Greedy

탐욕법

Sogang ICPC Tea

2020 Winter 초급 1차시

서강대 컴퓨터공학과 강효규

Algorithm

알고리즘이란 무엇일까?

✓ 어떠한 문제를 해결하기 위해 정해진 일련의 절차나 방법을 공식화한 형태로 표현한 것

✓ 즉 알고리즘이란 문제 해결 방식, 문제 풀이 패러다임

✓ 알고리즘의 성능평가를 하기 위한 지표로 시간 복잡도, 공간 복잡도 사용

Algorithm 빅-오 표기법 (Big-O notation)

✓시간, 공간 복잡도를 표현하는 점근적 표기 방식

✓ 최악의 경우를 생각하여 계산한다

✓ 시간,공간 복잡도의 가장 영향력 있는 항으로 표현하고 계수는 무시한다.

Algorithm 공간 복잡도(space complexity)

✓ 입력의 크기와 사용한 메모리 크기의 함수 관계

✓ 프로그램을 실행 및 완료하는 데 필요한 저장공간의 양

총 공간 요구 = 고정 공간 요구 + 가변 공간 요구: $S(P) = c + Sp(n)$

고정 공간: 입력과 관계없는 공간의 요구 $(\therefore$ 상수취급$)$
가변 공간: 입력과 연관이 있음 $(\therefore$ 알고리즘의 공간 복잡도 계산$)$

Algorithm 공간 복잡도(space complexity)

```
1  int factorial(int n)
2  {
3      if(n > 1) return n * factorial(n - 1);
4      else return 1;
5  }
```

재귀적으로 parameter에 1이 들어올 때까지 차곡차곡 쌓이다가 종료되므로 O(n)

```
1  int factorial(int n)
2  {
3      int fac = 1;
4      for (int i = 1; i <= n; i++)
5          fac = fac * i;
6      return fac;
7  }
```

상수 크기의 저장공간만이 필요로 하므로 O(1)

Algorithm 시간 복잡도(time complexity)

✓ 입력의 크기와 실행한 시간 함수 관계

✓ 프로그램을 실행 및 완료하는 데 필요한 연산의 양

보통 알고리즘 대회에서 1초당 약 10^8 (1억번) 의 연산이 가능하다고 가정

자신의 구상한 알고리즘의 시간복잡도를 계산하는 것이 문제풀이에 앞서 매우 중요한 일

Algorithm 시간 복잡도(time complexity)

```
1  int sum(int n)
2  {
3      int ret = 0;
4      for (int i = 1; i <= n; i++)
5          ret += i;
6      return ret;
7  }
```

for 반복문의 연산이 n번 이루어지므로 $O(n)$

```
1  int sum(int n) {
2      return n * (n + 1) / 2;
3  }
```

바로 sum 을 구할 수 있으므로 $O(1)$

Greedy

✓ 어떤 문제를 해결하는 순진한 방법은 모든 경우에 대한 최적해를 구하면 된다.

✓ 하지만 이런 방식으로는 시간복잡도와 공간복잡도가 매우 크게 나타난다.

✓ 다음과 같은 문제를 생각해보자.

Greedy - 예제

- n번, 정수 a와 b가 주어진다.
- 각각의 a와 b에 대해 둘 중 하나를 선택할 수 있다.
- 선택한 값들의 합이 최대가 되도록 할 때, 선택한 값들의 합은 얼마인가?
- 즉 n번의 선택에 대한 최댓값을 묻는 문제.

Greedy - 예제

모든 경우 탐색!!

4 8 6	18
4 8 5	17
4 3 6	13
4 3 5	12
7 8 6	21
7 8 5	20
7 3 6	16
7 3 5	15

시간복잡도 $O(2^n)$

그리디 알고리즘이란?

✓ 가장 직관적인 알고리즘 설계 패러다임

✓ 모든 선택지를 고려하지 않고, 각 단계마다 지금 가장 좋은(탐욕적인) 방법만을 선택

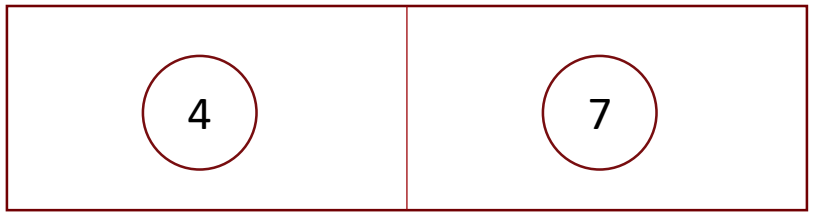
✓시공간 복잡도의 향상된 방법으로 문제를 해결할 수 있다.

✓ 그리디 알고리즘의 정당성 증명

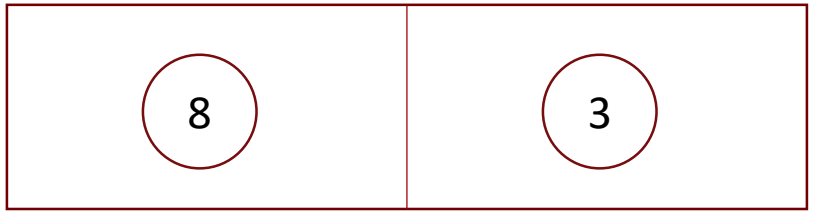
Greedy - 예제

이전 문제에서 그리디 알고리즘을 적용해보자

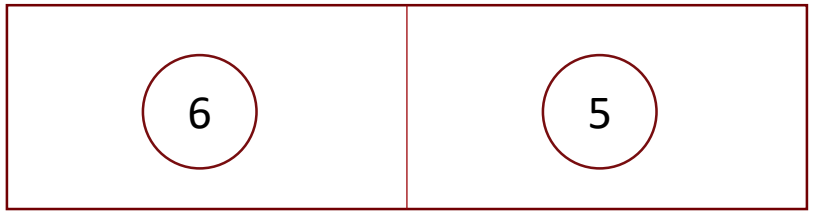
n=1



n=2



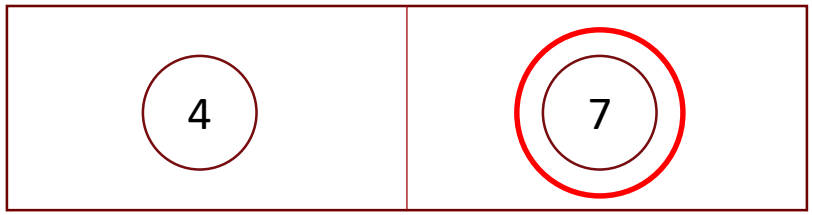
n=3



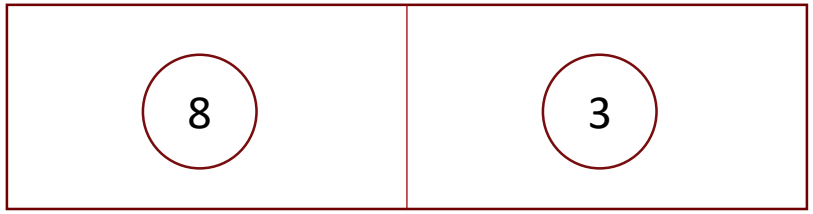
우리는 매 단계마다 더 큰 숫자를 선택할 것이다.

Greedy - 예제

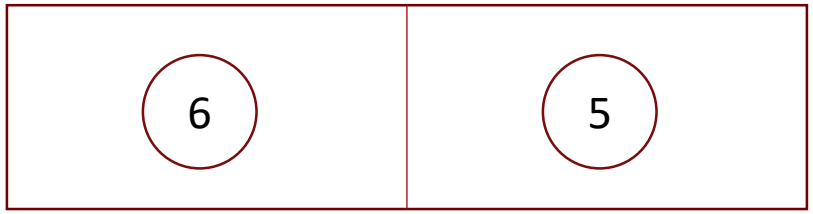
n=1



n=2

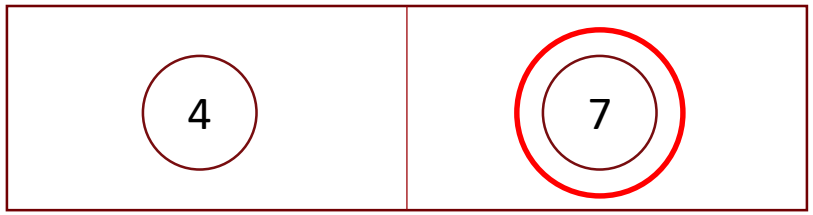


n=3

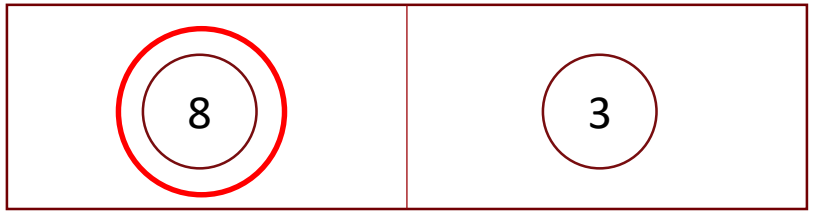


Greedy - 예제

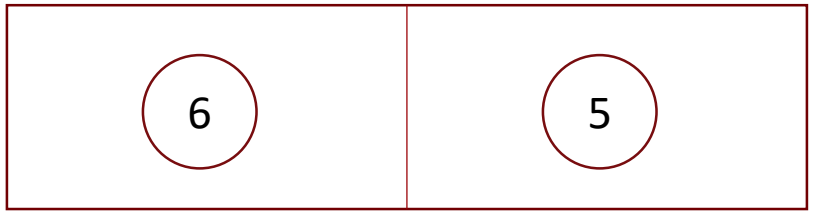
n=1



n=2

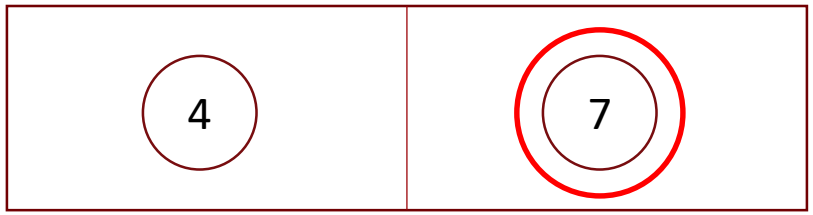


n=3

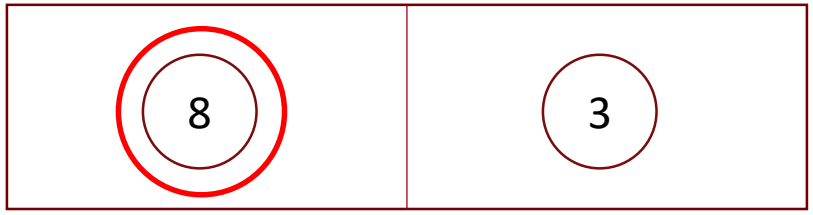


Greedy - 예제

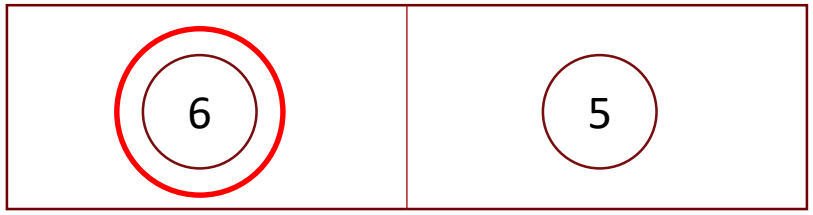
n=1



n=2



n=3

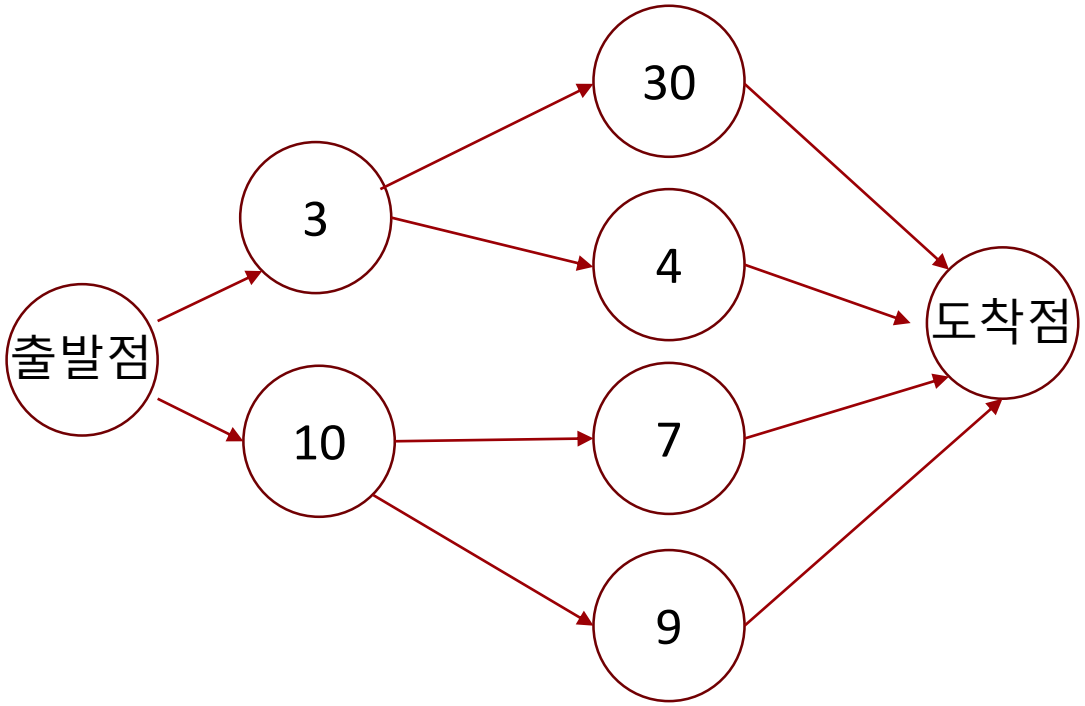


최적해를 그리디하게 구해냈다!

시간복잡도 $O(n)$

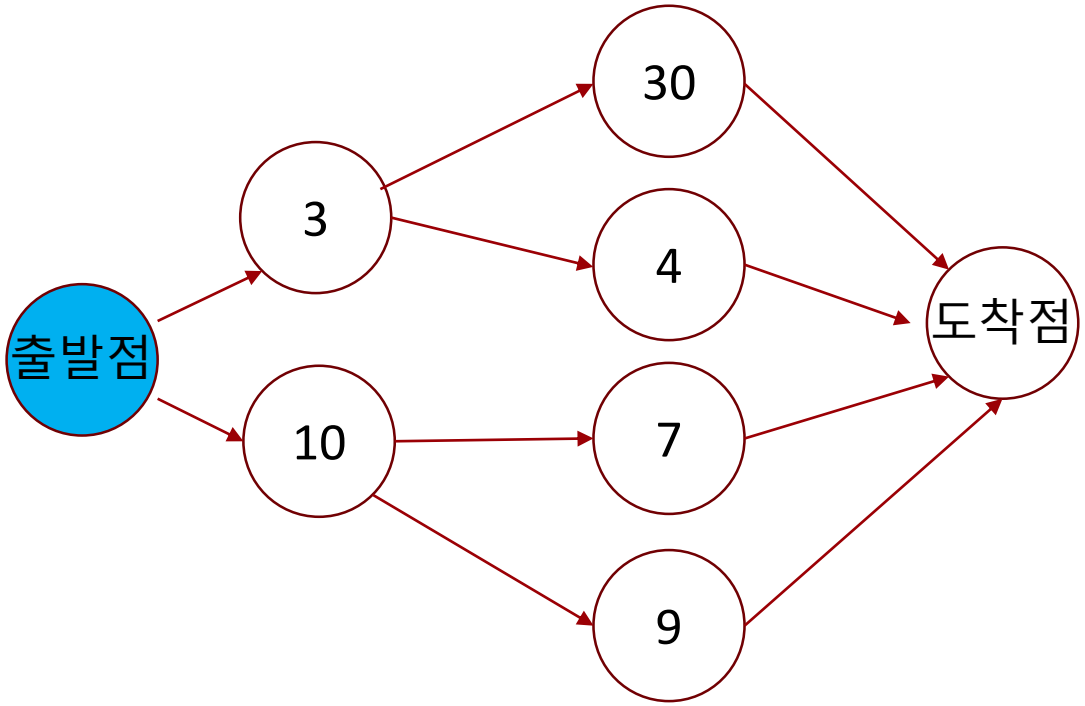
- 그리디 알고리즘이 항상 성립할까?

- 다음과 같은 그래프에서 출발점에서 도착점까지의 모든 이동경로 중, 각 지점에 적힌 수들의 합을 최대화 하는 방안을 생각해보자.

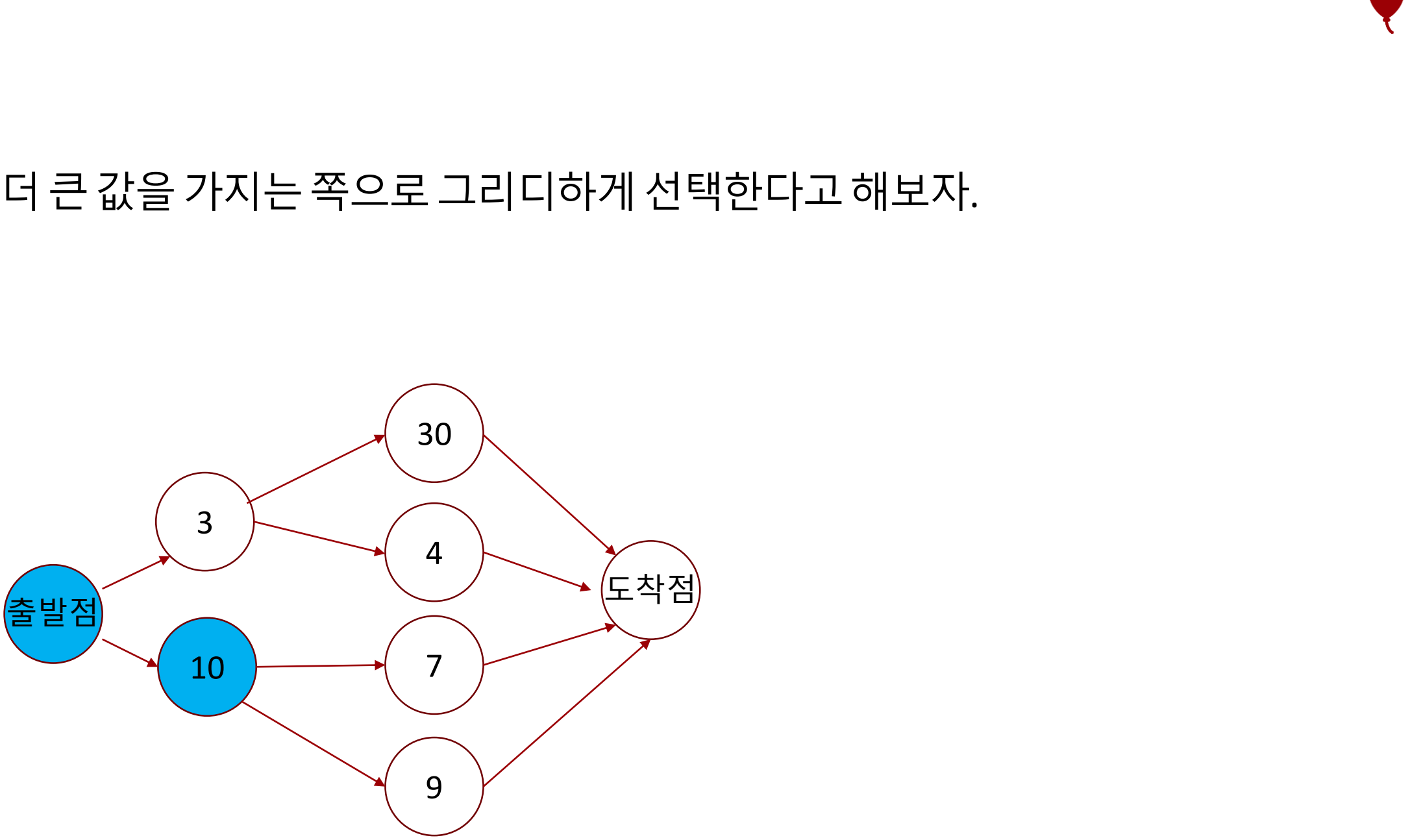


Greedy

- 항상 더 큰 값을 가지는 쪽으로 그리디하게 선택한다고 해보자.

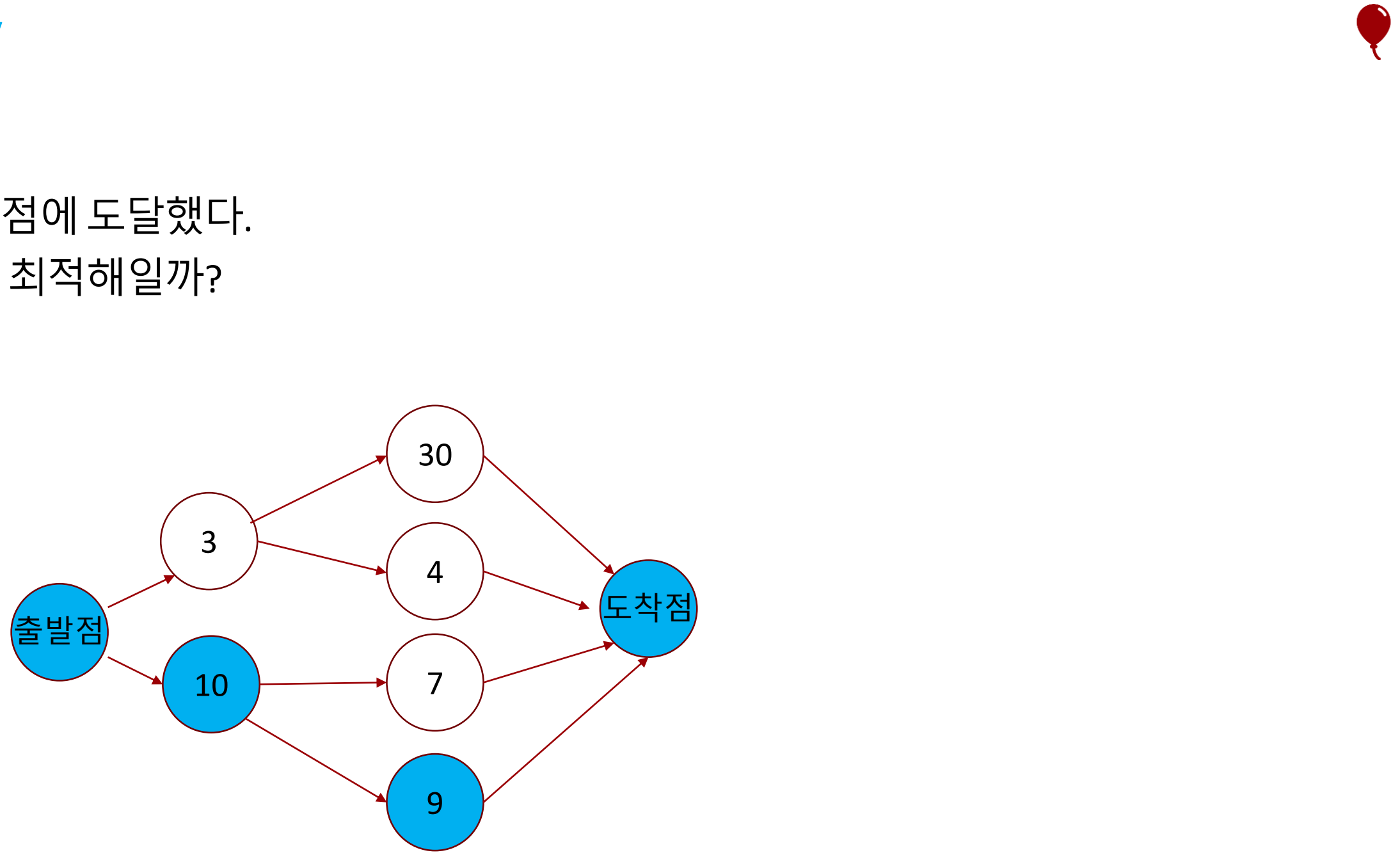


Greedy



- 항상 더 큰 값을 가지는 쪽으로 그리디하게 선택한다고 해보자.

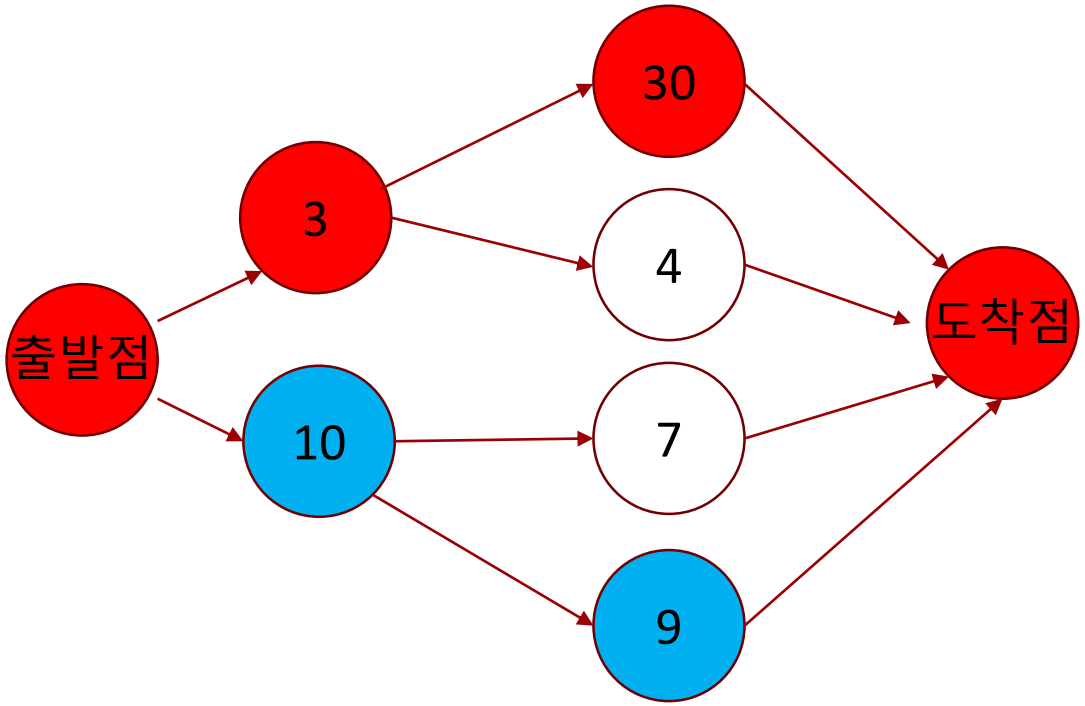
Greedy



- 도착점에 도달했다.
- 이게 최적해일까?

Greedy

- 최적해는 빨간색을 따라 선택하는 방법이다.



Greedy 정당성 증명

✓ 그리디한 방법은 많은 경우 최적의 정답을 찾아주지 못한다.

✓ 따라서 그리디 알고리즘을 사용할 때, 그 정당성을 증명하는 것이 중요하다.

그리디 알고리즘의 정당성 증명

1. 탐욕적으로 선택하더라도 문제의 최적해를 구할 수 있다.
2. 부분 문제의 최적해에서 전체 문제의 최적해를 만들 수 있다.

Greedy 정당성 증명

탐욕적 선택 속성(greedy choice property)

각 단계에서 탐욕적으로 내리는 선택은 항상 최적해로 가는 길 중 하나라는 것.
즉 탐욕적인 선택을 해서 손해를 볼 일이 없다는 것.

최적 부분 구조(optimal substructure)

부분 문제의 최적해에서 전체 문제의 최적해를 만들 수 있다는 것.
대부분 자명한 경우가 많다.
그리디가 아니더라도 성립할 수 있다. (ex)다이나믹 프로그래밍)

Greedy 정당성 증명

✓ 최적 부분 구조라면, 전체 문제를 부분문제들의 최적해를 구하는 것으로 치환할 수 있다.

✓탐욕적 선택 속성을 만족한다면, 그리디한 접근을 통해 부분문제의 최적해를 구할 수 있음을 보여주므로 정당성이 증명된다.

✓ 최적해를 얻을 수 있는 접근이 직관적이지 않은 경우도 많기 때문에 정당성을 증명하는 과정을 빼먹지 않고 연습하는 것이 좋다.

#1931 회의실 배정

한 개의 회의실이 있는데 이를 사용하고자 하는 N개의 회의에 대하여 회의실 사용표를 만들려고 한다. 각 회의 I에 대해 시작시간과 끝나는 시간이 주어져 있고, 각 회의가 겹치지 않게 하면서 회의실을 사용할 수 있는 회의의 최대 개수를 찾아보자. 단, 회의는 한번 시작하면 중간에 중단될 수 없으며 한 회의가 끝나는 것과 동시에 다음 회의가 시작될 수 있다. 회의의 시작시간과 끝나는 시간이 같을 수도 있다. 이 경우에는 시작하자마자 끝나는 것으로 생각하면 된다.

```
11
1 4
3 5
0 6
5 7
3 8
5 9
6 10
8 11
8 12
2 13
12 14
```

```
4
```

#1931 회의실 배정

✓ 우리는 전체 회의 중 최대한 많은 회의가 진행되도록 회의를 선택해야 한다.

✓ 이때 우리가 어떤 회의를 첫번째 회의로 선택하면, 그 회의와 시간이 겹치는 회의는 선택할 수 없다.

✓ 그 후 남은 회의들에서도 우리는 최대한 많은 회의가 진행될 수 있도록 똑같은 방식으로 회의를 선택해야 한다.

✓ 따라서 최적 부분 구조 문제이다

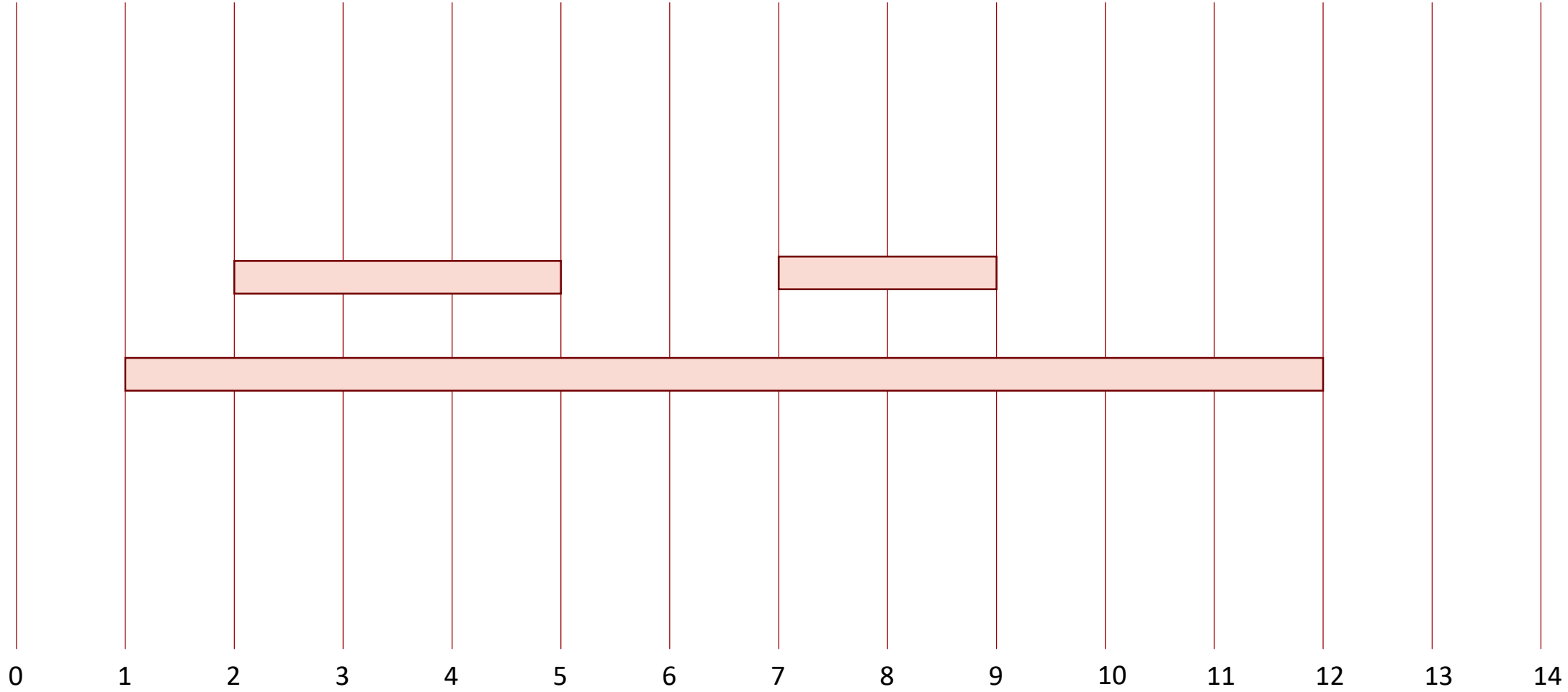
최적 부분 구조임을 알았으므로 각 단계마다 어떻게 선택할 것인지를 정해주고,

만약 그러한 방법이 항상 최적해를 보장해준다면, 그 방법대로 하는 것이 정답이다.

어떻게 선택할 것인지가 가장 중요한 문제!

#1931 회의실 배정

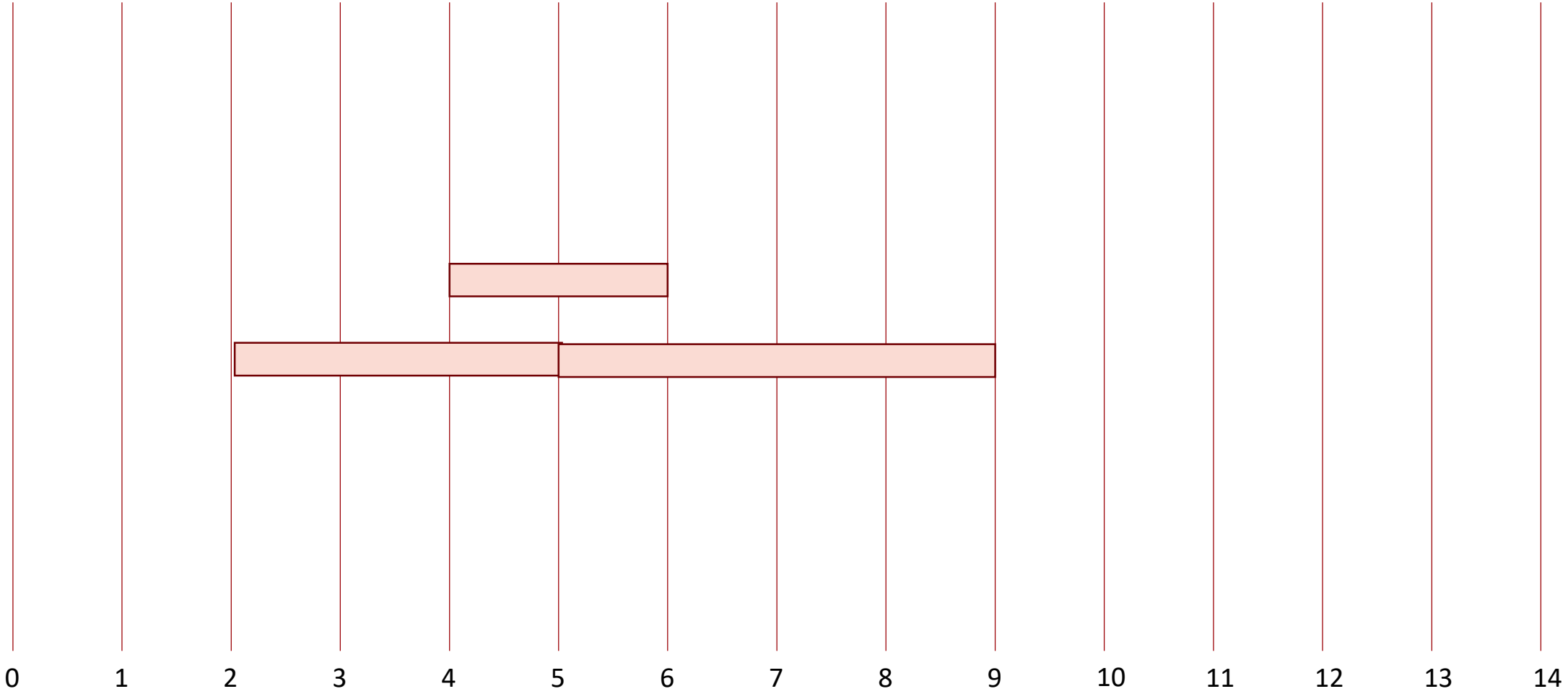
1. 가장 먼저 시작하는 회의를 그리디하게 선택한다.



위 같은 경우에 최적해를 구하지 못한다.

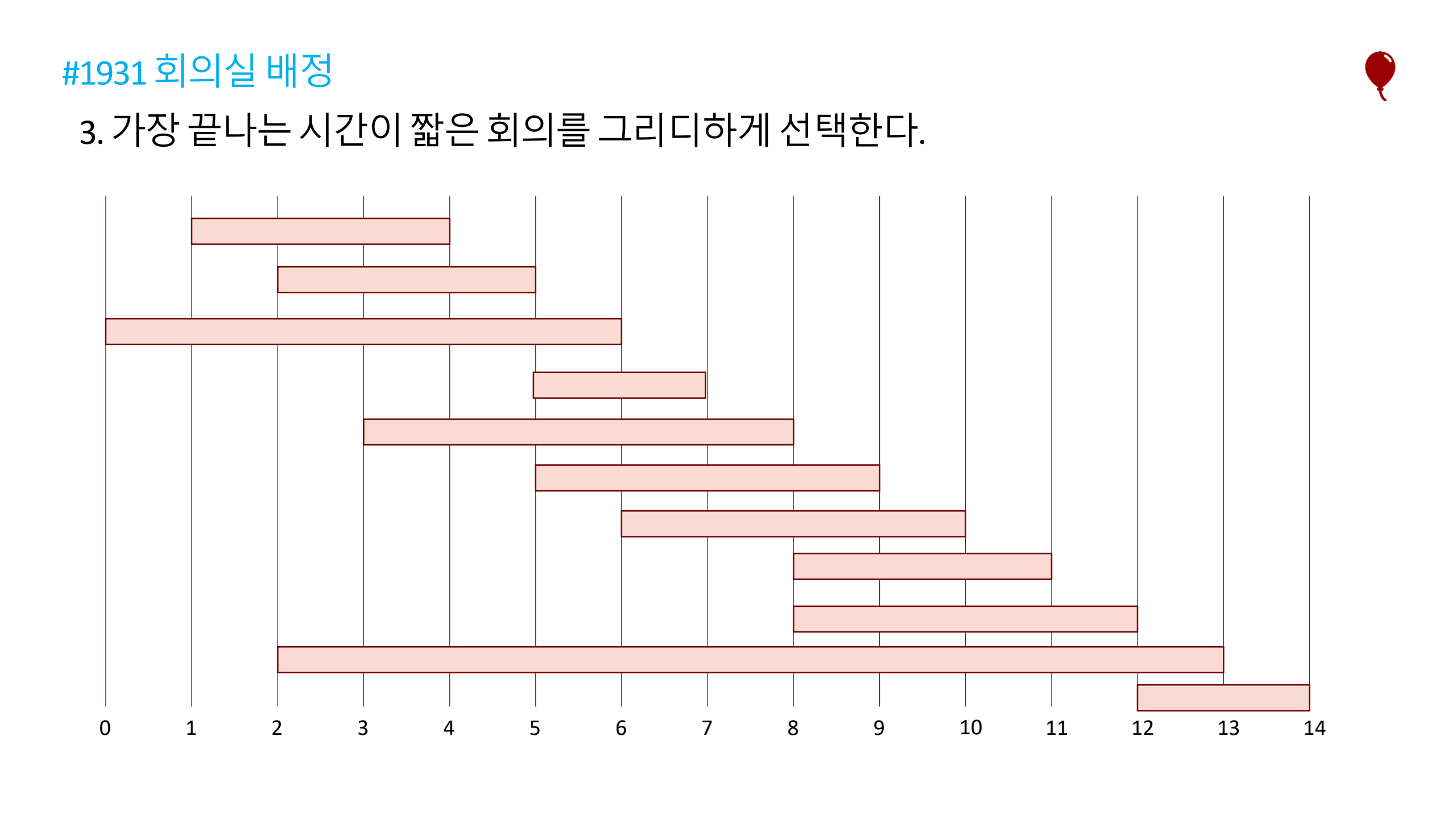
#1931 회의실 배정

2. 가장 진행 시간이 짧은 회의를 그리디하게 선택한다.



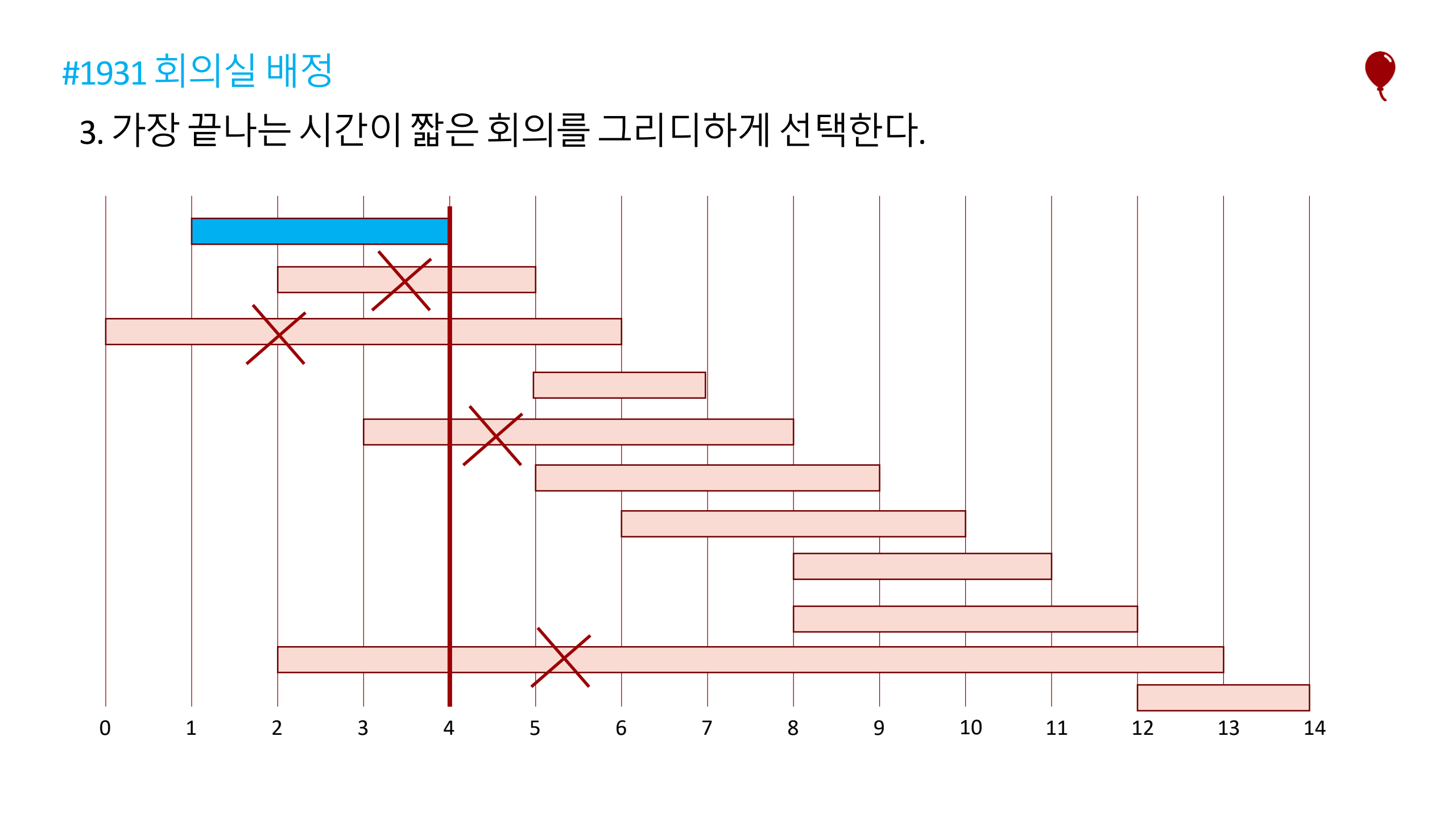
위 같은 경우에 최적해를 구하지 못한다.

#1931 회의실 배정



3. 가장 끝나는 시간이 짧은 회의를 그리디하게 선택한다.

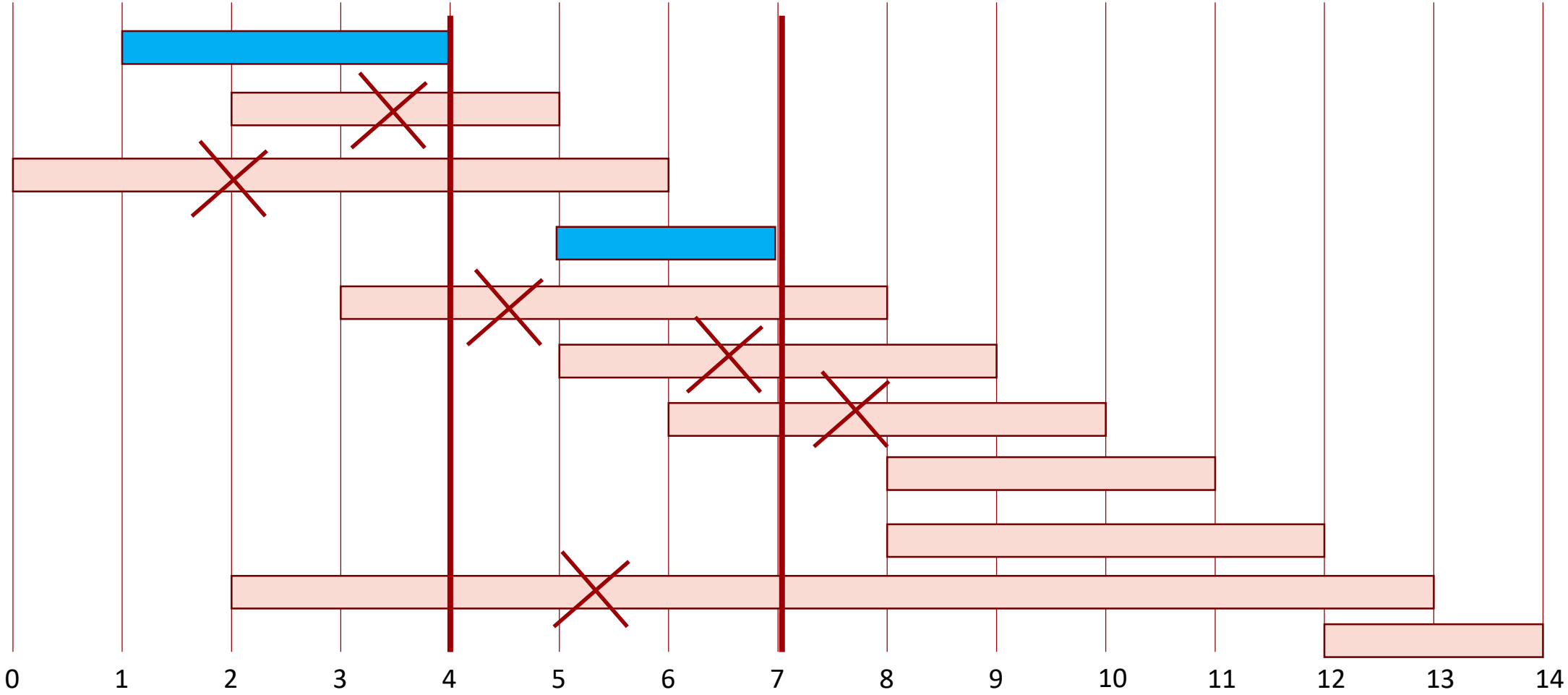
#1931 회의실 배정



3. 가장 끝나는 시간이 짧은 회의를 그리디하게 선택한다.

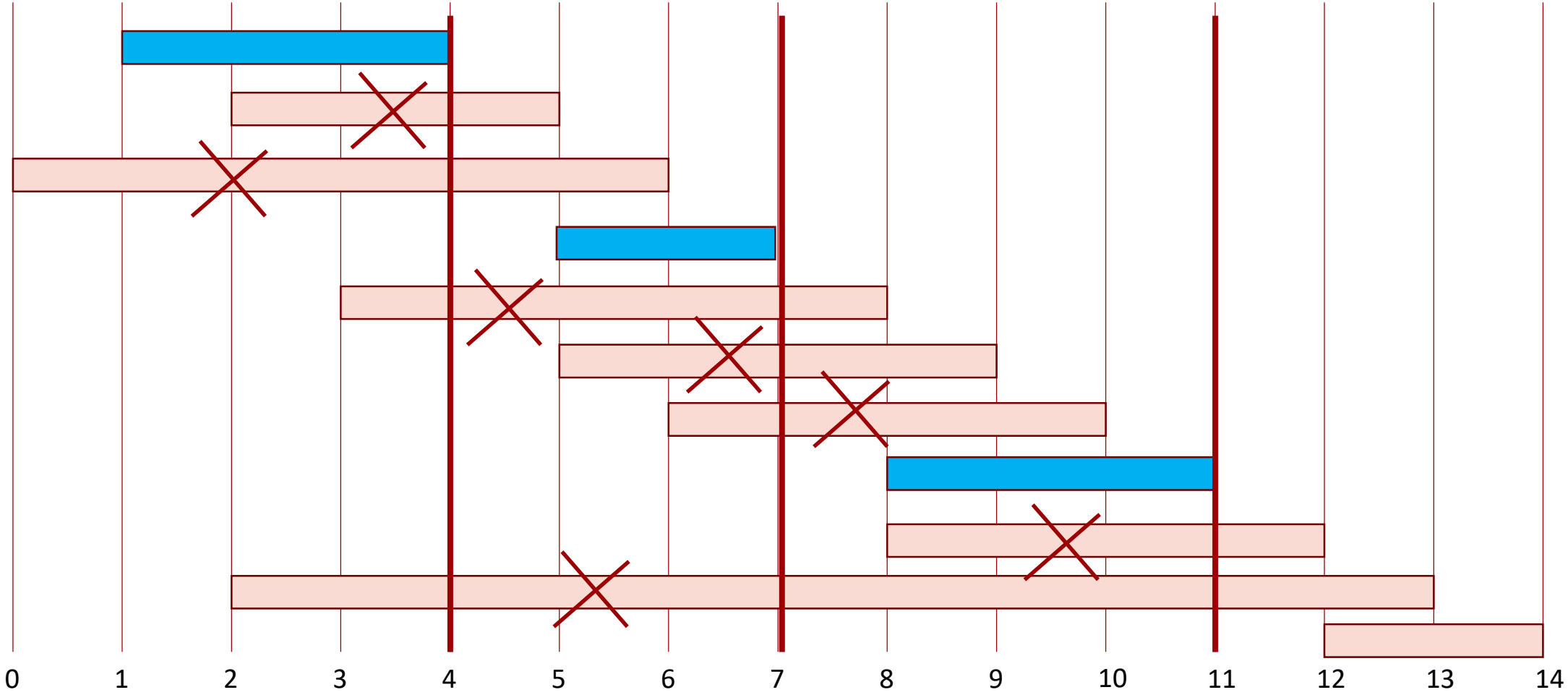
#1931 회의실 배정

3. 가장 끝나는 시간이 짧은 회의를 그리디하게 선택한다.



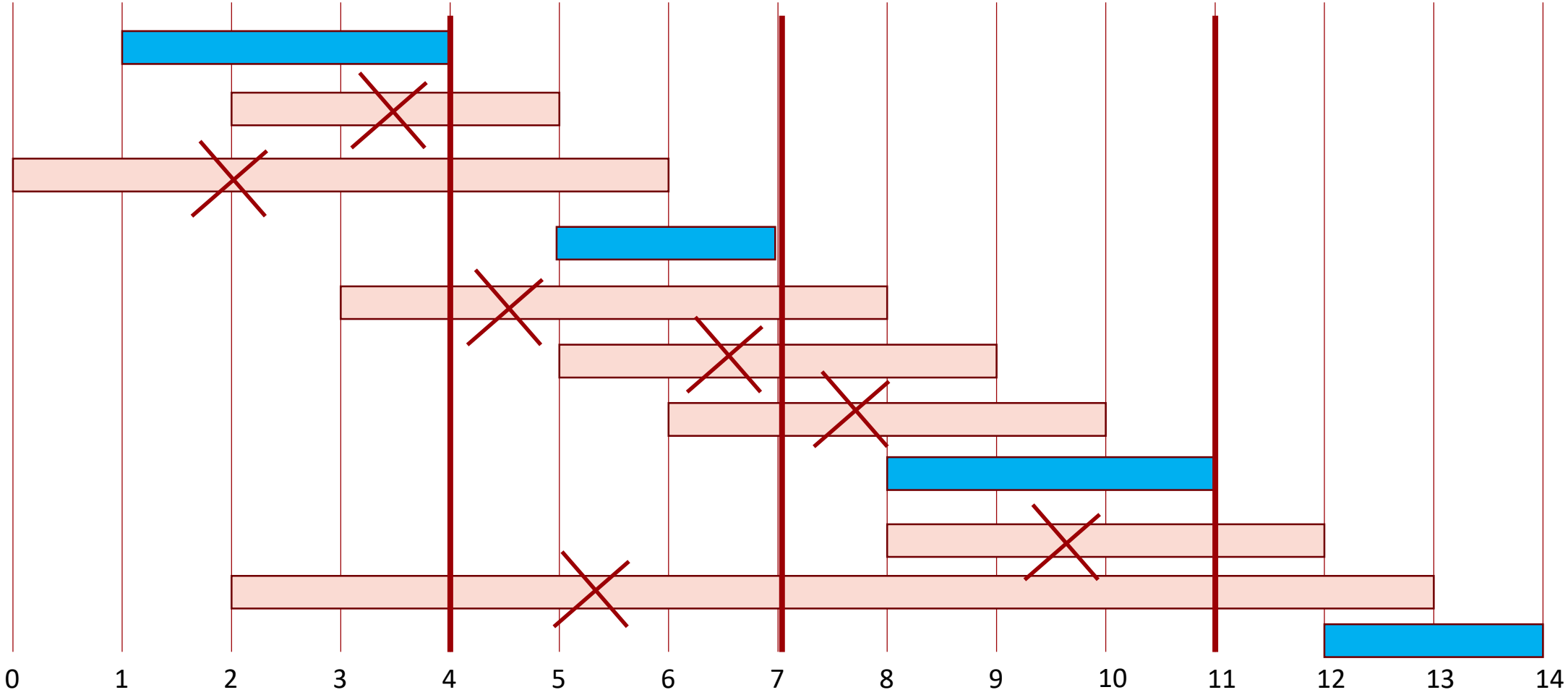
#1931 회의실 배정

3. 가장 끝나는 시간이 짧은 회의를 그리디하게 선택한다.



#1931 회의실 배정

3. 가장 끝나는 시간이 짧은 회의를 그리디하게 선택한다.



#1931 회의실 배정

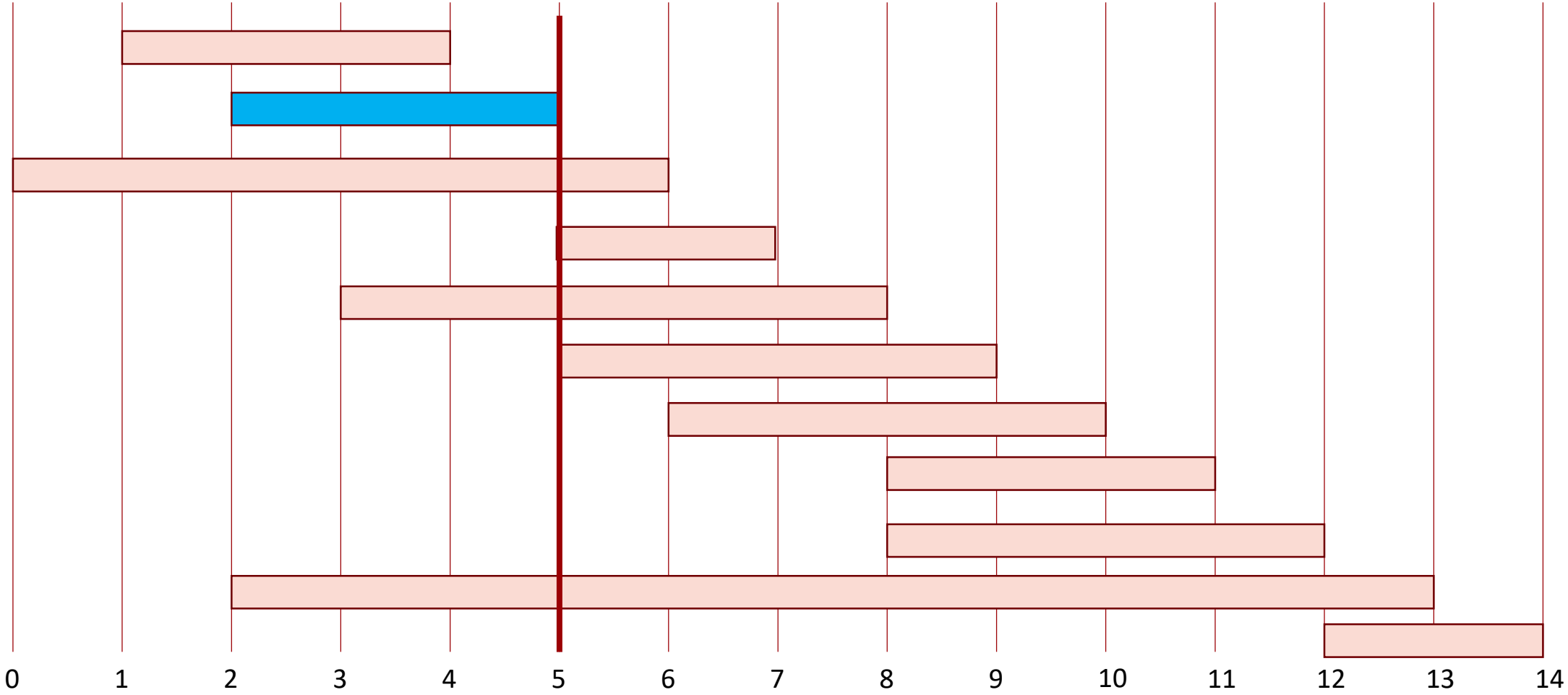
그리디한 선택의 정당성을 증명해 보자!

귀류법을 이용해 증명하겠다.

그리디한 선택이 최적해를 포함하지 않는 경우도 있다고 가정하자.
최적해가 존재함은 자명하기 때문에 그리디하지 않은 선택이 최적해를 포함할 것이다.

#1931 회의실 배정

그리디한 선택이 아니였을때 최적해가 존재한다고 할 경우 첫 회의 예시



#1931 회의실 배정

- 그런데 우리가 가정한 최적해의 제일 첫 회의의 끝나는 시간보다, 항상 제일 먼저 끝나는 회의의 끝나는 시간이 앞선다.

- 따라서 그 회의 대신에, 제일 먼저 끝나는 회의를 대신 선택할 수 있다.

그리디한 선택이 최적해를 포함하지 않는다는 가정에 모순이므로,

알고리즘의 정당성이 증명되었다.

#1931 회의실 배정

✓ 최적 부분 구조와, 탐욕적 선택 속성에 관한 예시를 보여주기 위해 위처럼 증명을 했다.

✓ 하지만 꼭 이처럼 증명할 필요는 없다. 내가 선택한 그리디한 방법의 정당성은 다양한 방법으로 증명할 수 있다.

#1931 회의실 배정 code

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

using namespace std;
int n, m, a, b;
int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0), cout.tie(0);
```

1~3 기본 헤더 파일

Algorithm헤더: sort함수
vector헤더: vector stl

5: 표준 namespace 사용

6: int 형 변수선언

8~9 표준 입출력 속도 향상

#1931 회의실 배정 code

```
10      cin >> n;
11      vector<pair<int,int>>v;
12      for (int i = 0; i < n; i++) {
13          cin >> a >> b;
14          v.push_back({ b,a });
15      }
16      sort(v.begin(), v.end());
```

10: n입력

11: pair, vector 이용

12~14:
시작 시간과 종료시간을 입력받음.
{종료시간,시작시간}순으로
vector에 넣는다.

16. Sort함수는 pair자료형의 앞에 자료 기준으로 오름차순 정렬해준다.

#1931 회의실 배정 code

```
17      a = 0, b = 0;
18      for (int i = 0; i < v.size(); i++) {
19          if (v[i].second < a)continue;
20          a = v[i].first;
21          b++;
22      }
23      cout << b;
```

17: 변수 선언
a는 지금 까지의 회의 중 가장 늦게 끝난 시각
b는 회의의 총 개수
18: 모든 회의에 대해 반복문을 통해 검사

19: 만약 i번째 회의의 시작시간이 a 보다 작다면 무시

20: 아니라면 이 회의를 선택하고 늦게 끝난 시각 갱신

21: 회의 개수 증가

23: 회의 개수 출력

#14659 한조서열정리하고옴 ㅋㅋ

반고흐#31555가 있는 금강 산맥에는 총 N개의 봉우리가 있고, 모든 봉우리마다 한 명의 활잡이가 서서 월식이 시작되기만을 기다리고 있다. 다만, 애석하게도, 천계에 맥도날드가 생겨 용들이 살이 찐 탓에 용들은 자신보다 낮은 봉우리에 서있는 적들만 처치할 수 있게 되었다. 또한 용들은 처음 출발한 봉우리보다 높은 봉우리를 만나면 그대로 공격을 포기하고 금강산자락에 드러누워 낮잠을 청한다고 한다. 봉우리의 높이는 모두 다르고 모든 용들은 오른쪽으로만 나아가며, 중간에 방향을 틀거나, 봉우리가 무너지거나 솟아나는 경우는 없다.

"달에 마구니가 끼었구나."

드디어 월식이 시작됐다! 과연 이들 활잡이 중 최고의 활잡이는 누구일까? 최고의 활잡이가 최대 몇 명의 적을 처치할 수 있는지 알아보자.

예제 입력 1 복사

```
7
6 4 10 2 5 7 11
```

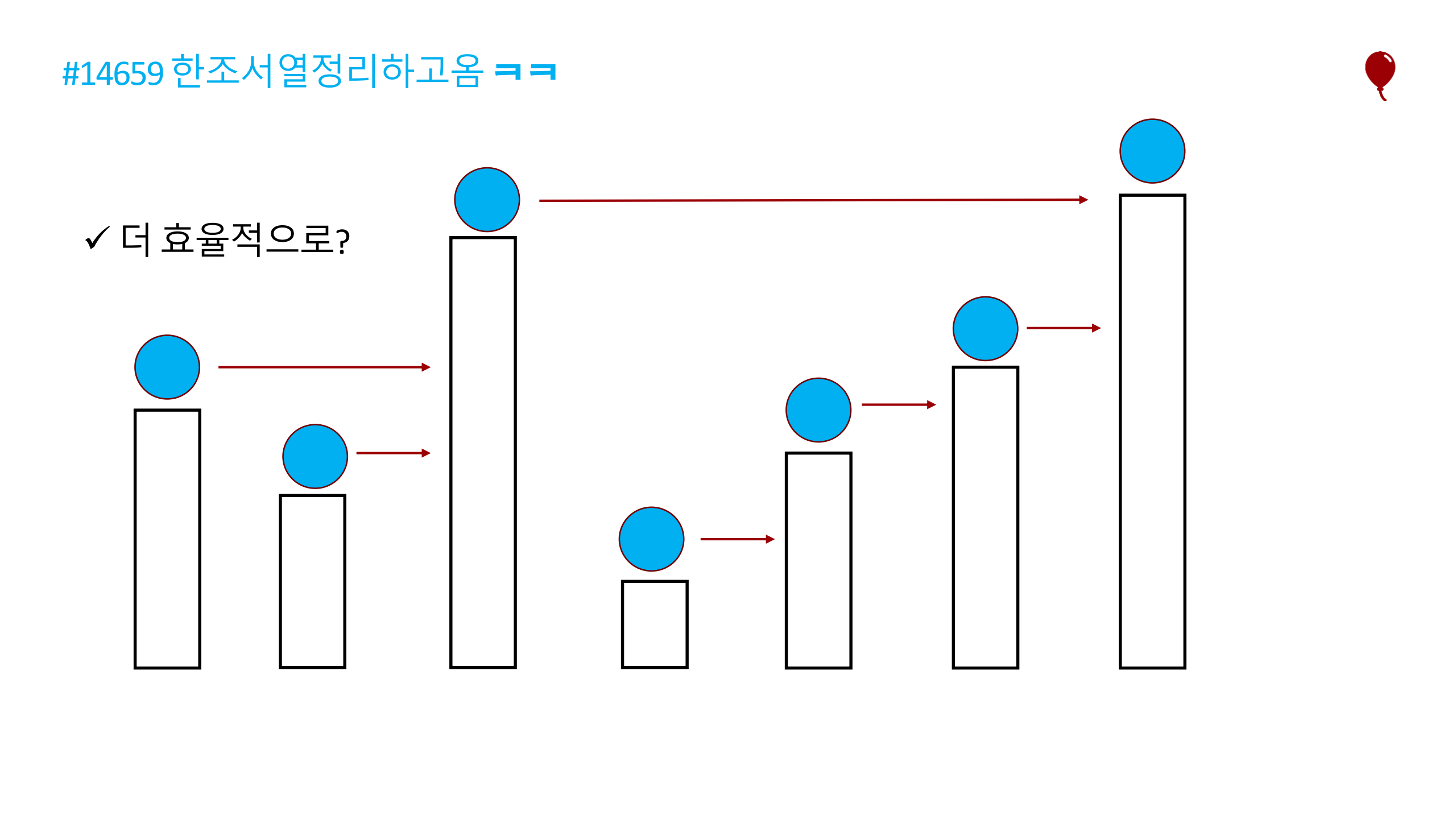
예제 출력 1 복사

```
3
```

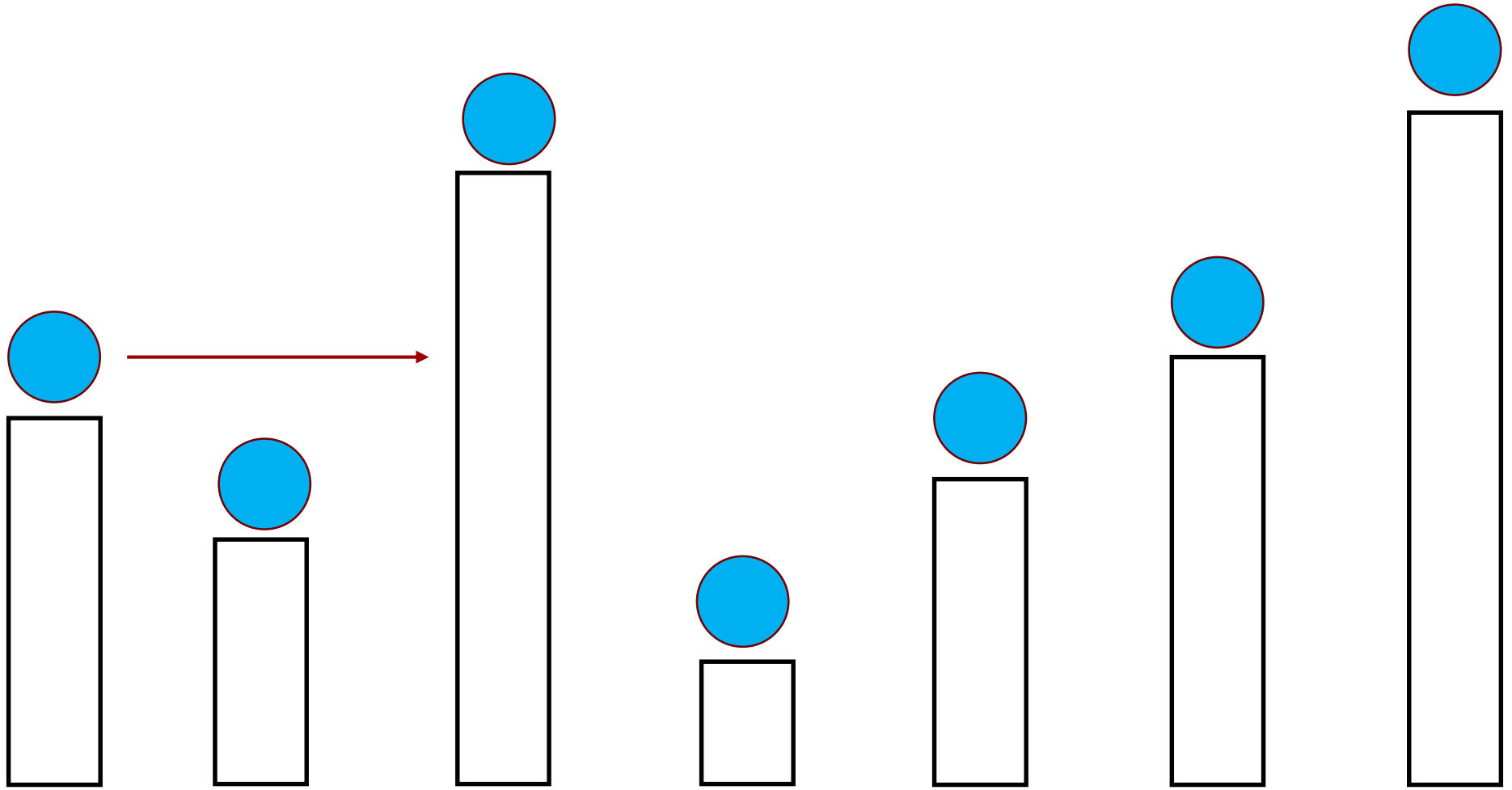
#14659 한조서열정리하고옴 ㅋㅋ

- 순진하게 접근해보자
- 각 봉우리의 한조들에 대해 활을 쏘며 자기 보다 높은 봉우리가 나올 때까지 체크
- 시간복잡도$O(n^2)$, 분명 비효율적인 부분이 있음!
- 자기 보다 뒤에 있는 한조인데, 봉우리 높이까지 작거나 같다면 볼 필요가 없다!

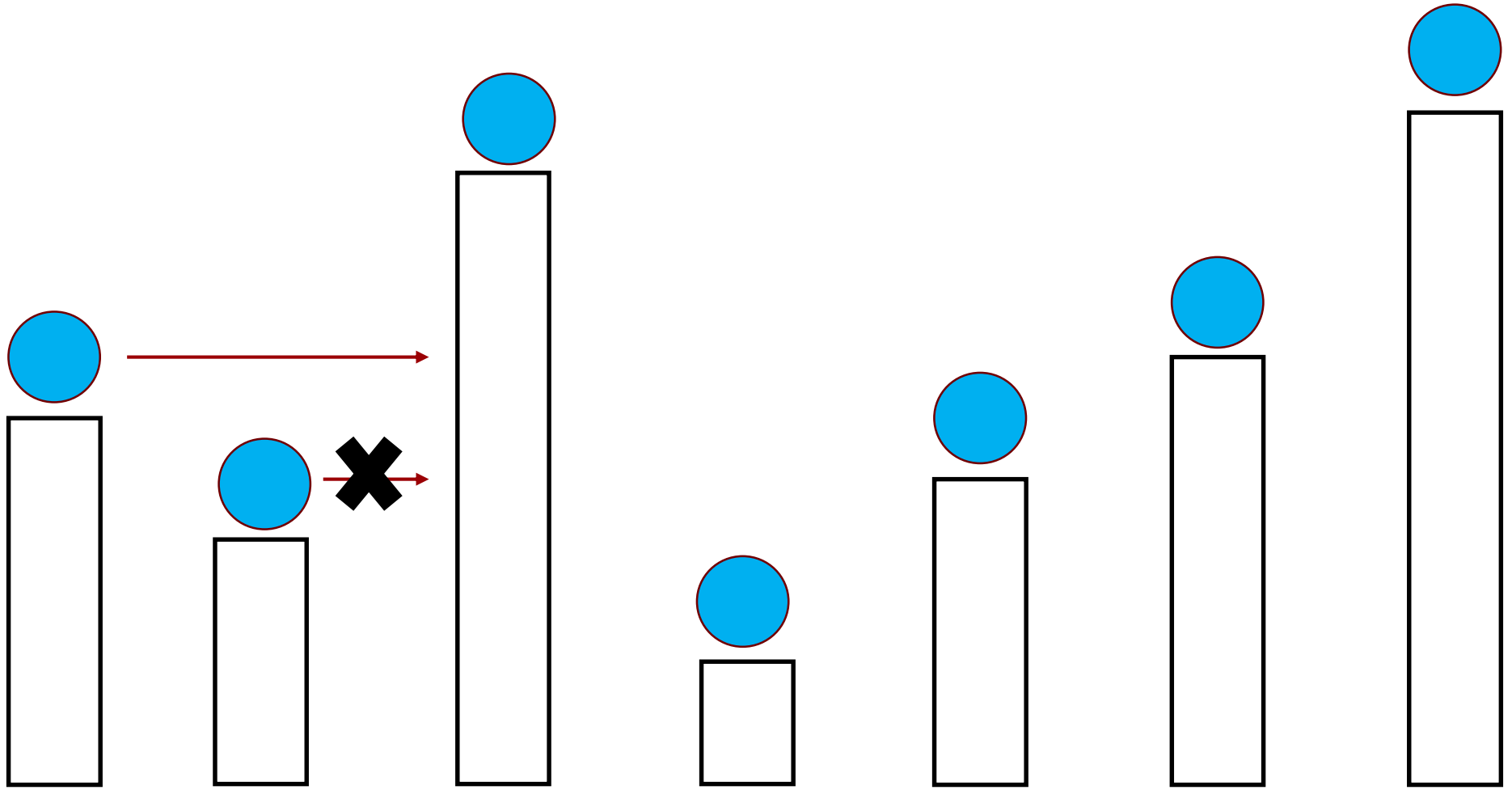
#14659 한조서열정리하고옴 ㅋㅋ



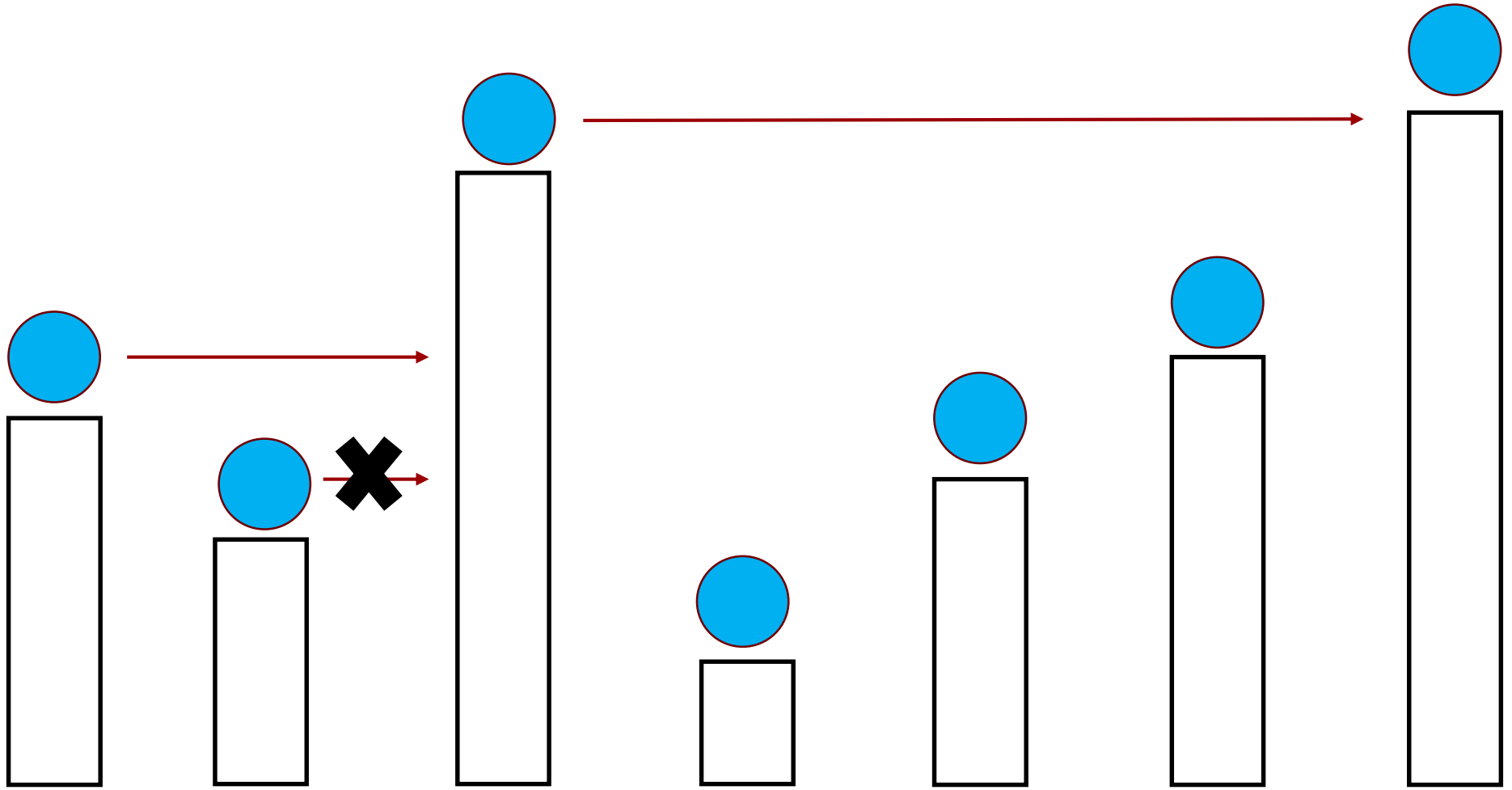
#14659 한조서열정리하고옴 ㅋㅋ



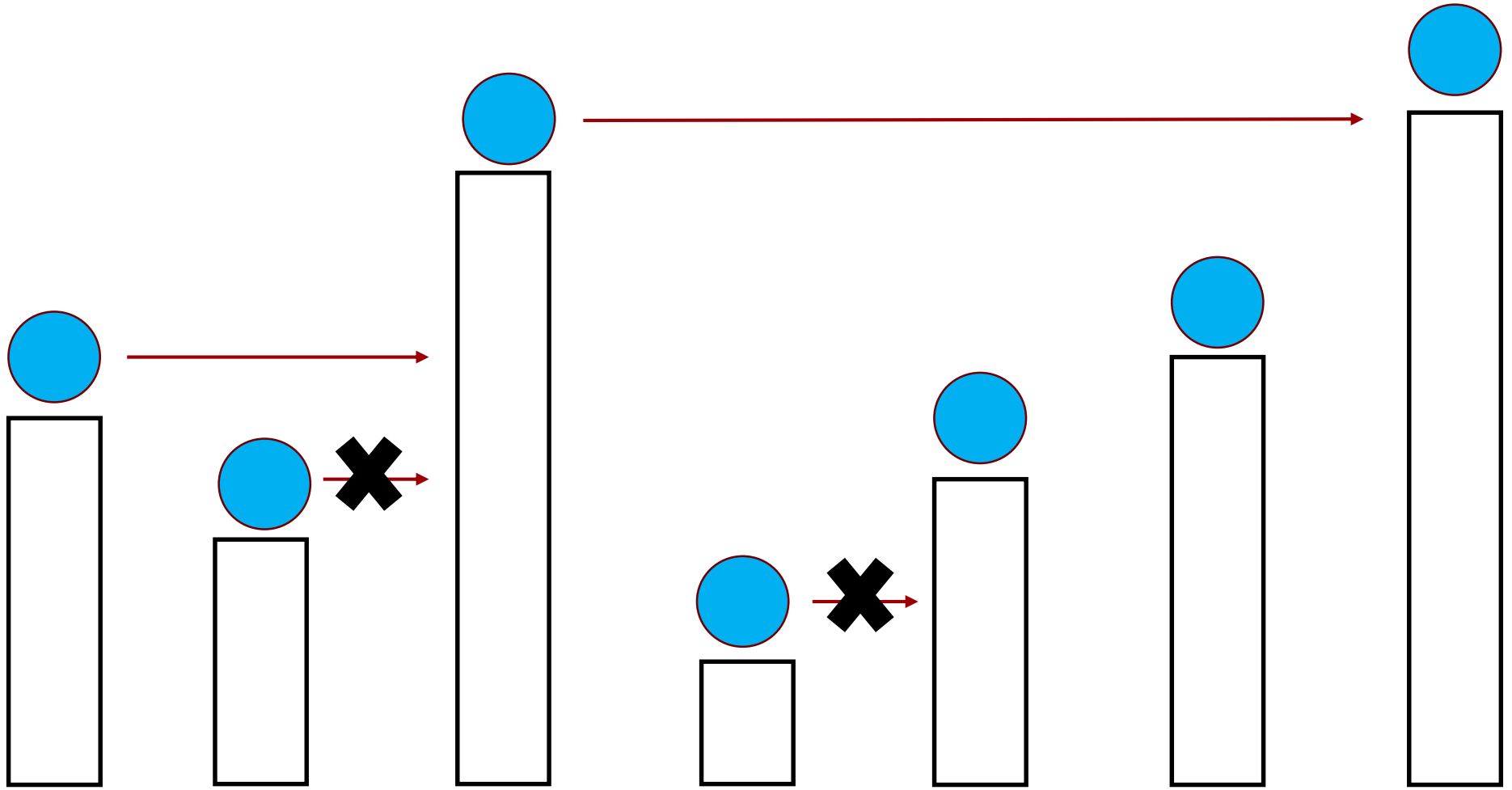
#14659 한조서열정리하고옴 ㅋㅋ



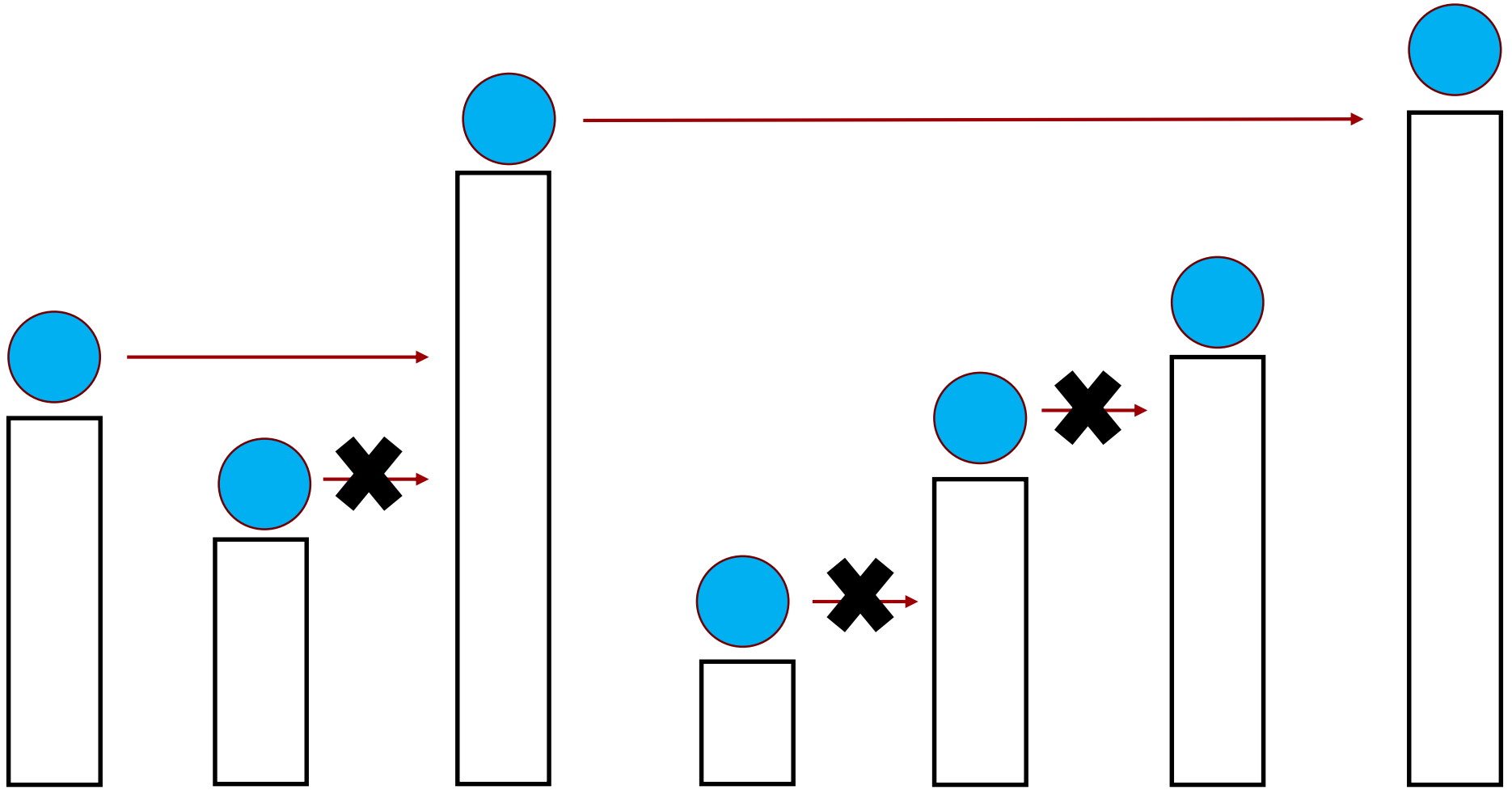
#14659 한조서열정리하고옴 ㅋㅋ



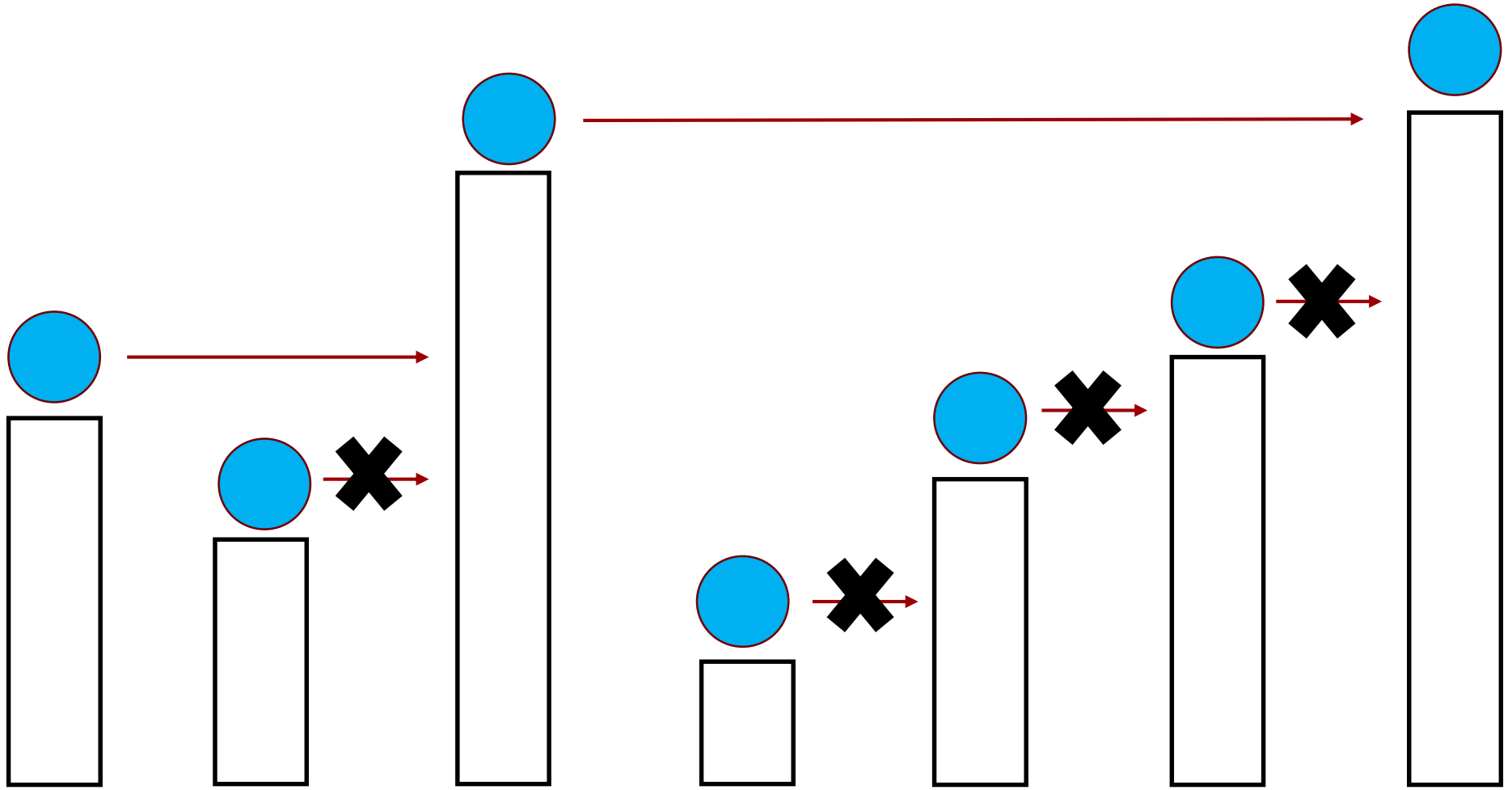
#14659 한조서열정리하고옴 ㅋㅋ



#14659 한조서열정리하고옴 ㅋㅋ



#14659 한조서열정리하고옴 ㅋㅋ



#14659 한조서열정리하고옴 ㅋㅋ

✓ 산봉우리의 MAX값을 저장할 변수,현재 MAX가 처치한 적을 저장할 변수 2개를 만든다.

✓ 순서대로 확인하면서 MAX보다 작은 산봉우리의 경우 적을 죽인다!

✓ MAX보다 큰 경우 MAX를 업데이트 하고, 처치한 적의 수를 0으로 초기화 한다.

✓ 지금까지의 처치한 적의 수의 최댓값을 출력 O(n)

Accepted

#14659 한조서열정리하고옴 ㅋㅋ code

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

using namespace std;
int n, a, cur_max, ans;

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0), cout.tie(0);
    cin >> n;
    vector<int>v;
    for (int i = 0; i < n; i++) {
        cin >> a;
        v.push_back(a);
    }
```

6: 변수 선언
전역변수는 0으로 초기하 되어 있음

12: int형 변수를 저장하는 vector선언

13: 각 한조들의 산봉우리 높이 입력

#14659 한조서열정리하고옴 ㅋㅋ code

```
17      a = 0;
18      for (int i = 0; i < n; i++) {
19          if (cur_max <= v[i]) {
20              cur_max = v[i];
21              ans = max(ans, a);
22              a = 0;
23          }
24          else a++;
25      }
26      ans = max(ans, a);
27      cout << ans;
```

17: 이제 변수 a는 현재까지 처치한 적의 수를 기록할 변수다.

19~24:
만약 현재의 봉우리 높이가 이전까지의 봉우리의 최대 높이보다 크거나 같다면 높이의 최댓값을 갱신하고, 현재까지 처치한 적의 수와 이전까지의 답 중 더 큰 값을 답으로 갱신해준다.
그렇지 않으면 처치한 적의 수를 한 명 늘린다

27: 지금까지 처치한 적의 수의 최댓값을 출력한다.

#2847 게임을 만든 동준이

학교에서 그래픽스 수업을 들은 동준이는 수업시간에 들은 내용을 바탕으로 스마트폰 게임을 만들었다. 게임에는 총 N개의 레벨이 있고, 각 레벨을 클리어할 때 마다 점수가 주어진다. 플레이어의 점수는 레벨을 클리어하면서 얻은 점수의 합으로, 이 점수를 바탕으로 온라인 순위를 매긴다. 동준이는 레벨을 난이도 순으로 배치했다. 하지만, 실수로 쉬운 레벨이 어려운 레벨보다 점수를 많이 받는 경우를 만들었다.

이 문제를 해결하기 위해 동준이는 특정 레벨의 점수를 감소시키려고 한다. 이렇게해서 각 레벨을 클리어할 때 주는 점수가 증가하게 만들려고 한다.

각 레벨을 클리어할 때 얻는 점수가 주어졌을 때, 몇 번 감소시키면 되는지 구하는 프로그램을 작성하시오. 점수는 항상 양수이어야 하고, 1만큼 감소시키는 것이 1번이다. 항상 답이 존재하는 경우만 주어진다. 정답이 여러 가지인 경우에는 점수를 내리는 것을 최소한으로 하는 방법을 찾아야 한다.

예제 입력 1 복사

```
4
5
3
7
5
```

예제 출력 1 복사

```
6
```

#2847 게임을 만든 동준이

✓문제를 간략화 해보자.

✓수열의 수를 하나 선택하여 1씩 감소시킬 수 있을 때,
증가수열(오름차순 정렬)이 되도록 하는 최소의 감소 횟수를 구하자!

✓감소연산만 사용할 수 있다는 것이 핵심

✓맨 마지막 원소에 감소 연산을 사용하면 어떻게 될까?

#2847 게임을 만든 동준이

- 수열이 각 원소를 $a_0, a_1, a_2, a_3, \ldots, a_{n-1}$ 이라고 하자

✓ a_{n-1}을 감소 시키지 않았을 때의 최적해를 x라 하자.

✓ **a_{n-1} 을 1 감소 시켰을 때, 최적해는 x보다 크다**

연산 이후 0부터 n-2번까지 수열의 값들을 $b_0, b_1, b_2, b_3, \ldots, b_{n-2}$ 라 하자.

$b_0 < b_1 < b_2 < b_3 \ldots < b_{n-2} < a_{n-1}$ 을 만족하는 집합을 A

$b_0 < b_1 < b_2 < b_3 \ldots < b_{n-2} < (a_{n-1} - 1)$ 을 만족하는 집합을 B라고 할때

$B \subset A$ 이다.

따라서 A에서의 최적해 <= B에서의 최적해 Q.E.D

#2847 게임을 만든 동준이

✓ 마지막 원소를 수열에서 제외하자.

✓ 남은 수열의 마지막 원소를 min(제외한 원소-1,마지막 원소) 로 바꾼다.

✓ 만약 그 보다 작아지게 된다면, 이전에 증명한 것에 의해 최적해가 아니다.

✓ 위 과정을 n번 반복한다.

Accepted

#2847 게임을 만든 동준이 code

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

using namespace std;
int n, a, ans;
int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0), cout.tie(0);
    cin >> n;
    vector<int>v;
    for (int i = 0; i < n; i++) {
        cin >> a;
        v.push_back(a);
    }
```

12~14: n개의 수열을 입력받는다

#2847 게임을 만든 동준이 code

```
16      for (int i = n - 2; i >= 0; i--) {
17          if (v[i + 1] <= v[i])
18              ans += v[i] - (v[i + 1] - 1);
19          v[i] = min(v[i], v[i + 1] - 1);
20      }
21      cout << ans;
22  }
```

17 ~ 18: if문 조건은 만약 제외한 원소보다 마지막 원소가 더 크다면 성립한다. 현재 원소를 제외한 원소-1 로 바꿔야 하므로 사용해야 하는 감소연산의 횟수만큼 ans에 더한다.

19: 마지막 원소를 변화시킨다.

#20044 Project Teams

코딩 프로젝트 수업을 가르치는 수찬이는 프로젝트 팀을 가능하면 공정하게 구성하려고 한다. 프로젝트 팀 하나는 두 명의 학생으로 구성되는데, 각 학생들의 코딩 역량은 모두 다르다. 각 학생은 한 팀의 팀원이어야 한다. 공정성을 높이기 위해 수찬이는 팀원 코딩 역량의 합을 최대한 일정하게 유지하려고 한다. 학생들이 코딩 역량이 주어졌을 때 수찬이가 팀을 구성하는데 도움이 되는 프로그램을 작성하라.

문제를 간단하게 하기 위해, 학생 수는 $2n$이라 가정하자 (n은 양의 정수). 각 학생 s_i의 코딩 역량은 양의 정수 $w(s_i)$로 나타내면 한 i번째 팀 G_i의 코딩 역량은 $w(G_i) = \Sigma_{s \in G_i} w(s)$로 나타낼 수 있다. 작성할 프로그램 목적은 $S_m = \min\{w(G_i) \mid 1 \leq i \leq n\}$이 최대화되도록 n개의 팀 $G_1, G_2, \ldots, G_n$ 을 구성하고 이때 S_m을 출력하는 것이다.

예를 들어, 학생들의 코딩 역량이 {1, 7, 5, 8}로 주어졌다면 (8, 1), (7, 5)로 2개의 조를 짤 수 있으며 프로그램은 9를 출력해야 한다.

예제 입력 1 복사

```
2
1 7 5 8
```

예제 출력 1 복사

```
9
```

우리는 팀의 최솟값이 최대가 되도록 팀을 짜주어야 함.

- 정렬한 후 수열이 각 원소를 $a_0, a_1, a_2, a_3, \ldots, a_{2n-1}$ 이라고 하자

greedy 하게 접근

- 정답은 $(a_0 + a_{2n-1}), (a_1 + a_{2n-2}), \ldots (a_{n-1} + a_n)$ 중 min값이 될거야

예제 접근

✓ 1번 정답: min((1+8),(5+7));

✓ 2번 정답: min((1+9),(2+7),(3+5));

#20044 Project Teams

- 크기순으로 정렬했을 때 작은 n 개의 원소를 그룹 A, 큰 n 개의 원소를 그룹 B로 묶자.

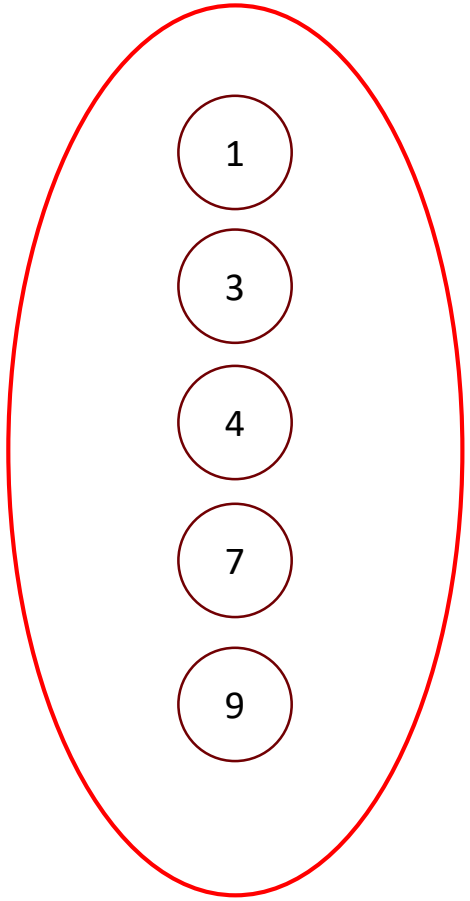
반드시 A원소 하나 , B원소 하나를 팀으로 묶어 주어야 한다.
아니라면 같은 그룹의 원소끼리 팀이 되는 경우가 생긴다. (A,B그룹 모두에 반드시 생긴다)

그런데 (a_x, a_y) (b_x, b_y) 가 되는 것 보다는 팀을 섞는 게 항상 이득이다!

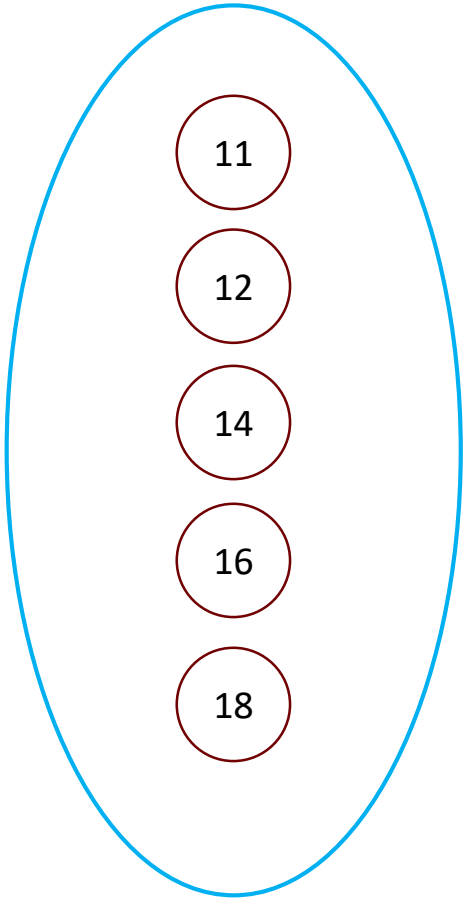
따라서 모든 팀은 A원소 하나, B원소 하나로 이루어져 있어야 최적이다.

우선 이런 식으로 원소를 정렬해서 묶어주자

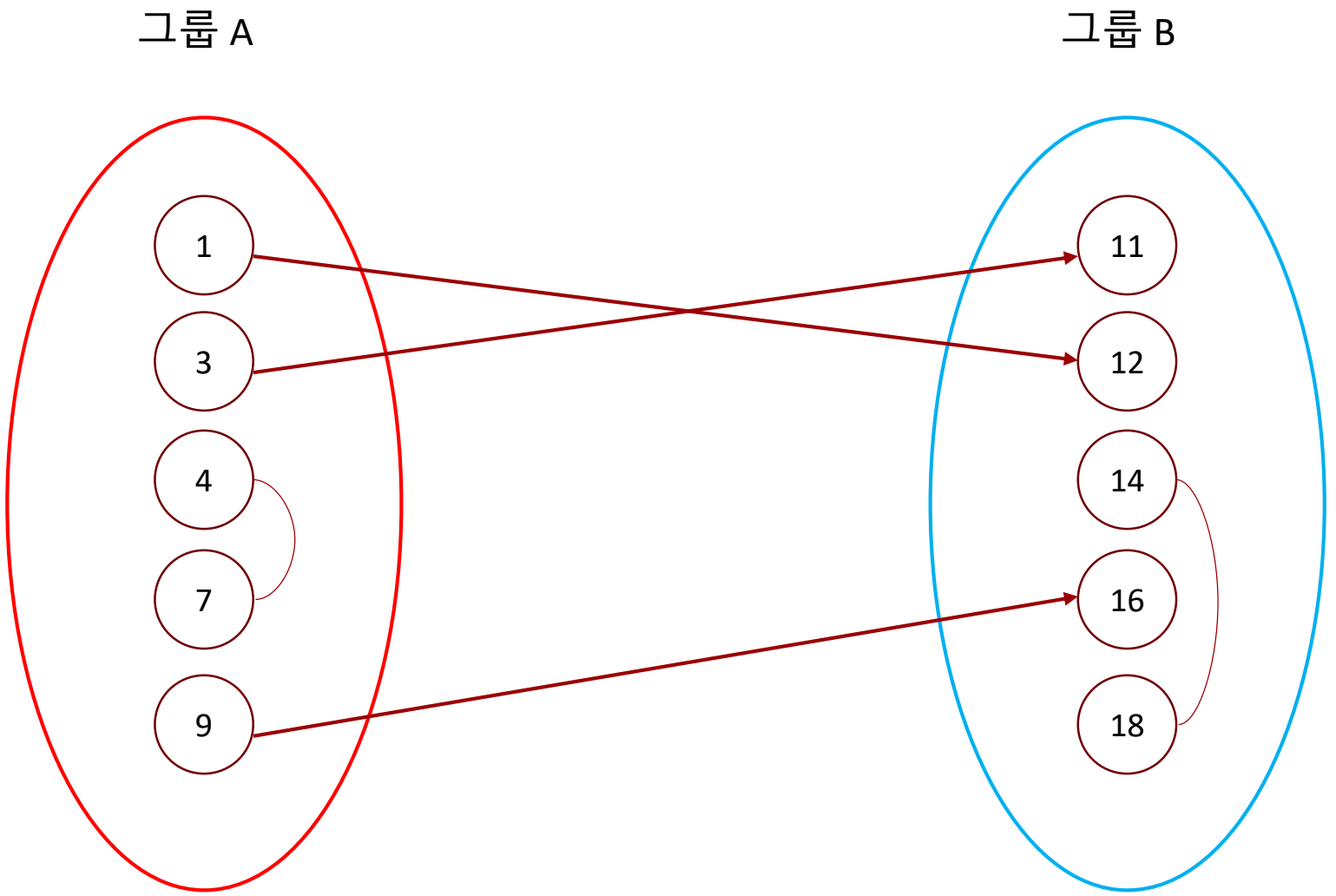
그룹 A



그룹 B

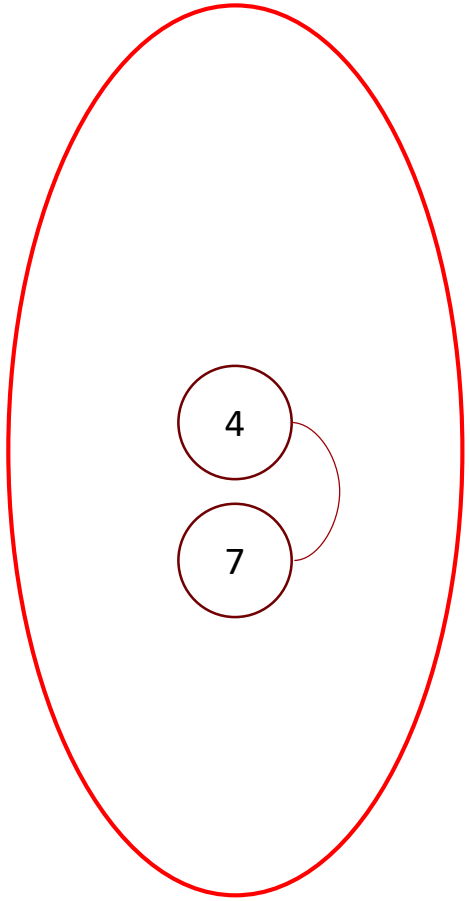


같은 팀에 같은 그룹인 경우가 존재할때

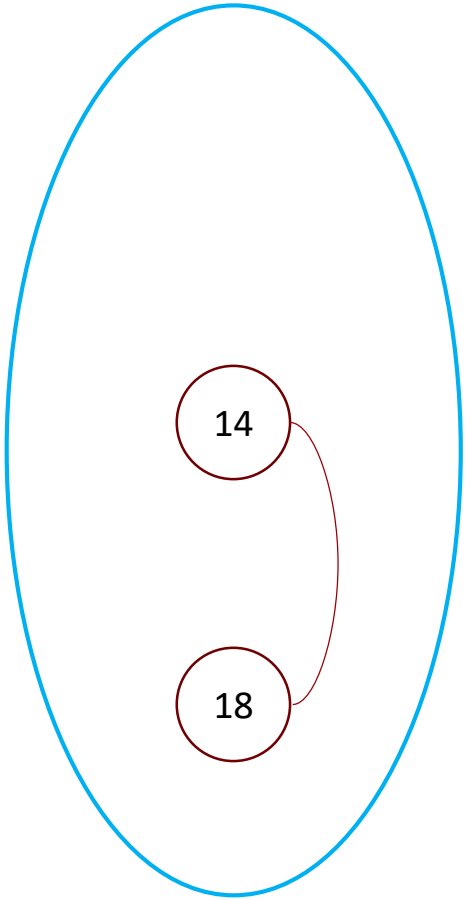


그러한 팀들만 살펴보자. 이때 최솟값 11

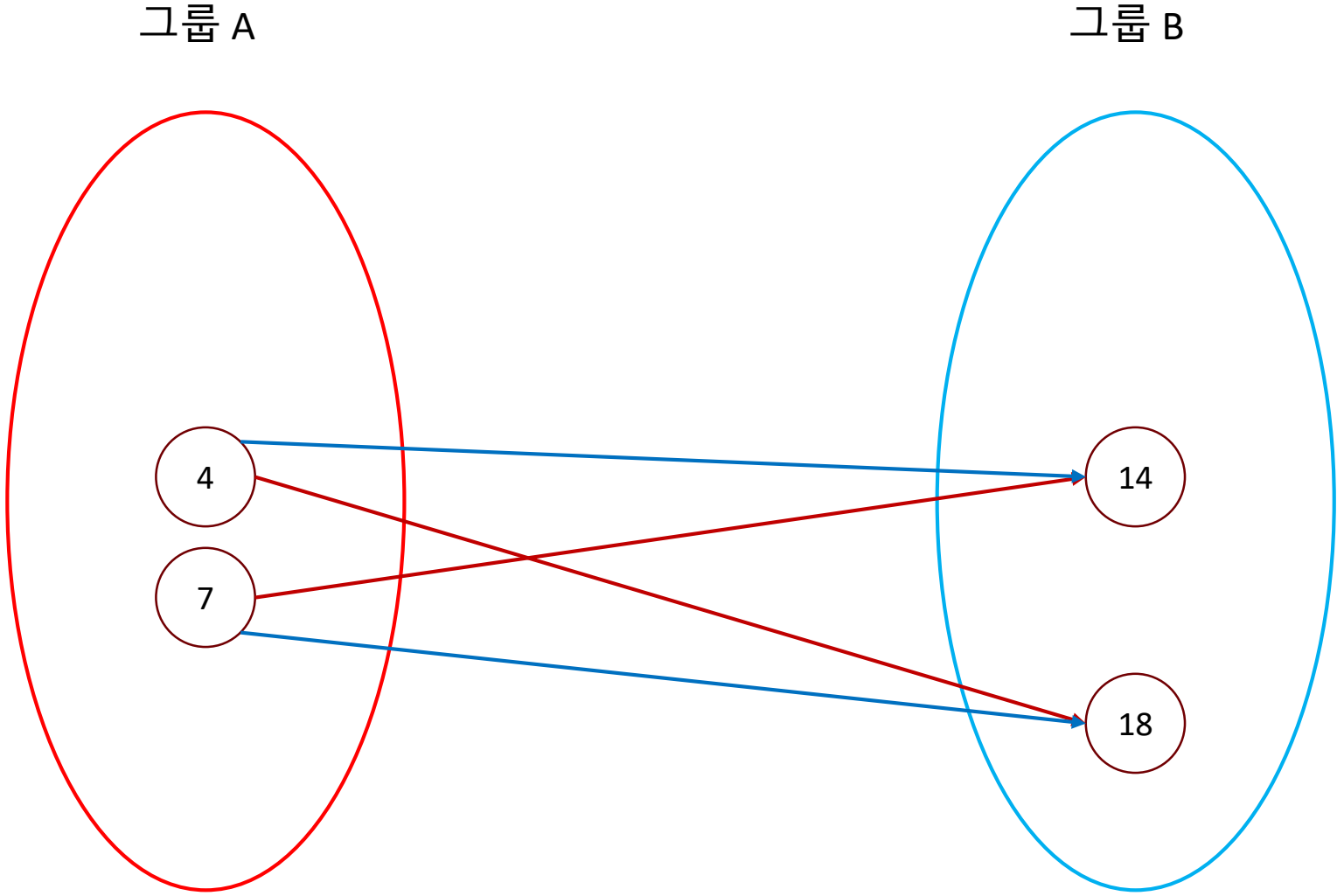
그룹 A



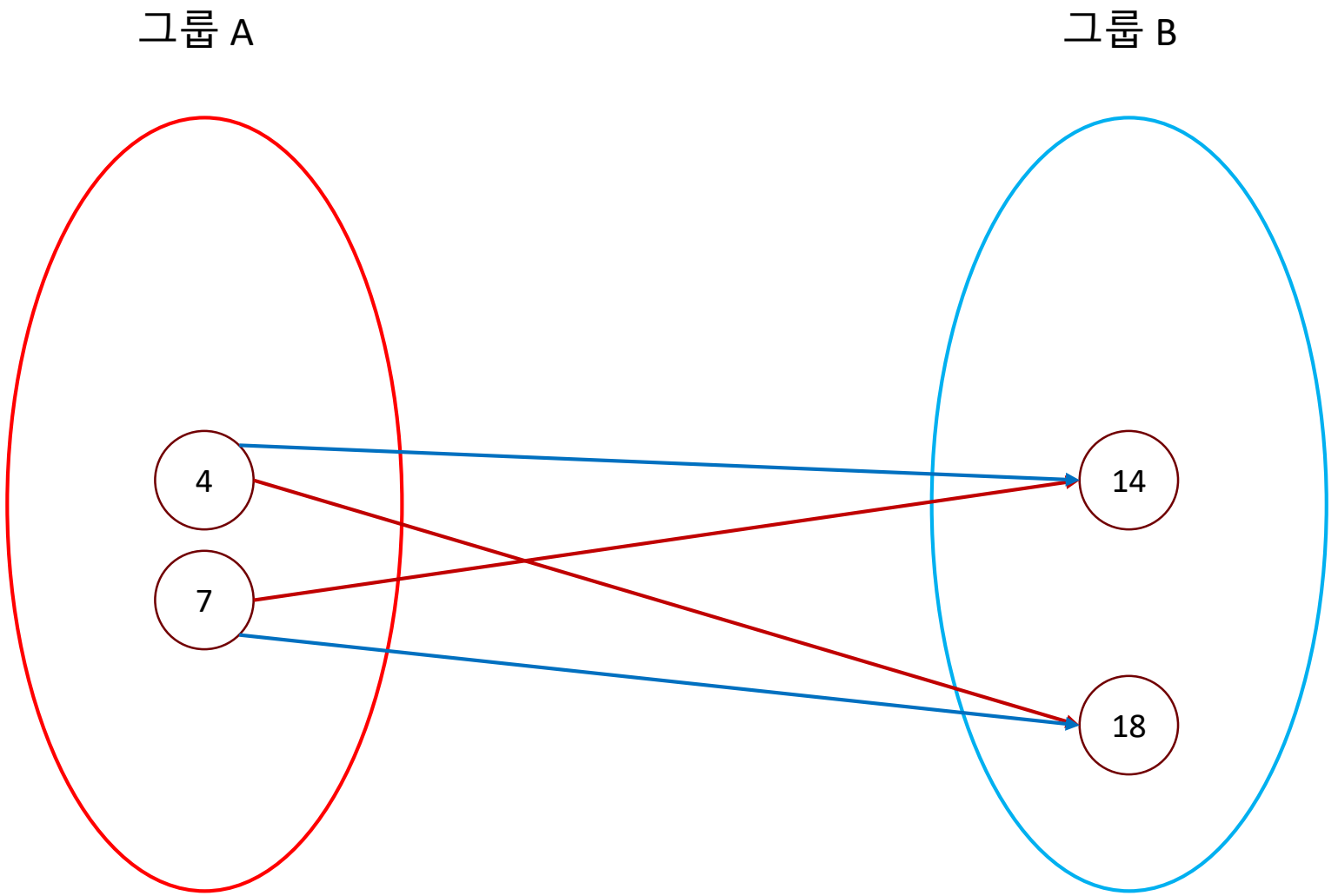
그룹 B



빨간색일때 최솟값은 21, 파란색일때는 18이다.



따라서 같은 그룹끼리 다른 팀이 되게 하는 것이 최적이다



그런데 어떤 팀 (a_x, b_x) (a_y, b_y) 에 대해,
만약 $(a_x < a_y)$이고, $(b_x < b_y)$ 라면 이 구성보다는
(a_x, b_y), (a_y, b_x) 이게 반드시 이득이다. (최솟값을 최대화 해야 하므로)

- 그런데 우리가 생각했던 이러한 $(a_0 + a_{2n-1})$, $(a_1 + a_{2n-2})$, … $(a_{n-1} + a_n)$ 구성이 아니라면
- 반드시 이러한 경우가 생긴다. 따라서 그리디한 접근이 옳다.

#20044 Project Teams code

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

int n, a;
vector<int> v;
int main() {
    cin >> n;
    n *= 2;
    for (int i = 0; i < n; i++) {
        cin >> a;
        v.push_back(a);
    }
    sort(v.begin(), v.end());
    a = 1234567891;
    for (int i = 0; i < n; i++) {
        a = min(a, v[i] + v[n - 1 - i]);
    }
    cout << a;
}
```

10: 2n개 의 input이 들어옴

15: sort함수를 이용하여 오름차순으로 정렬

16: 문제에서 주어진 입력값으로 절대 나올 수 없는 큰 값으로 초기화

17~19: 그리디하게 a 값을 계속 갱신해줌

추천 문제 셋

필수

14659	2 한조서열정리하고옴ㅋㅋ
2847	4 게임을 만든 동준이
20044	4 Project Teams
1931	2 회의실배정

Easy && normal

2839	1 설탕 배달
4796	5 캠핑
11399	3 ATM
1080	2 행렬
11047	1 동전 0
19639	4 배틀로얄
12931	4 두 배 더하기
19582	4 200년간 폐관수련했더니 PS 최강자가 된 건에 대하여
1339	4 단어 수학
13904	3 과제
15553	4 난로

Hard && challenging

16288	3 Passport Control
1736	2 쓰레기 치우기
19590	2 비드맨
1700	2 멀티탭 스케줄링
17954	1 투튜브
11092	1 Safe Passage