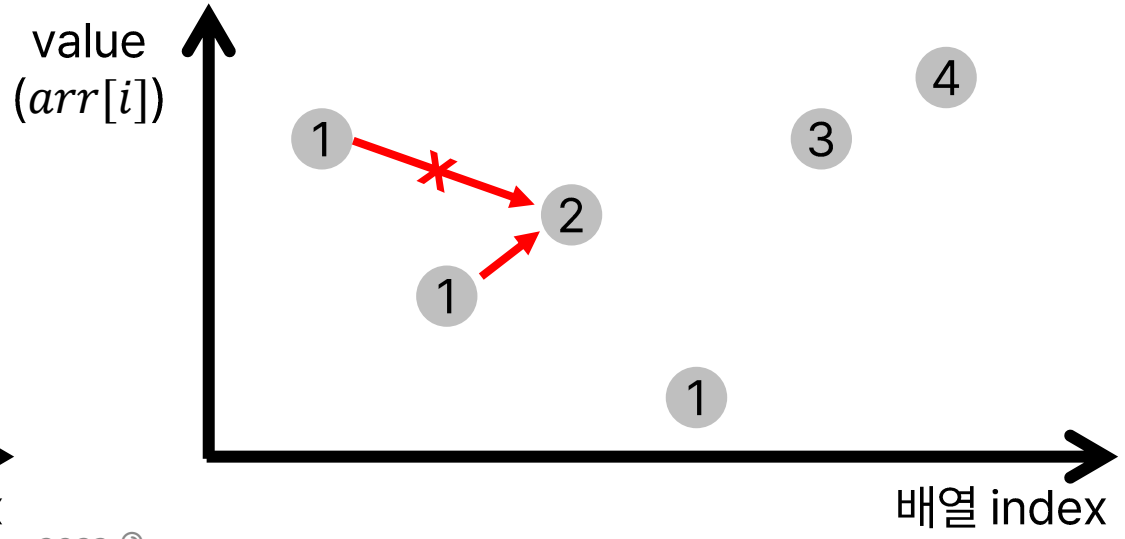
- 몇가지 성질을 파악해 보자
- 같은 길이의 부분수열을 가지는 끝 값 두 개가 어떻게 위치할까
- 노란색 점이 길이 3을 가질 수 있을까?



Longest Increasing Subsequence

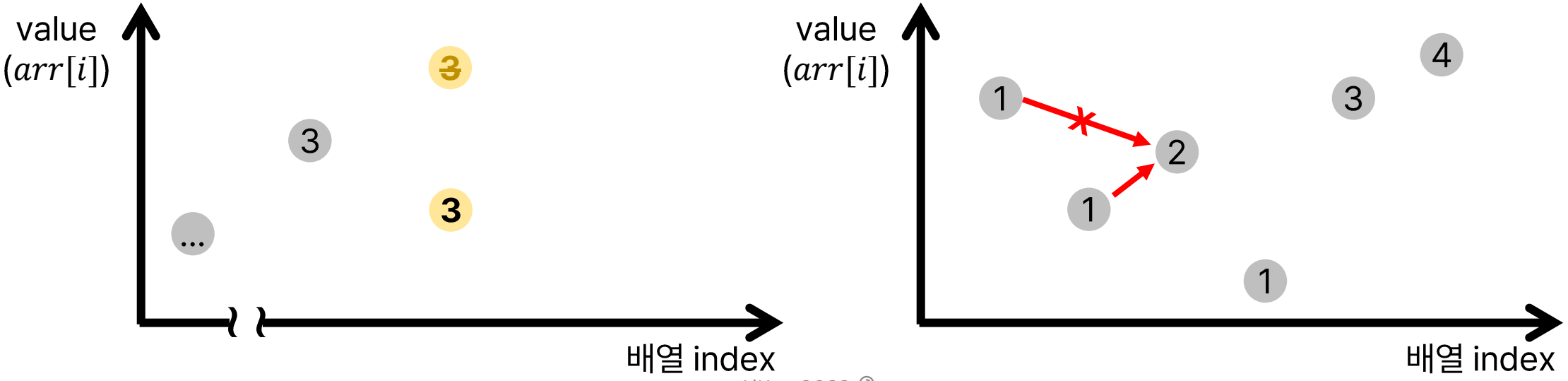
- 같은 부분 수열의 길이가 뒤쪽에 다시 나온다면, **값은 같거나 더 작다**
- **가장 뒤쪽의 값만 기억해도 된다**
- 왼쪽에서 오른쪽으로 계산, 더 낮은 값을 기억해야 수열의 길이를 더 길게 만들 수 있음





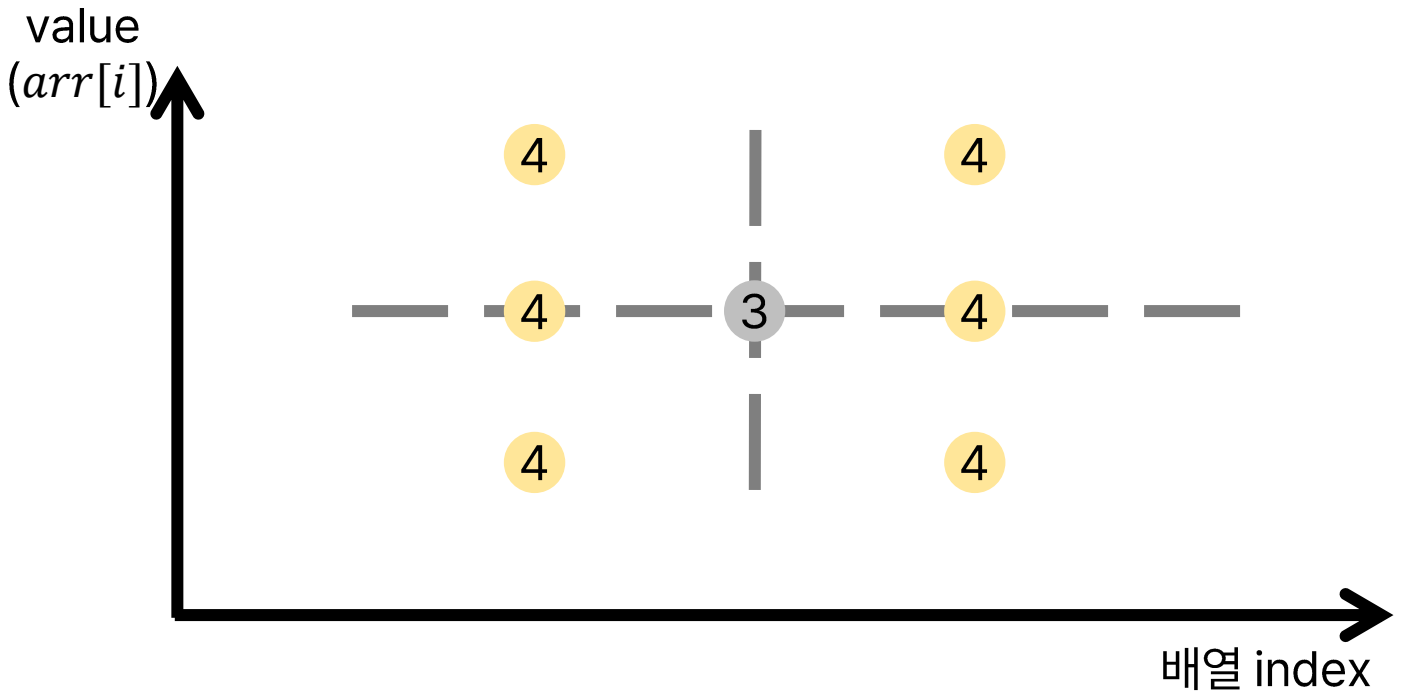
Longest Increasing Subsequence

- **수열의 길이가 같다**면 가장 낮은 값을 기억해야 LIS를 길게 만들 수 있다
- 가장 최근에 나온 값이 가장 낮으므로, **최근에 받은 값**을 기억하면 된다



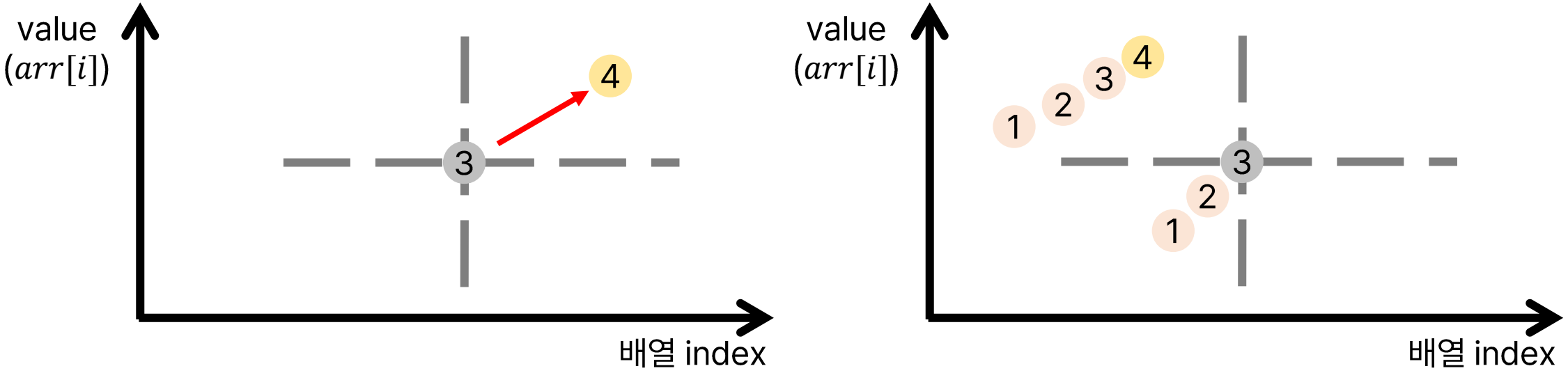
Longest Increasing Subsequence

- 마지막으로 기억하는 길이 3 (회색 원), 다음 6개의 위치 중 길이 4가 올 수 있는 곳은?



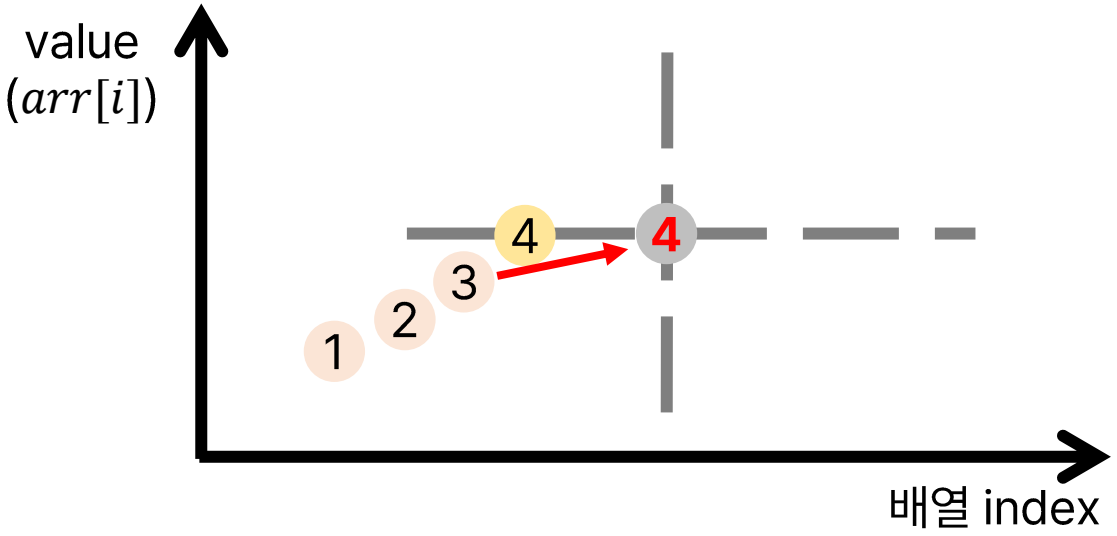
Longest Increasing Subsequence

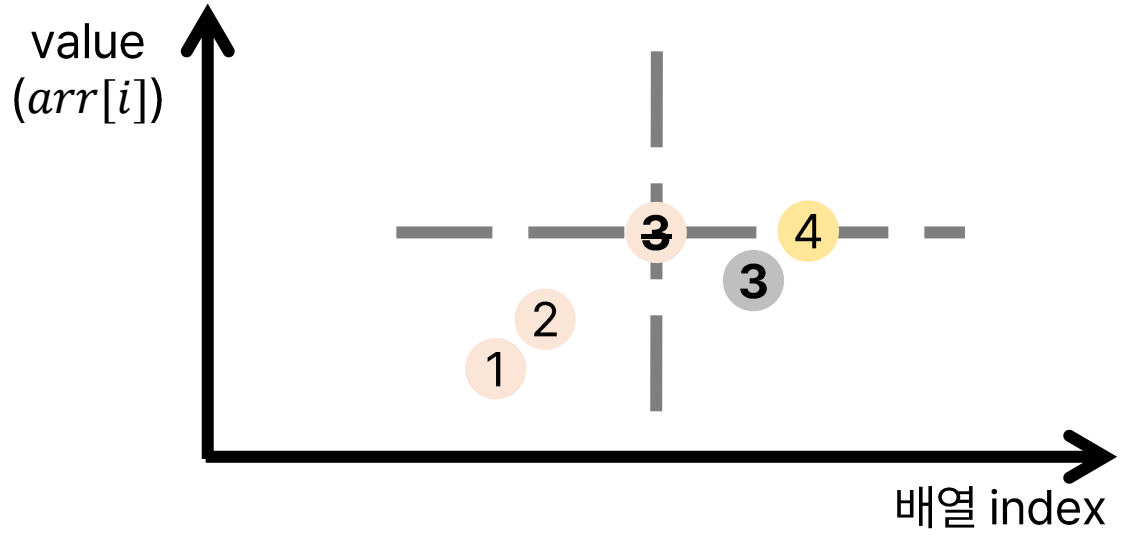
- 우선 길이 3이 가지는 값보다 큰 값을 가지는 경우는 둘 다 가능하다



Longest Increasing Subsequence

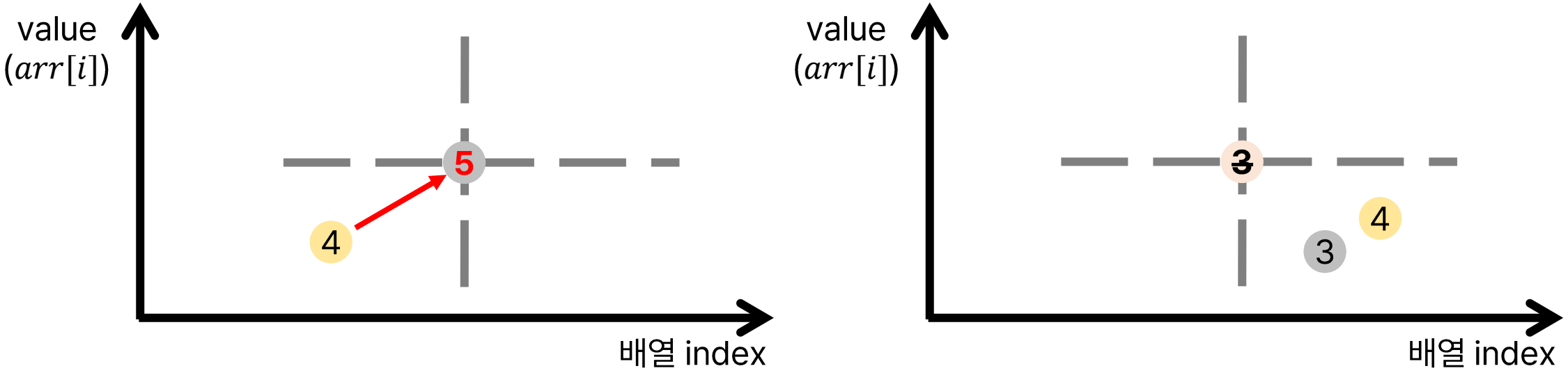
- 길이 3이 가지는 값과 같은 경우는 불가능하다
- 왼쪽의 경우에는 앞쪽 길이 4가 받는 것을 그대로 받으면 됨
- 오른쪽의 경우에는 길이 3인 가장 마지막 값을 기억하지 않았음





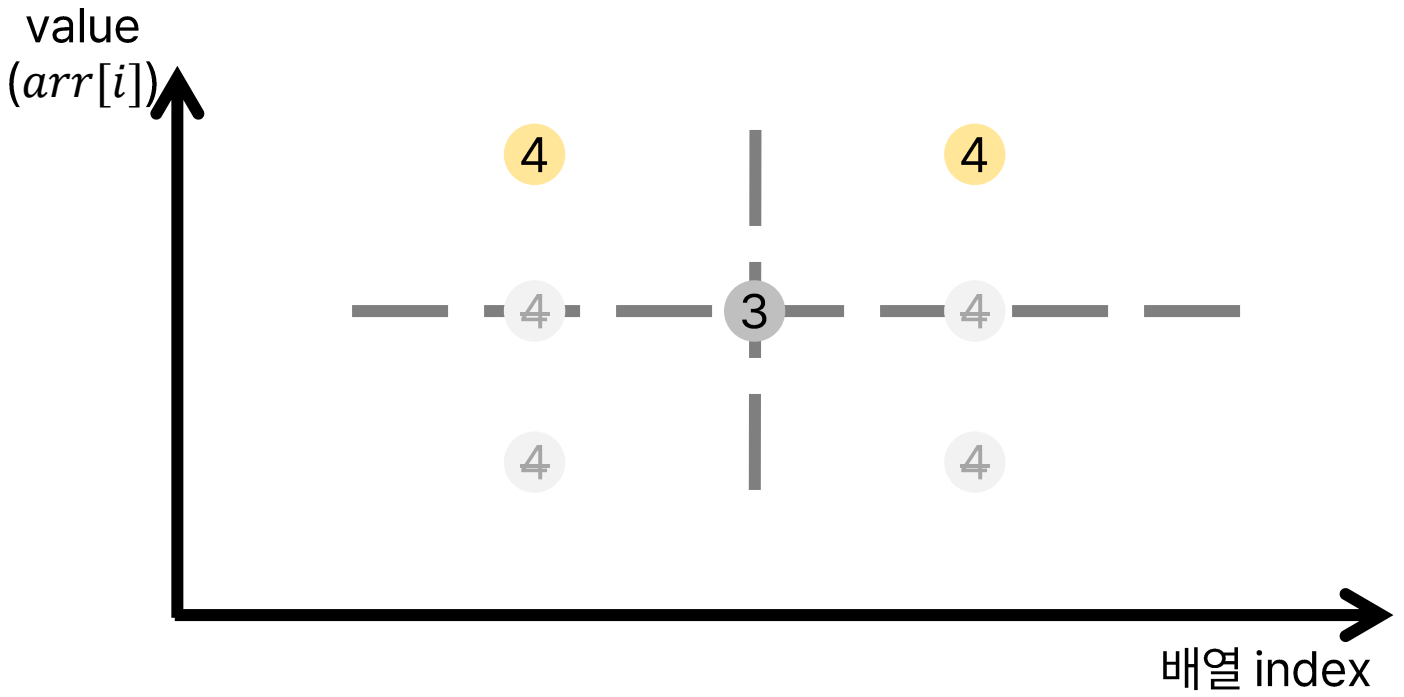
Longest Increasing Subsequence

- 길이 3이 가지는 값보다 작은 값을 가지는 경우에도 불가능하다
- 왼쪽의 경우에는 앞쪽 길이 4보다 크므로 길이 5가 됨
- 오른쪽의 경우에는 길이 3인 가장 마지막 값을 기억하지 않았음



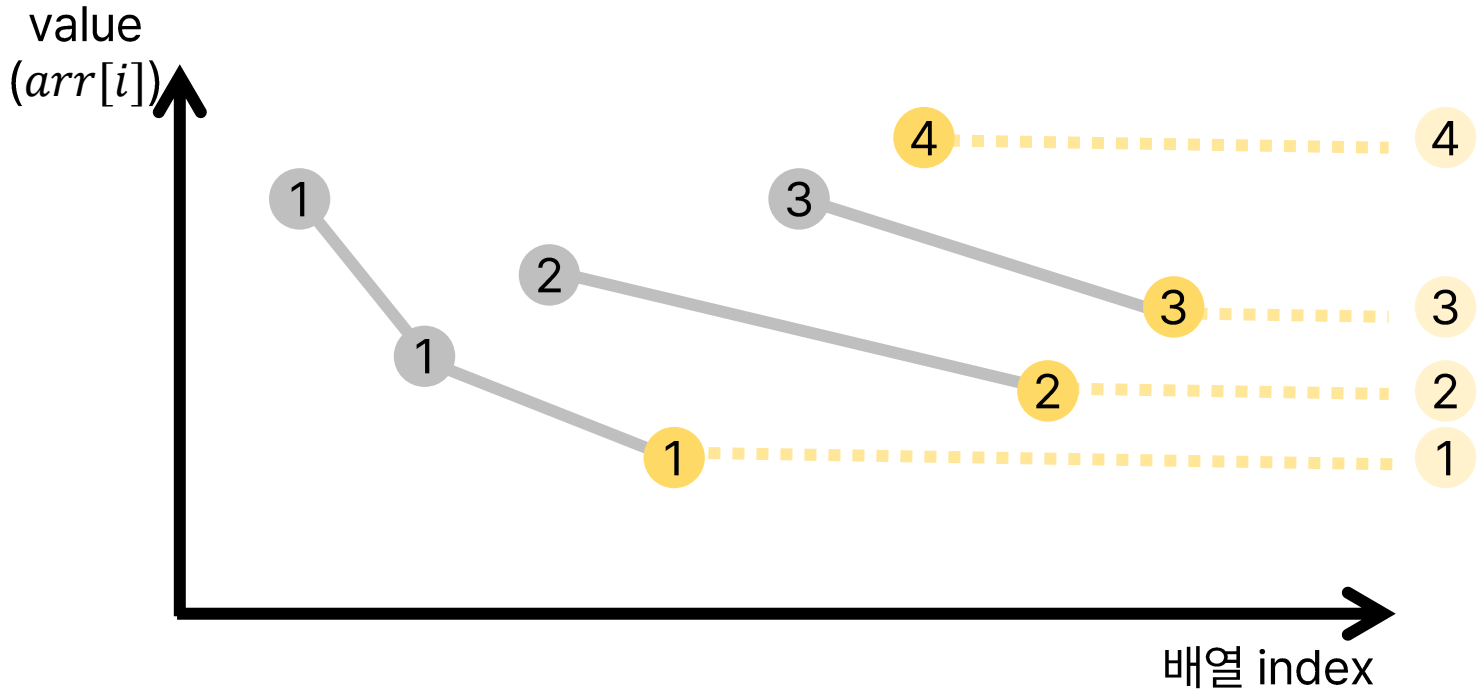
Longest Increasing Subsequence

- 마지막으로 기억하는 길이 3, 다음 위치 중 길이 4가 올 수 있는 곳은 **3보다 큰 두 곳**



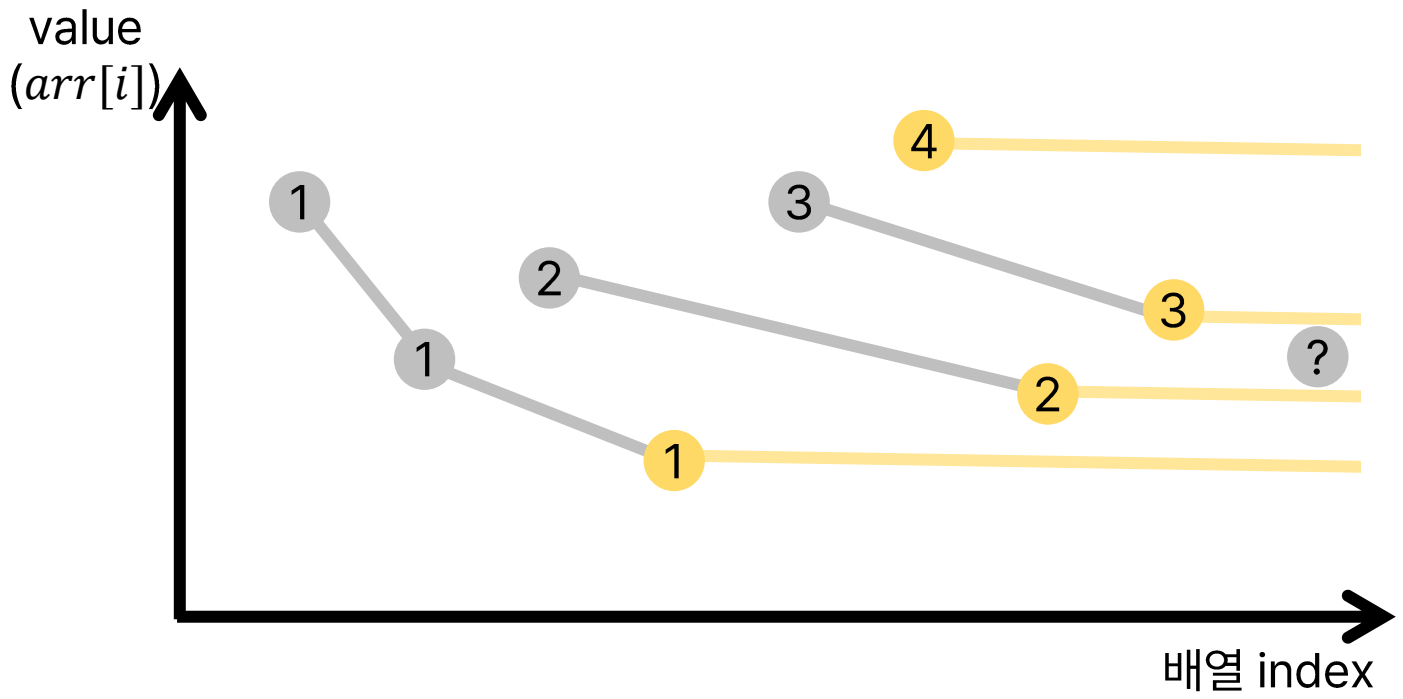
Longest Increasing Subsequence

- 길이가 같은 것을 서로 이으면 더 긴 것이 위에 존재하고, 선끼리 교차하지 않는다
- 같은 길이 중에서도 가장 나중 값을 기억한다면, 기억하고 있는 길이들의 값은 오름차순



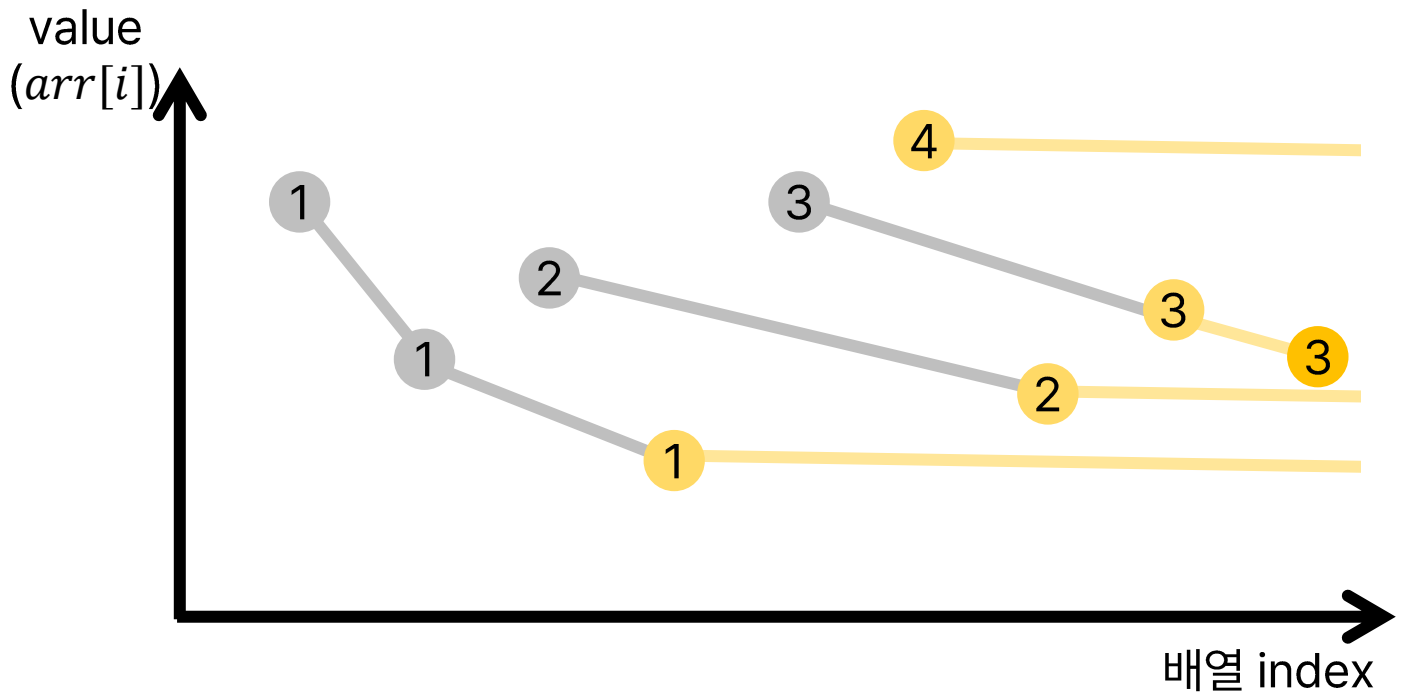
Longest Increasing Subsequence

- 이 상태에서 새로운 값이 들어온다면, LIS를 $O(logN)$에 구할 수 있다



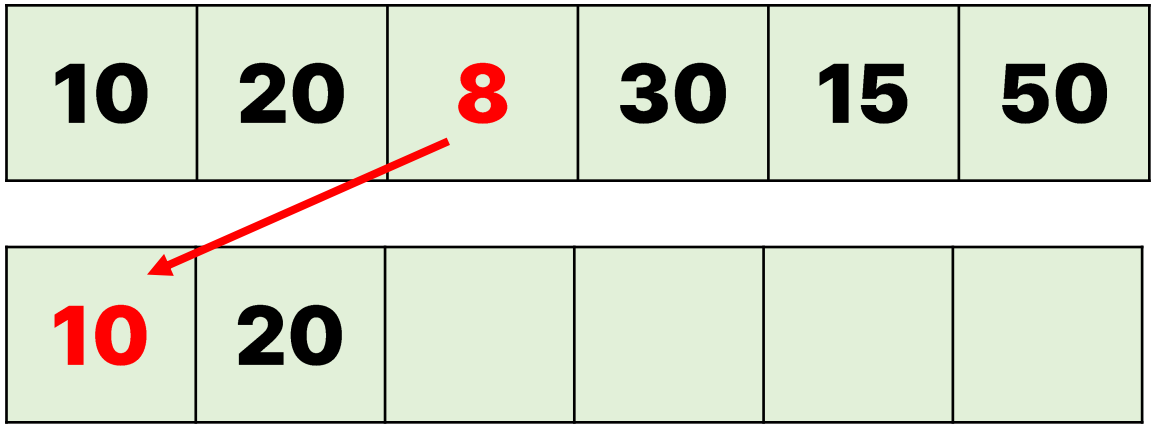
Longest Increasing Subsequence

- 이 상태에서 새로운 값이 들어온다면, LIS를 $O(logN)$에 구할 수 있다



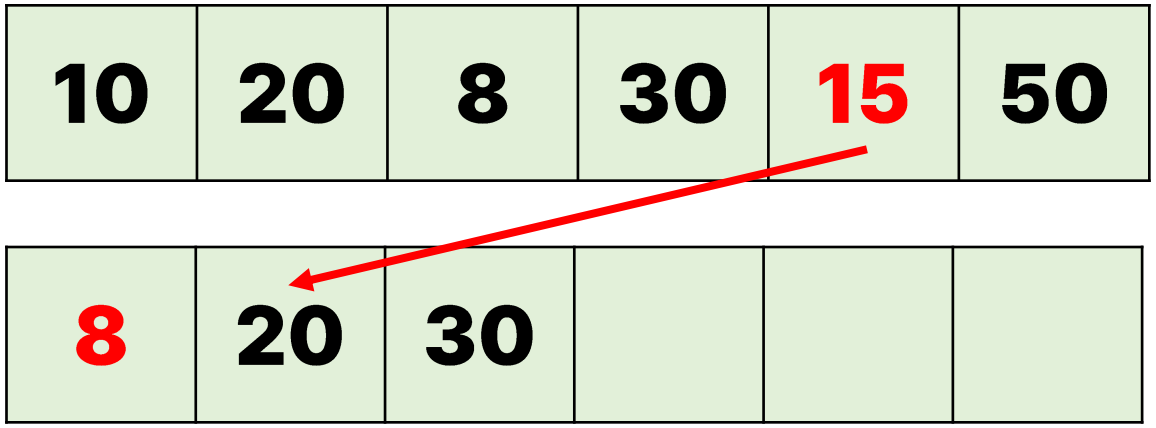
Longest Increasing Subsequence

- 배열을 정의하자, i번째 값은 길이 i의 LIS 중 가장 작은 값 (최근 값)
- LIS 배열의 가장 오른쪽 값보다 크다면, LIS 길이가 추가되는 것과 같다
- 그렇지 않으면, 해당 값이 어떤 LIS 길이에 속하는지 이분 탐색을 통해 찾는다



Longest Increasing Subsequence

- 배열을 정의하자, i번째 값은 길이 i의 LIS 중 가장 작은 값 (최근 값)
- LIS 배열의 가장 오른쪽 값보다 크다면, LIS 길이가 추가되는 것과 같다
- 그렇지 않으면, 해당 값이 어떤 LIS 길이에 속하는지 이분 탐색을 통해 찾는다



Longest Increasing Subsequence

- 배열을 정의하자, i번째 값은 길이 i의 LIS 중 가장 작은 값 (최근 값)
- LIS 배열의 가장 오른쪽 값보다 크다면, LIS 길이가 추가되는 것과 같다
- 그렇지 않으면, 해당 값이 어떤 LIS 길이에 속하는지 이분 탐색을 통해 찾는다

10	20	8	30	15	50

8	15	30	50		

Longest Increasing Subsequence

- 모든 값에 대해서 LIS를 처리했다면, LIS 배열의 길이가 곧 LIS의 길이
- 모든 값에 대해서 LIS를 처리하는 과정의 시간복잡도
- $O(NlogN)$

10	20	8	30	15	50

8	15	30	50		

Longest Increasing Subsequence

- 아래 LIS 배열에서 새로 들어오는 수가 10이라면?
- 결정 문제: 10 이상의 수인가? Y로 바뀌는 최초의 수를 찾아야 함
- hi가 목표, 범위 조절에 신경쓰자

lo hi

N	N	N	N	Y	Y	Y	Y	Y	Y
-1	2	4	7	11	13	15	17	18	19

Longest Increasing Subsequence

```
lis.push_back(-1);
for(int i = 0; i < N; i++) {
    cin >> a;
    if (lis.back() < a) {
        lis.push_back(a);
        continue;
    }
    int lo = 0, hi = lis.size()-1;
    while (lo + 1 < hi) {
        int mid = (lo + hi) / 2;
        if (!(lis[mid] >= a)) lo = mid;
        else hi = mid;
    }
    lis[hi] = a;
}
cout << lis.size() - 1 << '\n';
```

```
lis.push_back(-1);
for(int i = 0; i < N; i++){
    cin >> a;
    if (lis.back() < a) lis.push_back(a);
    else{
        int idx = lower_bound(lis.begin(),
                              lis.end(), a) - lis.begin();
        lis[idx] = a;
    }
}
cout << lis.size() - 1 << '\n';
```

References

- https://www.acmicpc.net/blog/view/109
- https://github.com/justiceHui/SSU-SCCC-Study/blob/master/2023-summer-basic/slide/04-4-binary-search.pdf
- https://blog.naver.com/jinhan814/222156334908
- 교수님 강의