

2024 겨울학기 동국대학교SW역량강화캠프

14일차. 그리디

● 예시

▶ 78000원을 오만원권, 만원권, 오천원권, 천원권으로 만들 때, 가장 적은 수의 지폐를 이용하여 만드는 방법은 무엇인가?

오만원권 1개, 만원권 2개, 오천원권 1개, 천원권 3개

▶ 큰 액수의 지폐를 최대한 많이 사용한다.

지폐의 액수가 오만원권, 삼만원권, 오천원권, 천원권으로 만들 때에도 같은 전략이 통할까?

● 그리디 알고리즘

- ▶ 탐욕 알고리즘 / 욕심쟁이 알고리즘
- ▶ 현재 단계에서 가장 최선의 선택을 하는 알고리즘
- ▶ 모든 경우에서 통하지는 않으며, 사용할 때 증명이 필요
- ▶ 주로 예시를 통한 관찰과 규칙 찾기로 문제 해결
- ▶ 가장 큰 ~, 가장 빠른 ~, 가장 긴 ~ 부터 ~ 한다.

● 배치하기 (2709)

| 문제

N개의 정수로 이루어진 배열 A와 B가 주어질 때, X는 아래와 같이 정의된다.

$$X = A[0]*B[0] + \dots + A[N-1]*B[N-1]$$

배열 A를 적당히 재배열해서 X의 값을 최소값으로 만들려고 한다.

X의 가능한 최소값을 출력해보자.

| 입력

첫째 줄에는 50 이하의 양의 정수 N이 주어진다.

두번째 줄에는 배열 A가, 세 번째 줄에는 배열 B가 N개의 자연수로 주어진다.

| 출력

첫째 줄에 X의 최솟값을 출력한다.

▶ 예시 2~3개를 통해 관찰해봅시다.

3	1	2
1	2	3



3	2	1
1	2	3

3	1	2	4
1	2	3	4



4	3	2	1
1	2	3	4

1	5	7	3
2	6	4	8



7	3	5	1
2	6	4	8



▶ B 배열 역시 재배열 되어도 답에 영향을 주지 않는다.

1	5	7	3
2	6	4	8

7	5	3	1
2	4	6	8

3	1	2
1	2	3

3	2	1
1	2	3

3	1	2	4
1	2	3	4

4	3	2	1
1	2	3	4

▶ 예측 : A 배열을 오름차순, B 배열을 내림차순으로 정렬했을 때, 최소값이 나온다

증명: 귀류법을 이용합니다.

A 배열을 오름차순 정렬했을 때, 답을 최소로 하는 B 배열에 내림차순이 아닌 부분이 존재한다고 해봅시다.

최선의 답에서 $A[i] < A[i+1]$ 이면서 $B[i] < B[i+1]$

A[1]	A[2]	...	A[i]	A[i+1]	...	A[N]
B[1]	B[2]	...	B[i]	B[i+1]	...	B[N]

현재의 답 = $\dots + A[i] * B[i] + A[i+1] * B[i+1] + \dots$

A[1]	A[2]	...	A[i]	A[i+1]	...	A[N]
B[1]	B[2]	...	B[i]	B[i+1]	...	B[N]

개선된 답 = $\dots + A[i] * B[i+1] + A[i+1] * B[i] + \dots$

A[1]	A[2]	...	A[i]	A[i+1]	...	A[N]
B[1]	B[2]	...	B[i+1] ↔ B[i]		...	B[N]

현재의 답 - 개선된 답 = $A[i] * B[i] + A[i+1] * B[i+1] - A[i] * B[i+1] - A[i+1] * B[i]$

= $(A[i] - A[i+1]) * (B[i] - B[i+1]) > 0$

개선된 답 < 현재의 답 이므로 현재의 답이 최선이 아님. 따라서 모순!

- ▶ 사실 이렇게까지 명확한 증명은 Too Much..
- ▶ 예시를 통해 얻은 직감이나, 특정 선택이 언제나 다른 선택들보다 더 나은 선택이라는 느낌만으로도 충분합니다.
- ▶ Proof by AC

```
int N = Integer.parseInt(br.readLine());

ArrayList<Integer> A = new ArrayList<>();
ArrayList<Integer> B = new ArrayList<>();

st = new StringTokenizer(br.readLine());
for(int i=0; i<N; i++) A.add(Integer.parseInt(st.nextToken()));

st = new StringTokenizer(br.readLine());
for(int i=0; i<N; i++) B.add(-Integer.parseInt(st.nextToken()));

Collections.sort(A);
Collections.sort(B);

long X=0;
for(int i=0; i<N; i++) X -= A.get(i) * (long)B.get(i);

System.out.print(X);
```

● 체육대회 (337)

| 문제

도원이가 다니는 백금고등학교의 1학년은 A반과 B반 2개의 반으로 이루어져 있다.

이번에 백금고등학교에서는 체육대회를 개최하는데, 많은 참여를 위해 1학년은 A반과 B반의 1대1 줄다리기를 진행하기로 하였다.

A반과 B반은 둘 다 n 명의 학생들로 이루어져 있는데, 각 반은 이 n 명의 학생을 1번부터 n 번까지 번호를 붙혀, 같은 번호의 학생끼리 1대1 줄다리기를 진행하게 된다.

A반에 다니고 있는 도원이는 인맥을 활용하여 B반의 1번 학생부터 n 번 학생까지 줄을 당기는 힘이 얼마인지 알아내었다..!

도원이는 A반 학생들의 줄을 당기는 힘이 얼마인지도 전부 알고 있기 때문에, 번호를 붙힐 때 B반의 학생들을 최대한 많이 이길 수 있도록 하려고 한다.

이 때 도원이가 B반 학생을 최대 몇 명 이기도록 A반 학생을 배치할 수 있을 지 출력해보자. (모든 학생의 당기는 힘은 전부 다르다!)

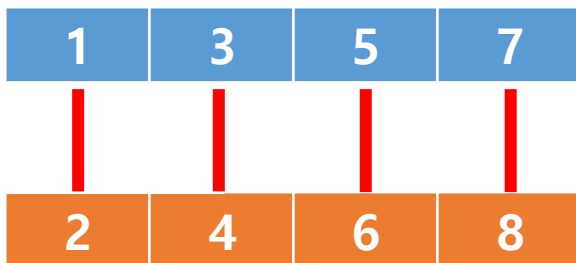
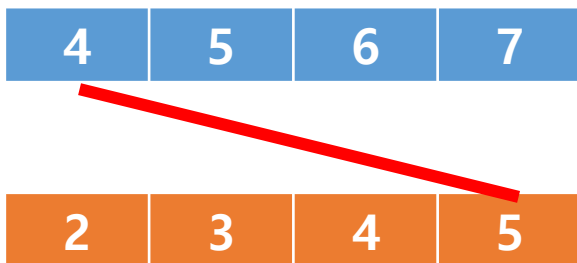
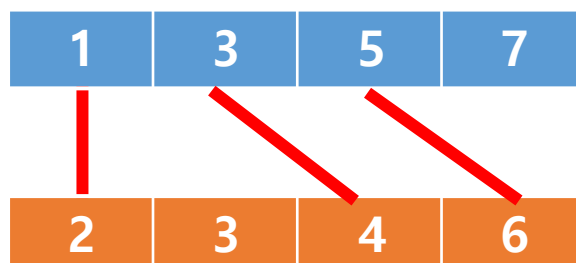
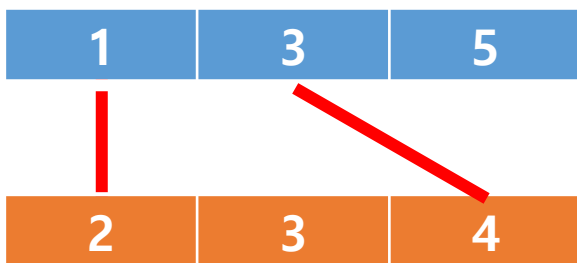
| 입력

첫째 줄에 A반과 B반의 학생의 수 n 이 주어진다 ($1 \leq n \leq 100,000$)

둘째 줄에 B반의 1번 학생부터 n 번 학생까지의 당기는 힘이 주어진다.

셋째 줄에 A반의 학생 n 명의 당기는 힘이 주어진다. (당기는 힘은 100만 이하의 자연수로 주어진다)

▶ 예시 3~4 개를 통해 관찰



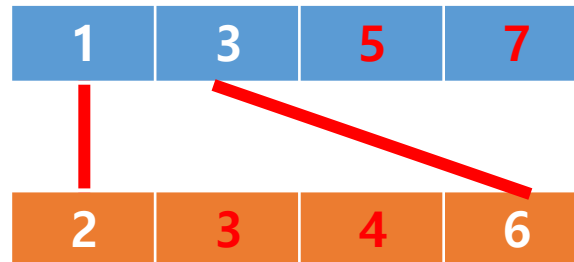
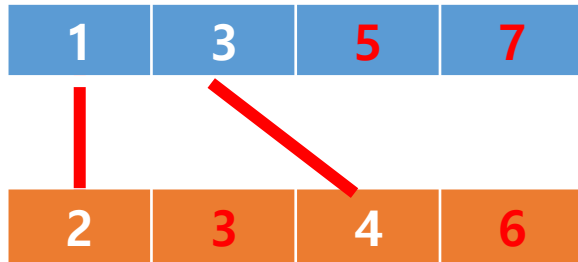
▶ 나타나는 공통점

B반 학생들 중 K명을 이긴다면,
그 K명은 B반 학생들 중 가장 약한 K명이다. (B반 학생들을 약한 순서대로 이기는 것이 최선이다.)

B반 학생을 이길 때, 이길 수 있는
가장 약한 힘으로 이기는 것이 최선이다.

▶ 예측 : B반 학생들을 힘이 약한 순서대로 이기는데, 이길 때 최소한의 힘으로 이기는 것이 최선이다.

이길 때 최소한의 힘으로 이기는 것이 최적인 것은 상태의 비교 우위로 증명할 수 있습니다.



이길 때 최소한의 힘으로 이기는 것은 왼쪽에서부터 탐색하여 처음으로 이길 수 있는 사람을 탐색

단, 아래 배열에서도 선택을 오름차순으로 한다는 사실을 이용하면 빠르게 해결 가능

```
st = new StringTokenizer(br.readLine());
for(int i=0; i<N; i++) B.add(Integer.parseInt(st.nextToken()));

st = new StringTokenizer(br.readLine());
for(int i=0; i<N; i++) A.add(Integer.parseInt(st.nextToken()));

Collections.sort(A);
Collections.sort(B);

int j=0, cnt=0;
for(int i=0; i<N; i++){
    while(j<N && A.get(j) <= B.get(i)) j++;
    if(j == N) break;
    cnt++;
    j++;
}

System.out.print(cnt);
```

● 밧줄 (7)

문제

$N(1 \leq N \leq 100,000)$ 개의 로프가 있다. 이 로프를 이용하여 이런 저런 물체를 들어올릴 수 있다. 각각의 로프는 그 굵기나 길이가 다르기 때문에 들 수 있는 물체의 중량이 서로 다를 수도 있다.

하지만 여러 개의 로프를 병렬로 연결하면 각각의 로프에 걸리는 중량을 나눌 수 있다. k 개의 로프를 사용하여 중량이 w 인 물체를 들어올릴 때, 각각의 로프에는 모두 고르게 w/k 만큼의 중량이 걸리게 된다.

각 로프들에 대한 정보가 주어졌을 때, 이 로프들을 이용하여 들어올릴 수 있는 물체의 최대 중량을 구해내는 프로그램을 작성하시오. 모든 로프를 사용해야 할 필요는 없으며, 임의로 몇 개의 로프를 골라서 사용해도 된다. 단, 각각의 로프는 한 개씩만 존재한다.

입력

첫째 줄에 정수 N 이 주어진다. 다음 N 개의 줄에는 각 로프가 버틸 수 있는 최대 중량이 주어진다. 이 값은 10,000을 넘지 않는다.

출력

첫째 줄에 답을 출력한다.

▶ 예시 2~3개를 통해 관찰

15	4	10	4	→	15	4	10	4	최대 중량 20
----	---	----	---	---	----	---	----	---	----------

15	6	10	7	→	15	6	10	7	최대 중량 24
----	---	----	---	---	----	---	----	---	----------

15	6	10	9	→	15	6	10	9	최대 중량 27
----	---	----	---	---	----	---	----	---	----------

K개의 밧줄을 고른다면, 가장 최대 중량이 큰 K개의 밧줄을 고른다.
 그 때의 전체 최대 중량은 $K * (\text{가장 최대 중량이 큰 K개의 밧줄의 최대 중량의 최소값})$

15	9.5	6	5
----	-----	---	---



15	9.5	6	5
----	-----	---	---

중량 15

15	9.5	6	5
----	-----	---	---

중량 19

15	9.5	6	5
----	-----	---	---

중량 18

15	9.5	6	5
----	-----	---	---

중량 20

```
int N = Integer.parseInt(br.readLine());

ArrayList<Integer> w = new ArrayList<>();
for(int i=0; i<N; i++) w.add(-Integer.parseInt(br.readLine()));

Collections.sort(w);

int ans=0;
for(int i=0; i<N; i++) ans = Math.max(ans, -(i+1)*w.get(i));

System.out.print(ans);
```

● 대표 자연수 (658)

| 문제

정보초등학교의 연아는 여러 개의 자연수가 주어졌을 때, 이를 대표할 수 있는 대표 자연수에 대하여 연구하였다. 그 결과 어떤 자연수가 다음과 같은 성질을 가지면 대표 자연수로 적당할 것이라고 판단하였다.

“대표 자연수는 주어진 모든 자연수들에 대하여 그 차이를 계산하여 그 차이들 전체의 합을 최소로 하는 자연수이다.”

예를 들어 주어진 자연수들이 [4, 3, 2, 2, 9, 10]이라 하자. 이때 대표 자연수는 3 혹은 4가 된다. 왜냐하면 (4와 3의 차이) + (3과 3의 차이) + (2와 3의 차이) + (2와 3의 차이) + (9와 3의 차이) + (10과 3의 차이) = $1+0+1+1+6+7 = 16$ 이고, (4와 4의 차이) + (3과 4의 차이) + (2와 4의 차이) + (2와 4의 차이) + (9와 4의 차이) + (10과 4의 차이) = $0+1+2+2+5+6 = 16$ 으로 같으며, 이 두 경우가 차이들의 합을 최소로 하기 때문이다. 비교를 위하여 평균값인 5의 경우를 생각하여 보면, (4와 5의 차이) + (3과 5의 차이) + (2와 5의 차이) + (2와 5의 차이) + (9와 5의 차이) + (10과 5의 차이) = $1+2+3+3+4+5 = 18$ 로 위의 두 경우보다 차이들의 합이 더 커짐을 볼 수 있다.

연아를 도와서 위의 성질을 만족하는 대표 자연수를 구하는 프로그램을 작성하시오.

▶ 예시 2~3개를 통해 관찰

2	5	3	4	6
---	---	---	---	---

1	5	3	6	100
---	---	---	---	-----

2	3	4	5	6
10	7	6	7	10

1	3	5	6	100
110	104	102	103	385

▶ 입력된 순서는 상관없다. → 정렬해서 보자



$$A[k] \leq x < A[k+1]$$

$|A[1] - x| + \dots + |A[k] - x| + |A[k+1] - x| + \dots + |A[N] - x|$ 의 최소를 구한다.

만약 x 가 1 증가하면 $+(k) - (N-k) = 2*k - N$ 만큼 변한다.

즉 $2*k - N < 0$ 이라면 x 를 증가시키는 것이 값을 감소시킨다.

$k < (N/2)$ 까지 x 를 증가, $x = A[(N+1)/2]$

$$A[(N-1)/2] \leq x < A[(N-1)/2 + 1]$$

$$x == A[(N-1)/2 + 1]$$

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
StringTokenizer st;

int N = Integer.parseInt(br.readLine());

ArrayList<Integer> arr = new ArrayList<>();
st = new StringTokenizer(br.readLine());
for(int i=0; i<N; i++) arr.add(Integer.parseInt(st.nextToken()));

Collections.sort(arr);

System.out.print(arr.get((N-1)/2));
```

● 파일 합치기 (4152)

문제

윌리의 직박구리 폴더에는 N개의 파일이 있다. 윌리는 이 파일들을 전부 합쳐 하나의 파일로 만드려고 한다. 파일을 합칠 때에는 두 파일을 하나로 합치는 것을 반복해야 한다. 합칠 때에는 두 파일의 크기를 더한 것만큼의 시간이 걸린다.

놀랍게도, 파일을 고르는 순서에 따라서 비교 횟수가 달라진다. 예를 들어 크기가 10, 20, 40인 파일이 있다면 10과 20을 합친 뒤, 합친 30과 40을 합친다면 $(10 + 20) + (30 + 40) = 100$ 의 시간이 필요하다. 그러나 10과 40을 합친 뒤, 합친 50과 20을 합친다면 $(10 + 40) + (50 + 20) = 120$ 의 시간이 필요하므로 덜 효율적인 방법이다.

N개의 파일 크기가 주어질 때, 필요한 시간의 최솟값을 구해보자.

파일 합치기 많이 틀리는 예제

4개의 파일 크기가 50,30,40,60일 경우 < 360

4개의 파일 크기가 30 60 70 80인 경우 < 480

▶ 예시 2~3개를 통해 관찰



가장 작은 2개의 병합을 반복한다?

- ▶ 가장 작은 2개의 병합을 반복한다.
- ▶ 엄밀한 증명은 Skip! Proof by AC
- ▶ 가장 작은 2개를 합친다 → 우선순위 큐를 이용 (TreeSet도 가능합니다)

```
int N = Integer.parseInt(br.readLine());

PriorityQueue<Integer> pq = new PriorityQueue<>();
for(int i=0; i<N; i++) pq.add(Integer.parseInt(br.readLine()));

int sum=0;
for(int i=1; i<N; i++){
    int x = pq.poll() + pq.poll();
    pq.add(x);

    sum += x;
}

System.out.print(sum);
```

● 회의실 배정(2709)

| 문제

컴돌이는 한 개의 회의실을 관리하게 되었다. 이 회의실에는 N 개의 회의들이 예약되어 있는데, 컴돌이가 자세히 살펴본 결과 이 중 서로 시간이 겹쳐있는 것들이 있었다.

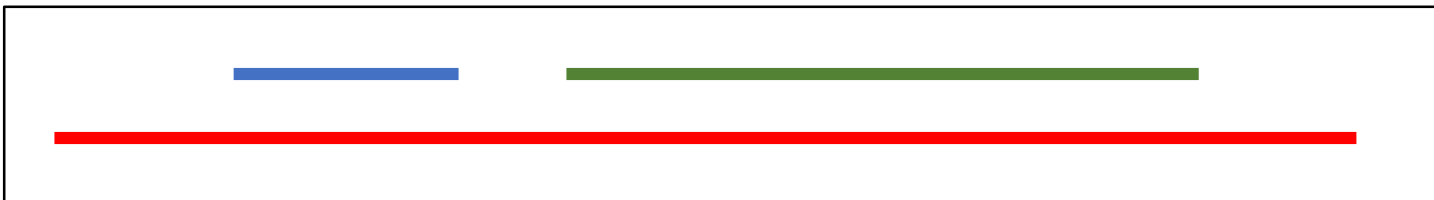
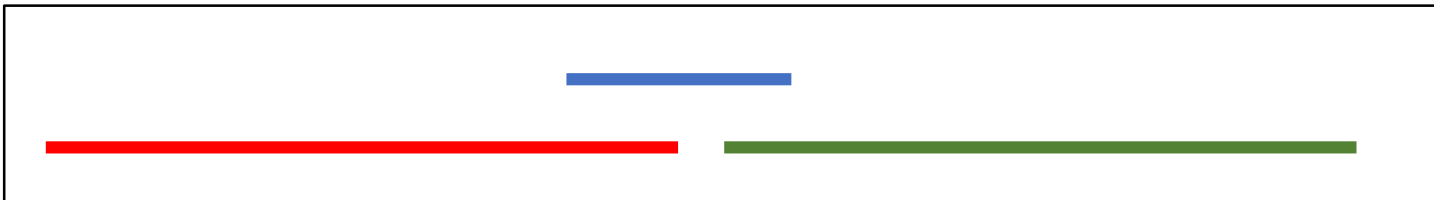
한 회의실에서 두 개 이상의 회의가 열릴 수 없기 때문에 컴돌이는 예약되어있는 몇 개의 회의를 취소하고 회의실 시간표를 짜려고 한다.

회의를 하나 취소할 때마다 입는 타격이 크기 때문에 회의는 최소한으로 취소하려고 할 때, 컴돌이가 열 수 있는 회의의 최대 개수를 구하는 프로그램을 작성하여라.

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.

문제 해설

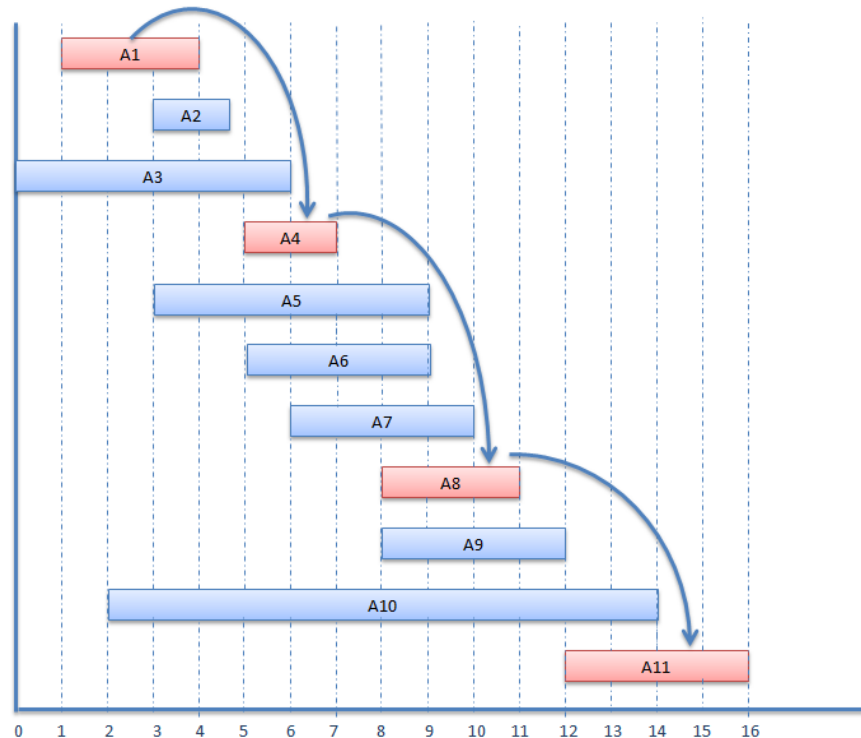
- ▶ 최선의 선택은?
- ▶ 가장 회의시간이 짧은 회의부터 선택
- ▶ 가장 빨리 시작하는 회의부터 선택
- ▶ 가장 빨리 끝나는 회의부터 선택



▶ 가장 빨리 끝나는 회의부터 선택

다음으로 진행되는 회의는 이 회의가 종료된 이후부터 진행 가능

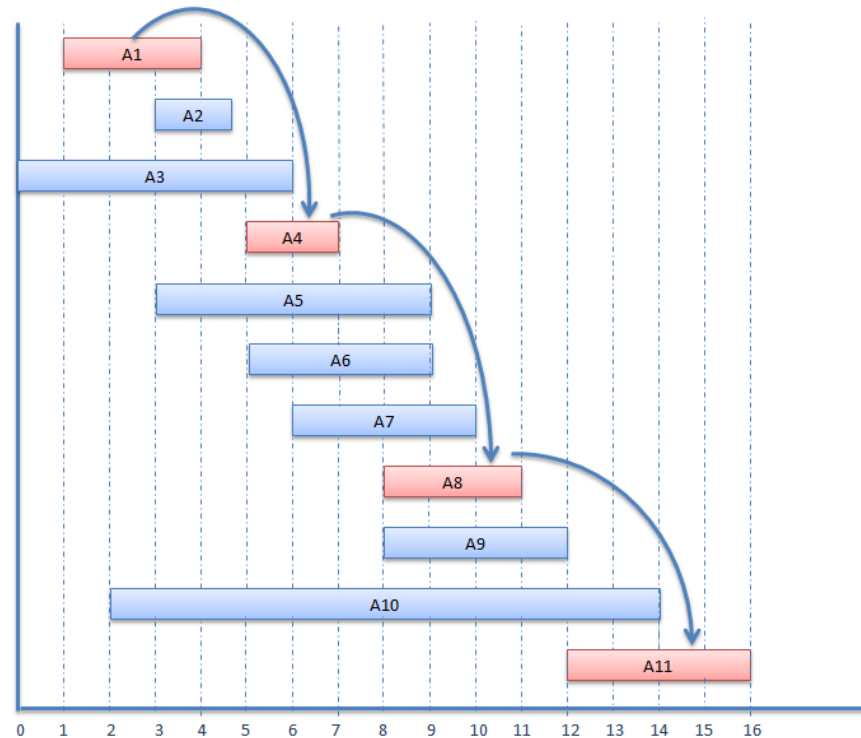
따라서, 가장 먼저 종료되는 회의를 선택하는 것이 이후 선택에 있어 비교우위를 가질 수 있다.



▶ 회의들을 종료시각 순으로 정렬

진행할 수 있는 회의들 중 가장 종료시각이 빠른 회의를 선택

→ 앞에서부터 보면서 가장 먼저 나오는 진행가능한 회의 선택



```
ArrayList<Meeting> meetings = new ArrayList<>();
for(int i=0; i<N; i++){
    st = new StringTokenizer(br.readLine());
    int start = Integer.parseInt(st.nextToken());
    int end = Integer.parseInt(st.nextToken());

    meetings.add(new Meeting(start, end));
}

Collections.sort(meetings);

int end=0, cnt=0;
for(int i=0; i<N; i++){
    if(meetings.get(i).getStart() >= end){
        cnt++;
        end = meetings.get(i).getEnd();
    }
}
```

```
class Meeting implements Comparable<Meeting>{
    private int start, end;

    public Meeting(int start, int end){
        this.start = start;
        this.end = end;
    }

    public int getStart(){ return this.start; }
    public int getEnd(){ return this.end; }

    @Override
    public int compareTo(Meeting x){
        return getEnd() - x.getEnd();
    }
}
```