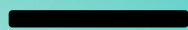


다양한 문제해결 기법



ALOHA

week1

T 김재형

2020.4.6



Introduction

001 개요

010 알고리즘 소개

011 문제풀이

- **Brute force**
- **Greedy algorithm**
- **Divide and conquer**

1. Brute Force Algorithm



- Brute : 짐승,야만적인
- Force : 힘

브루트 포스 알고리즘은 말 그대로 Brute + Force 를 말합니다.

완전탐색 알고리즘. 즉 가능한 모든 경우의 수를 모두 탐색하면서 요구조건에 충족되는 결과만을 가져오는 알고리즘으로 생각하시면 됩니다.

이 알고리즘의 강력한 점은 **예외없이 100%확률로 정답만을 출력**합니다.

하지만 반대로 그만큼 시간이 오래 걸리고 많은 자원을 필요로 한다고 생각하시면 됩니다! π

Brute Force Algorithm 예시

알고리즘 반 여러분들이라면 이미 다 알 수 도 있는 시시콜콜한 브루트 포스에 대해서는 넘어가도록 하겠습니다.!

예를 들어,

▶ 1부터 n 까지 **선형으로 탐색**해서 원하는 값 x 를 찾는가...

▶ A,B의 최대공약수를 구하고 싶을 때는 1부터 $\max(A,B)$ 까지 돌면서 **두 수 동시에 나누어 떨어지는 마지막 수**를 찾거나...

▶ A,B 의 최소공배수를 구하고 싶을 때는 $A*B$ 부터 1까지 쭉 내려가면서 **가장 처음에 A,B로 나누어 떨어지는 수**를 찾는 등

이 이외에도 **BFS,DFS** 모두 **Brute Force** 라고 할 수 있고
이번 수업에서는 백트래킹을 제외한 브루트 포스에 대해서 다루도록 하겠습니다!

Brute Force Algorithm 문제풀이

백준 2309
일곱 난쟁이

백준 2309

◆Key idea

9명중에 7명을 선택하는 것보다 (7중 for문)
9명중에 2명을 선택하는 것이 (2중 for문)
더 나은 선택이 아닐까요?

브루트 포스중 더 나은 브루트 포스를 선택해 봅시다!

- 아홉 난쟁이 중 7명의 키를 더해 100 이되어야 한다.
- 전체 키 합 - 100 = x 라 가정
- 아홉 명 중 두 명을 뺀고 **두명의 키의 합이 x** 가 될 시 **나머지 7명 출력**하자!

시간복잡도는 $9C7 == 9C2$ (C -> combination)

백준 2309 코드

```
8 int main() {
9     for (int i = 0; i < 9; i++) {
10         cin >> arr[i];
11         x += arr[i];
12     }
13     // ↑이때까지 구한 x는 9명 전체의 키의 합을 말함
14     x -= 100;
15
16     for (int i = 0; i < 9; i++) {
17         for (int j = 0; j < i; j++) {
18             //9명의 난쟁이중 2명을 선택하는 과정
19             if (arr[i] + arr[j] == x) {
20
21                 arr[i] = 0, arr[j] = 0;
22                 sort(arr, arr + 9);
23                 //0을 제외한 값들 출력
24                 for (int k = 2; k < 9; k++) {
25                     cout << arr[k] << "Wn";
26                 }
27                 return 0; //메인함수 종료
28             }
29         }
30     }
31 }
32
33 }
```

Brute Force Algorithm 문제풀이

백준 18111
마인크래프트

백준 18111

- 땅 고르기로 인해서 선택할 수 있는 땅의 높이는 0층부터 256층이다!
- $N, M \leq 500$ 이고 모든 땅의 높이에 대해서 시간을 계산해도
시간복잡도는 $500 * 500 * 256$ 으로 1억보다 작아서 1초 이내로 마무리 할 수 있다!

0 이때 중요한 것은 $x[0, 256]$ 라는 층으로 만들기 위해서 **작업의 순서는 중요하지 않다**는 것이다!

만일 해당 작업을 끝내고 인벤토리 **값이 양수일시** x층으로 땅 고르기는 **가능**하지만
만일 인벤토리 **값이 음수일시**에는 블록이 부족하므로 X층 땅 고르기는 **불가능**하다는 것이다!

- 작업도중 인벤토리 값이 음수가 되도 마지막에 결국 인벤토리 값이 양수가 되기만 하면 x 층으로 땅고르기 가능!
(블록 뺐다 집어넣는 순서만 잘 조정하면 되기때문에!)

백준 18111 주요 코드

```
16 int ans_time = 987654321, ans_layer = -1;
17 //정답시간은 가장 크게, 정답높이는 가장작게 초기화
18 //가능한 모든 층에대해서 0층부터 256 층까지 모두확인
19 for (int now_layer = 256; now_layer >=0 ; now_layer--) {
20
21     int now_time = 0;
22     int now_inventory = B;
23
24     //각각의 해당 a층을 만드는데 걸리는 시간구하기!
25     for (int i = 1; i <= N; i++) {
26         for (int j = 1; j <= M; j++) {
27             if (arr[i][j] > now_layer) {
28                 //만일 현재확인하는 칸이 목표층보다 높을시
29                 now_inventory += (arr[i][j] - now_layer);
30                 now_time += (arr[i][j] - now_layer) * 2;
31             }
32             else if (arr[i][j] < now_layer) {
33                 //만일 현재확인하는칸이 목표층보다 낮을시
34                 now_inventory -= (now_layer - arr[i][j]);
35                 now_time += (now_layer - arr[i][j]);
36             }
37         }
38     }
39     //작업이 끝나고 만일 인벤토리값이 음수면 해당층으로 통일하는것은 불가능하다!
40     //(블럭부족)
41     if (now_inventory < 0) continue;
42     if (now_time < ans_time) ans_time = now_time, ans_layer = now_layer;
43 }
44 cout << ans_time << " " << ans_layer << "\n";
```

2. Greedy algorithm



그리디 알고리즘은
탐욕 알고리즘 혹은 욕심쟁이 알고리즘으로도 불리는 데요.
(요즘 SW 테스트, 경시대회에서 핫 함)

미래를 생각하지 않고 각 단계에서 가장 최선의 선택을 하는 기법
입니다. 이렇게 각 단계에서 최선을 한 것이 전체적으로 최선이길
바라는 알고리즘이라고 생각하시면 됩니다!

저희가 다음에 배울
Prim algorithm 과 다익스트라 알고리즘 역시
Greedy의 일종이라고 보시면 됩니다!

Greedy Algorithm (활동 선택문제)

“그게 돼?”

■ 2-2 알고리즘 수업시간에 배울 거라 미리 알아 두면 좋아요 ☺

백준 19331
회의실 배정

➔ 요번에 구글 코드잼 온라인 예선에서도 너무나 비슷한 문제가 하나가 나왔다는...

Greedy algorithm (활동 선택문제)

♣ 문제 이해하기

우리에게 주어진 건 단 하나의 회의실이다. 시작시간과 종료시간을 가진 여러 회의를 입력하고 회의가 서로 겹치지 않게 최대한 많은 수의 회의를 진행할 수 있는 방법을 찾아야 한다.

문제에 주어진 조건

1. 회의가 한번 시작하면 중단되지 않는다.
2. 한 회의가 끝나는 것과 동시에 다음회의가 시작될 수 있다.
3. 회의의 시작시간과 종료시간이 같을 수가 있다!

★ 이를 만족하는 최대 회의 진행 가능 수를 출력해주면 된다!

Greedy algorithm (활동 선택문제)

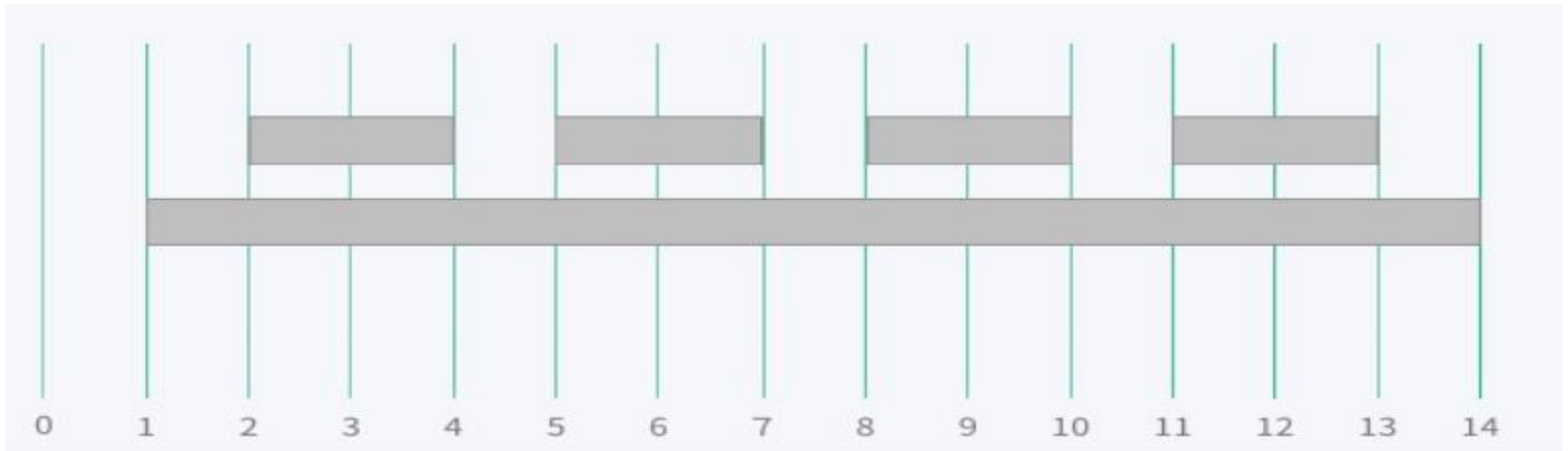
현재 문제 예제에 나온 입력 결과에 따라 회의 진행을 막대 그림으로 나타내면 다음과 같다.



Greedy algorithm (활동 선택문제)

- 회의를 일찍 시작하는 순으로 접근하면?

가장 처음 생각나는 방법이다. 일찍 시작하는 순으로 정렬하면 구할 수 있지 않을까?
하지만 아래 그림과 같은 반례가 존재한다.

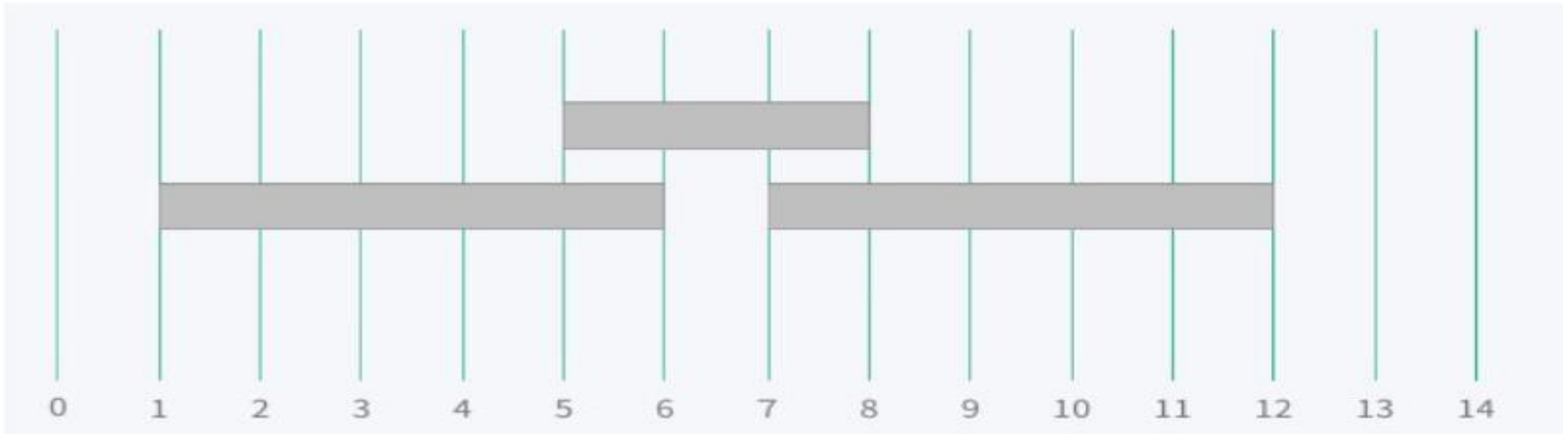


일찍 시작했지만, 종료시간이 늦어서 중간에 빨리 끝나는 회의가 있으면 최대 회의 수를 구할 수 없다.

Greedy algorithm (활동 선택문제)

● 그렇다면 회의가 짧은 순으로 구하면?!

위에서 존재했던 반례를 이용해 짧은 순으로 정렬하면 가능할까 싶지만, 또 반례가 존재한다.

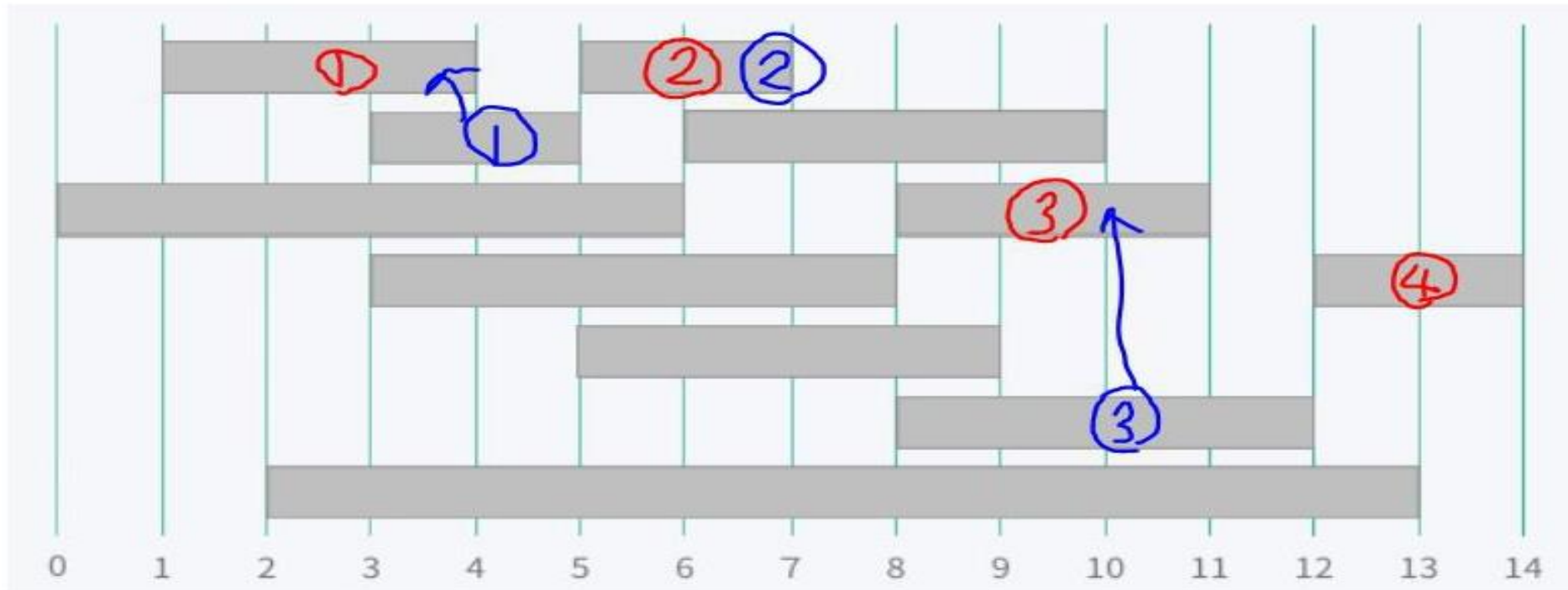


위 그림과 같이 존재한다면, 아무리 짧은 회의 더라도 최대 회의 수를 구할 수 없다.

Greedy algorithm (활동 선택문제)

● 일찍 끝나는 회의를 기준으로 잡으면?!

회의가 일찍 끝나는 기준을 잡으면, 회의를 선택할 수 있는 범위가 넓어진다.
일찍 끝나면 더 회의를 많이 진행할 수 있기 때문이다.



Greedy algorithm (활동 선택문제)

♣ 여기서 몇 가지 의문을 가져봅시다.

1. 과연 **예외**는 없을까?
2. 그냥 단순히 내가 감으로 푸는게 아닐까?
3. 진짜 빨리 끝나는 순으로 정렬했을 때 그것이 전체적으로 최선의 선택이라고 단정짓기에는 좀 이른 감이 있지 않을까?

-

“그게 돼?”



Greedy algorithm (정당성의 증명)

1. 탐욕적 선택 속성

동적 계획법처럼 답의 모든 부분을 고려하지 않고 탐욕적으로만 선택하더라도 최적해를 구할 수 있는 속성

◆ 가장 종료 시간이 빠른 회의 S_{\min} 을 포함하는 최적해가 반드시 존재한다.

증명)

S 의 최적해 중 S_{\min} 을 포함하지 않는 답이 있다면
이 답의 목록에서 첫번째로 개최되는 회의를 지운다.
그리고 S_{\min} 을 대신 추가해서 새로운 목록을 만든다.
 S_{\min} 은 S 에서 가장 일찍 끝나는 회의이기 때문에
지워진 회의는 S_{\min} 보다 일찍 끝날 수 없다.

두번째 회의와 S_{\min} 이 겹치는 일이 없으며,
새로 만든 목록도 최적해 중 하나가 된다.

Greedy algorithm (정당성의 증명)

2. 최적 부분 구조

항상 **최적의 선택만을 내려서 전체 문제의 최적해**를 얻을 수 있음을 보여야한다!
즉, **부분문제의 최적해에서 전체 문제의 최적해**를 만들 수 있음을 보여야한다!

모든 회의를 **종료시간의 오름차순으로 정렬**해 두고
정렬 된 배열의 첫 번째 회의는 무조건 선택해도 된다!! (아까 전에 증명했어용 😊)
그 후 정렬 된 배열을 순회하면서 **첫 번째 회의와 겹치지 않는 회의**를 찾습니다.

회의들은 오름차순으로 정렬 되어있기에
겹치지 않는 회의를 찾자마자 **나머지를 보지 않고 선택해도 된다.**

백준 1931 구조체 설명

```
7  //회의라는 구조체를 미리 만들어서 시작시간 끝나는시간을 정의하고 생성자
8  //역시 미리 만들어둡니다.
9  struct conference {
10     int start;
11     int finish;
12     conference(int a, int b) : start(a), finish(b) {}
13 };
14 // 정렬순서는 끝나는시간이 빠른순으로 갖다놓고 같은 시간에 끝날시 시작시간이 빠른것을 앞에다 둡니다.
15 bool operator<(conference a, conference b) {
16     if (a.finish == b.finish) {
17         return a.start < b.start;
18     }
19     return a.finish < b.finish;
20 }
```

백준 1931 메인 함수 설명

```
21 int main() {
22     int N; cin >> N;
23     vector<conference> vec;
24     for (int i = 0; i < N; i++) {
25         int start_time, end_time;
26         cin >> start_time >> end_time;
27         vec.push_back(conference(start_time, end_time));
28     }
29     sort(vec.begin(), vec.end());
30
31     int answer = 1; //회의의 최대개수(0번 추가한 상태)
32     int before_conference = 0; //직전에 하였던 회의의번호
33     //vec[0]은 무조건 정답에 해당하므로 1번부터 검사합니다.
34     for (int i = 1; i < N; i++) {
35         //만일 직전회의랑 현재보고있는 회의의 시작시간이 겹치지 않는다면
36         if (vec[i].start >= vec[before_conference].finish) {
37             before_conference = i;
38             answer++;
39         }
40     }
41     cout << answer << '\n';
42 }
43
```

Greedy Algorithm

백준 1026
보물

Greedy algorithm

문제 풀다 보면..?

어??? 이거 왠지 하나는 오름차순 하나는 내림차순 하면 될 거 같지 않아?

라는 느낌 적인 느낌이 들고 그것을 코드로 돌려보면 정답이 나와요 ㅎㅎㅎ

근데...

Greedy algorithm (백준 1026)

- ▶ 과연 하나는 오름차순, 하나는 내림차순으로 정렬했다고 해서 각각의 값들은 곱한 것의 합이 최소라고 할 수 있을까?
- ▶ 과연 반례가 진짜 하나도 없는 걸까???

“그게 돼?”

Greedy algorithm (백준 1026)

▶ 증명해 봅시다

만일 $A_1 < A_2$, $B_1 < B_2$ 라고 가정할 시 $A_1 * B_1 + A_2 * B_2 > A_1 * B_2 + A_2 * B_1$ 이 됩니다.

이를 증명하기 위해서는 각 변의 식을 요리조리 넘기고 정리하면

$$A_1(B_1 - B_2) + A_2(B_2 - B_1) > 0 \text{ 이 되고}$$

위 식 다시 정리 시

$$A_1(B_1 - B_2) - A_2(B_1 - B_2) > 0 \rightarrow (A_1 - A_2)(B_1 - B_2) > 0 \text{ 이 되므로 증명됩니다!}$$

...명제1

만일 A배열, B배열을 오름차순으로 정리 후

$$S = A[1] * B[N] + A[2] * B[N-1] + A[3] * B[N-2] \dots \text{이라 가정 시}$$

어느 한부분이라도 A와 B의 연결을 임의로 바꿔버리면

위의 명제1과 같이 $A_1 * B_1 + A_2 * B_2 > A_1 * B_2 + A_2 * B_1$ 과 같은 부분이 나오기 때문에

최소라는 조건에 벗어나게 된다!

백준 1926
코드

```
6      using namespace std;
7
8      int N, answer;
9      int arr1[55], arr2[55];
10
11      int main() {
12          cin >> N;
13          for (int i = 0; i < N; i++) cin >> arr1[i];
14          for (int i = 0; i < N; i++) cin >> arr2[i];
15
16          sort(arr1, arr1 + N);           //정방향 정렬
17          sort(arr2, arr2 + N, greater<int>()); //역방향 정렬
18
19          for (int i = 0; i < N; i++) {
20              answer += arr1[i] * arr2[i];
21          }
22          cout << answer << '\n';
23      }
```

3. Divide and conquer

■ 분할정복

정의 : 분할정복 알고리즘은 문제를 나눌 수 없을 때까지 나누어서 각각을 풀면서 다시 합병하여 문제를 푸는 알고리즘이다.

알고리즘 설계 요령

- (1) Divide : 문제가 분할이 가능한 경우, 2개 이상의 문제로 나눈다.
- (2) Conquer : 나누어진 문제가 여전히 분할이 가능하면, 또 다시 Divide 를 수행한다.
분할이 안될 시 그 문제를 푼다.
- (3) Combine : Conquer한 문제들을 통합하여 문제의 답을 얻는다.

3. Divide and conquer → Part_1

■ 병합 정렬 (Merge sort) (2-2 알고리즘 시간에 배우게 돼요 😊)

◦ 시간복잡도 $O(n \log n)$, 공간복잡도는 $O(n)$ 이다.

◦ 알고리즘

1. 정렬할 데이터 집합의 크기가 0또는 1이면 이미 정렬된 것으로 보고, 그렇지 않으면 데이터의 **집합을 반으로** 나눈다.
2. 원래 같은 집에서 나뉘어져 나온 데이터 **집합 둘을 병합하여 하나의 데이터 집합으로** 만든다.
3. 데이터 집합이 다시 하나가 될 때까지 1,2를 반복한다.

3. Divide and conquer → 분할정복 응용

■ 병합 정렬 (Merge sort)

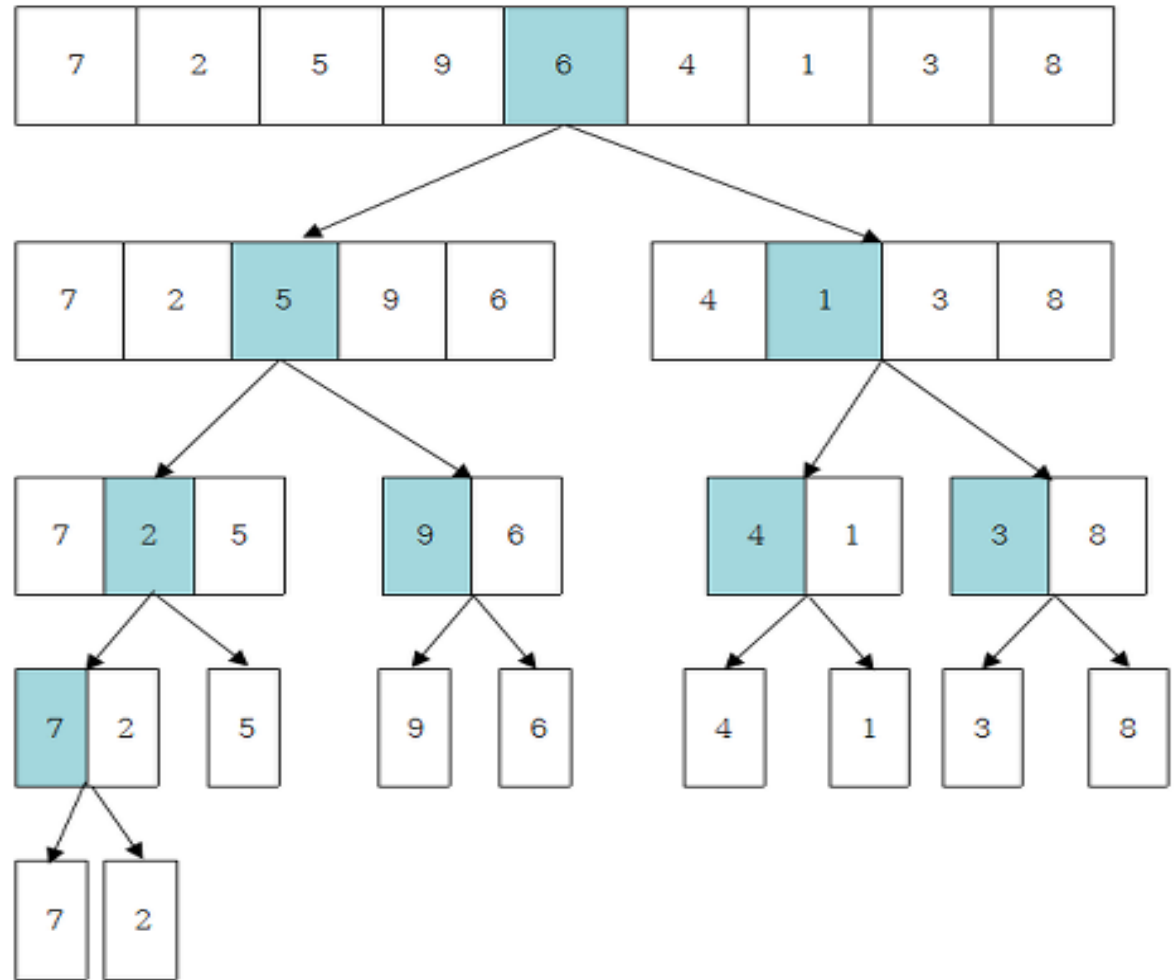
7	2	5	9	6	4	1	3	8
---	---	---	---	---	---	---	---	---

다음과 같은 배열을
공간복잡도 $O(N)$
시간복잡도 $O(N\log N)$
분할정복을 이용해서 **오름차순으로 정렬**해봅시다.

3. Divide and conquer → 분할정복 응용

1. 정렬할 데이터 집합의 크기가 0또는 1이면 이미 정렬된 것으로 보고, 그렇지 않으면 데이터의 집합을 반으로 나눈다.

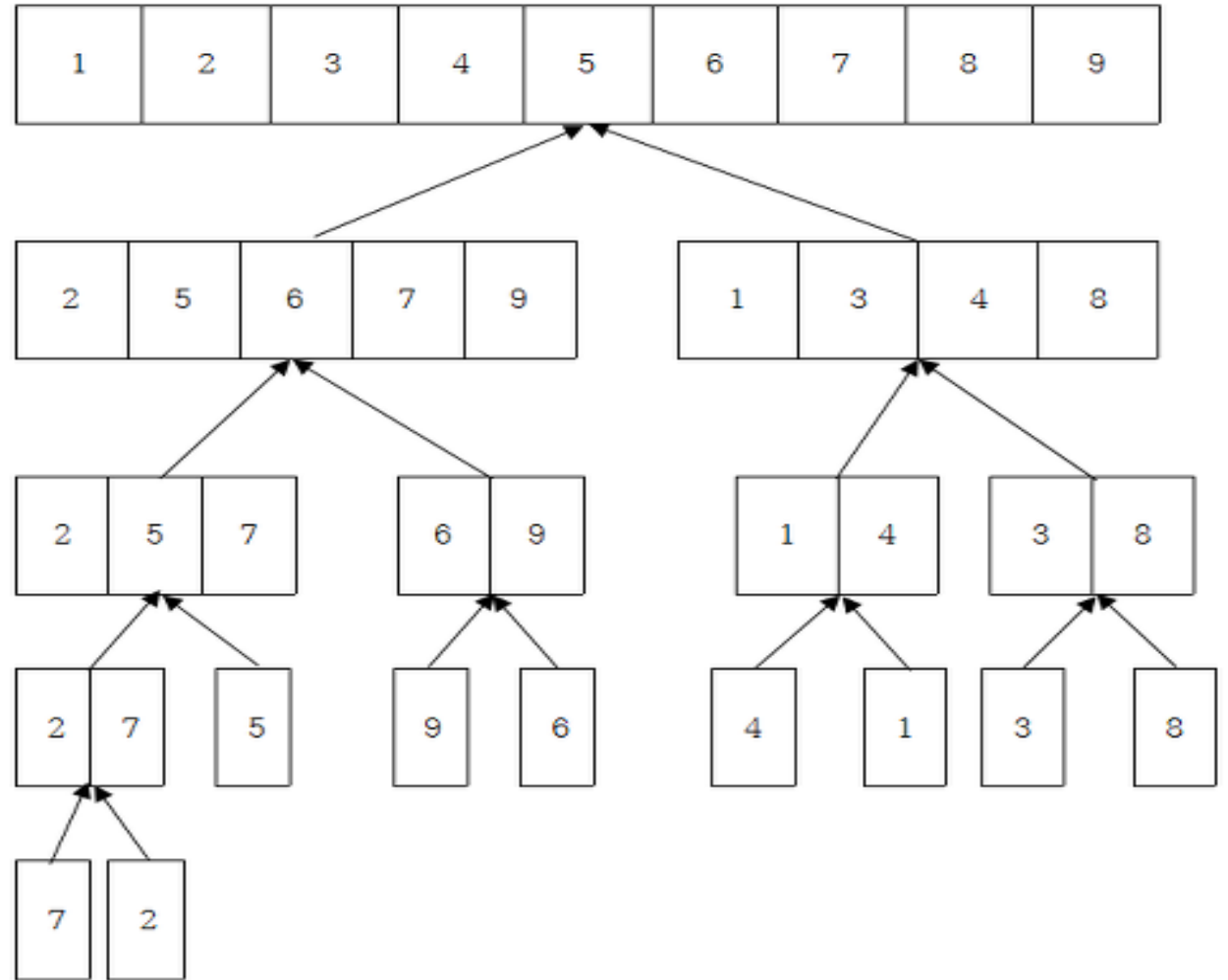
▶ 이때
 $\text{mid} = (\text{start} + \text{end}) / 2$ 를 기준으로
[start, mid] 와 [mid+1, end] 를 기준으로
나누었으며
크기가 1보다 클 때에는 divide 한다!



3. Divide and conquer → 분할정복 응용

2. 원래 같은 집에서 나뉘어져 나온 데이터 집합 둘을 병합하여 하나의 데이터 집합으로 만든다.

▶ 이때 병합이 완성된 후의 집합은 오름차순이 되어있어야 한다!
How???? (다음page에 나옵니당)



3. Divide and conquer → 분할정복 응용

■ 두 데이터 집합을 정렬하면서 합치는 방법

1. 두 데이터 집합의 크기의 합만큼의 크기를 가지는 빈 데이터 집합을 만든다.
2. 두 데이터 집합의 첫 번째 요소들을 비교하여 작은 요소를 빈 데이터 집합에 추가한다.
이 때 새 데이터 집합에 추가한 요소는 원래 데이터 집합에서 삭제한다.
3. 원래 두 데이터 집합의 요소가 모두 삭제 될 때까지 2를 반복한다. -

A

2	5	6	7	9
---	---	---	---	---

B

1	3	4	8
---	---	---	---

C

--	--	--	--	--	--	--	--	--

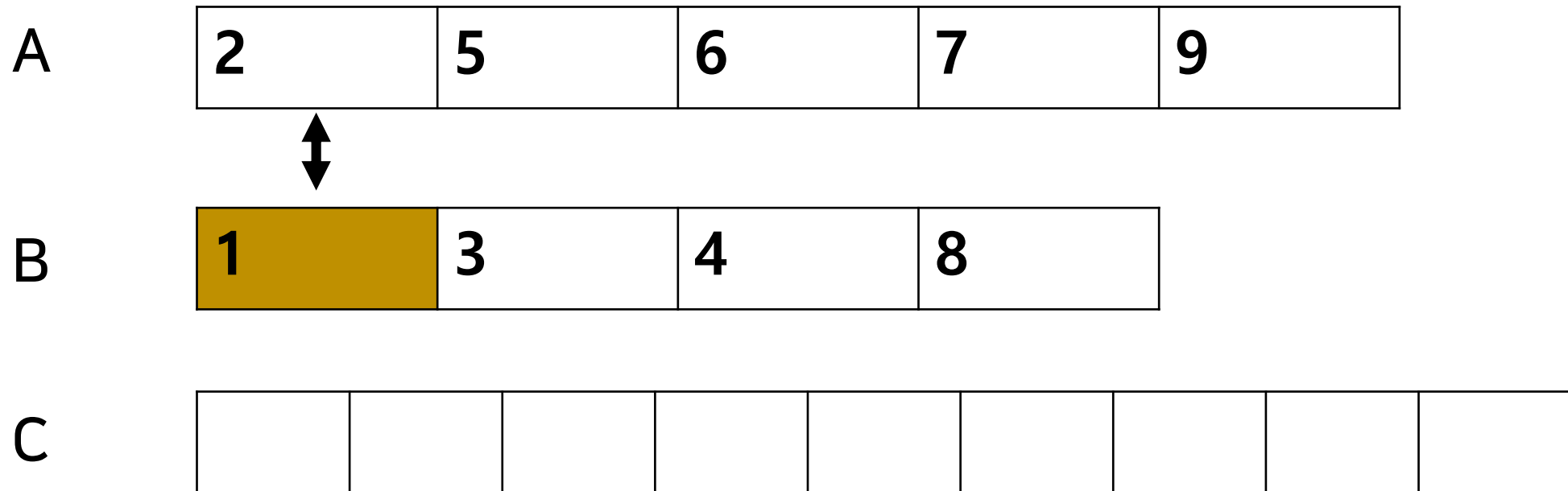
3. Divide and conquer → 분할정복 응용

2. 두 데이터 집합의 첫 번째 요소들을 비교하여 작은 요소를 빈 데이터 집합에 추가한다.
이 때 새 데이터 집합에 추가한 요소는 원래 데이터 집합에서 삭제한다.

두데이터 집합의 첫번째 요소를 비교한다.

A의 첫번째 요소는2, B의 첫번째 요소는 1이므로 B의 것이 더 크다.

C에 1을 추가하고 B에 1을 삭제한다.



3. Divide and conquer → 분할정복 응용

3. 원래 두 데이터 집합의 요소가 모두 삭제 될 때까지 1, 2를 반복한다.

■ A의 2와 B의 3을 비교한다.
2가 작으니 C를 2에 추가하고 A에 2를 삭제한다!

A	2	5	6	7	9
---	---	---	---	---	---



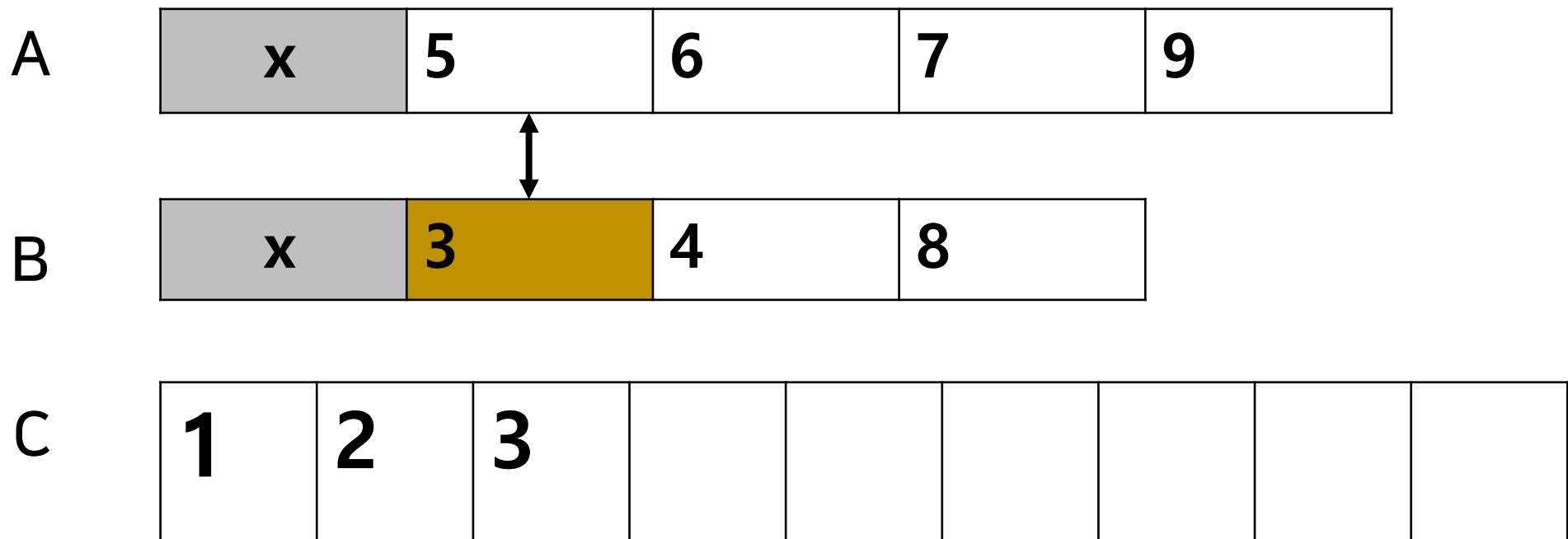
B	x	3	4	8
---	---	---	---	---

C	1	2						
---	---	---	--	--	--	--	--	--

3. Divide and conquer → 분할정복 응용

3. 원래 두 데이터 집합의 요소가 모두 삭제 될 때까지 1, 2를 반복한다.

■ A의 5와 B의 3을 비교한다. 3이 작으니 C에 3을 추가하고 B에서 3을 삭제한다.



3. Divide and conquer → 분할정복 응용

3. 원래 두 데이터 집합의 요소가 모두 삭제 될 때까지 1, 2를 반복한다.

■ 이렇게 데이터 A와 B의 요소들을 비교해서 C에 넣고 A와 B의 각 요소들을 삭제해 나가보면 A에 9하나만 남게 되고 B에는 비교할 요소가 남아있지 않으므로 그냥 9를 C에 추가하고 A에서 9를 삭제한다.

이로써 A와 B의 요소들이 모두 삭제 되었고 C는 정렬된 데이터 집합이 되었다.

A	X	X	X	X	9
---	---	---	---	---	---

B	X	X	X	X
---	---	---	---	---

C	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

3. Divide and conquer → Merge Sort Code

■ 1. 입력부터 받고 초기조건

```
8      int n, number, sorted[500013]; // ← 정렬된 결과값들 sorted 에 저장;
9      void merge(int start, int mid, int end);
10     void merge_sort(int start, int end);
11
12     vector<int> vec; //vector 로 입력받을거임!
13     int main() {
14         cin >> n; //n개의 수 입력받고
15         for (int i = 0; i < n; i++) {
16             cin >> number; vec.push_back(number); //임의의 정렬되지 않은 수 입력
17         }
18         merge_sort(0, n - 1); //merge sort로 한번 정렬해보자!
19     }
20
```

3. Divide and conquer → Merge Sort Code

■알고리즘 설계 요령

(1) Divide : 문제가 분할이 가능한 경우, 2개 이상의 문제로 나눈다.

(2) Conquer : 나누어진 문제가 여전히 분할이 가능하면, 또 다시 Divide 를 수행한다.
분할이 안될시 그 문제를 푼다.

(3) Combine : Conquer 한 문제들을 통합하여 문제의 답을 얻는다.

```
50 void merge_sort(int start, int end) {  
51     if (start < end) {  
52         //merge_sort 할려는 배열의 크기가 2개이상일시  
53         int mid = (start + end) / 2; //집합을 두개로 나누어서  
54         merge_sort(start, mid); // 앞에꺼 또 나누어주고  
55         merge_sort(mid + 1, end); // 뒤에꺼 또 나누어줍시다  
56         merge(start, mid, end); //나누어진 두개의 집합을 병합하자  
57     }  
58 }
```

3. Divide and conquer → Merge Sort Code

■ 분할되었고 정렬된 두개의 배열을
합쳐서 하나의 정렬된 배열로 만드는 merge 함수의 코드를 봅시다☺

Merge 함수 보기전에!!! Merge 함수의 목적과 기능

● 사전지식

1. 두 개의 배열은 정렬된 상태이거나 크기가 1이다.
2. 우리의 목표는 두개의 배열을 합쳐서 하나의 정렬된 배열로 만들 것이다
3. 첫 번째 배열은 `vector_index : start~mid` 까지
두 번째 배열은 `vector_index : mid+1~end` 까지이다!
4. 두 배열의 크기는 모두 1이상이다. (merge_sort의 if문참조)

Merge 함수

Code 1

```
21 void merge(int start, int mid, int end) {
22     int i = start;      //첫번째 배열 시작 index
23     int j = mid + 1;    //두번째 배열 시작 index
24     int k = start;      //결과값을 sorted 배열에 저장시 사용할 index -> k
25
26     while (i <= mid && j <= end) {
27         // 첫번째 배열 두번째배열 모두 아직 i,j가 끝까지 확인안했을때?
28         //아마 i,j의 index를 각각 모두 비교해서 작은수를 집어넣어야겠죠?
29     }
30     if (i > mid) {      //첫번째 배열 index를 모두 돌았을때
31         for (int t = j; t <= end; t++) {
32             sorted[k] = vec[t];      //두번째 배열값만 주구장창 집어넣시다.
33             k++;
34         }
35     }
36     else {              //두번째 배열 index를 모두 다 돌았을때
37         for (int t = i; t <= mid; t++) {
38             sorted[k] = vec[t];      //첫번째 배열값만 주구장창 집어넣시다.
39             k++;
40         }
41     }
42     //sorted 에는 정렬된 vector 값들이 들어가게됩니다!
43     for (int t = start; t <= end; t++) {
44         vec[t] = sorted[t];
45     } //마무리는 vector 값역시 sorted의 값을 대입해서 다음 conquer 때 사용해야죠!
46 }
47
```

While 함수 내부

```
26 while (i <= mid && j <= end) {  
27     if (vec[i] <= vec[j]) { //첫번째 배열값이 두번째 배열보다작을경우  
28         sorted[k] = vec[i];  
29         i++; //index 증가  
30     }  
31     else { //두번째 배열값이 첫번째 배열보다 작을경우  
32         sorted[k] = vec[j];  
33         j++; //index 증가  
34     }  
35     k++;  
36 }
```


성질 하나만 알고갑시다!

■ 정렬의 안정성이란?

같은 값을 가진 데이터의 순서가 정렬 후에도 바뀌지 않고 그대로 유지 되는 정렬을 안정적인 정렬이라고 함.

Merge sort는 안정적인 정렬이라고 할 수 있겠죠?

Merge_Sort

백준 1517
버블소트

이미 저는 90퍼센트 이상의 정답을 주고 갔기에 ..
더 이상 풀이를 주는 것은 무리네요 ㅠㅠ

풀이 생략! ^____^

은 거짓말이고..
계속 고민하시다가 안되면 설명해 드리겠습니다

솔직히

▶ `long long answer=0 ;`

▶ `answer += (long long)(j-k)`

이 두줄만 집어 넣으면 해답이 나와요

근데 이 두줄에 좀 많은 내용이 함축되어 있으므로 혹시나 하는 분들을 위해서
뒷 페이지에 설명드리겠습니다!

혹시 좀 풀 수 있을 거 같다 하시는 분들은 계속 고민하셔도 됩니다!

분할정복 - Merge_Sort

■ 기초배경 -> bubble_sort , merge_sort 둘 다 안정적이다!

안정적이다 정의 : 같은값을 가진 데이터가 정렬후에도 같은 순서를 유지한다!

즉 이말은 버블소트의 swap 하는 과정에서 내림차순으로 정렬하는 일은 절대일어나지 않는다고 생각해주시면 됩니다😊

1	3	4	8
---	---	---	---

2	5	6	7	9
---	---	---	---	---

버블소트에서 swap 이 일어나는 조건 ▶ 앞에 수가 뒤에 수 보다 클때 ▶ 뒤에 수가 앞에 수보다 작을 때

만약 이 두배열을 일렬로 합친다고 가정 했을 때 2라는 수는 1 뒤로 가서

1 2 3 4 8 5 6 7 9 가 되고 이때 swap 은 3번 일어나게 되며

같은 방식으로 5를 적용할 시

1 2 3 4 5 8 6 7 9 가 되고 이때 swap 은 1번 일어나게 되며

같은 방식으로 6,7,9에 대해서 swap 의 개수만 세어주면 swap 의 총 횟수를 구해줄 수 있습니다!

즉, 두번째 탐색하는 배열 값이 첫번째 탐색하는 배열 값보다 작을 경우에만 swap 횟수를 세어주면

자동으로 오름차순 정렬이 되므로 그 부분에만 swap 횟수를 더해줍니다!

3. Divide and Conquer - Part_2

■ 거듭제곱 (Exponentiation)

N 거듭 제곱은 자신을 N 번 곱해야 하므로 $O(N)$ 의 시간 이 소요된다. 이것을 계산하기 위해 조금 바꾸어 보자

$$C^8 = C * C * C * C * C * C * C * C$$

로 정의 되지만 다음과 같이 표현이 가능하다

$$C^8 = C^4 * C^4 = (C^4)^2 = ((C^2)^2)^2$$

C^8 을 구할 때 C를 8번 곱하지 않고 제곱을 두 번 더 반복하면 결국 세번의 연산만으로 같은 결과를 얻을 수 있다.

- ▶ 지수가 짝수일때는 지수를 반으로 나누어서 곱한다.
- ▶ 지수가 홀수일때는 지수에서 1을 빼고 반으로 나누어서 곱하고 밑은 한번 더 곱하면 된다.

$$C^n = \begin{cases} C^{n/2} C^{n/2} & (n \text{은 짝수}) \\ C^{(n-1)/2} C^{(n-1)/2} C & (n \text{은 홀수}) \end{cases}$$

거듭제곱 코드

```
9 long long A, B;  
10 //A를 B번 곱한값 구하는 코드  
11 long long what(long long b) {  
12     long long tmp; //memorization  
13     if(b>1) tmp = what(b / 2);  
14  
15     if (b == 1) return A; //1일때는 그냥 A값 반환  
16     else if (b % 2 == 0) {  
17         return (tmp*tmp); //짝수일때는 두번 나눈 값반환  
18     }  
19     else if (b % 2 == 1) {  
20         return ((tmp*tmp)*A);  
21         //홀수일때는 2번나눈값 제공하고 홀수이므로 A값 한번 더 곱하자!  
22     }  
23 }  
24  
25
```


분할정복 - 거듭제곱 문제

■ 거듭제곱과 행렬을 자유롭게 다룰 수 있다면 ? 백준 (1629), 백준(2740) 참조

백준 10830
행렬제곱

분할정복 - 거듭제곱 문제

■ 아무래도 모두다 쉽게 풀 수 있을 거라 믿습니다.

근데 행렬 곱을 조금 더 멋지고 간편하게 풀 수 있는 코드가 있어서 이 코드 사용해서 풀면 아마 50줄 이내로 풀 수 있을 듯☺

```
7  typedef vector<vector<int>> matrix;
8  //matrix 라는 데이터를 vector<vector<int>> 로 정의합니다!
9  int N; //matrix 사이즈라고 생각해주시면되요:)
10
11  //창소프시간때 배운 연산자 오버로딩+ struct 반환 함수를 정의해줍시다!
12  matrix operator*(matrix mat1, matrix mat2) {
13      matrix ret(N, vector<int>(N)); //return 할 matrix 정의 (초기값은 자동적으로 모두 0입니다)
14
15      for (int i = 0; i < N; i++) {
16          for (int j = 0; j < N; j++) {
17              for (int k = 0; k < N; k++) {
18                  ret[i][j] = (ret[i][j] + (mat1[i][k] * mat2[k][j]));
19                  //행렬곱은 기본적으로 다 하실줄 안다고 믿겠습니다:)
20              }
21          }
22      }
23      return ret;
24  }
25  }
26
```

분할정복 - 문제 풀기

- 분할정복 같은 분할정복 문제 하나 풀어봅시다😊
- 전형적인 분할정복 문제

백준 2630
색종이만들기

분할정복 - 문제풀기(2630)

■ 다시 본론으로 돌아가서 분할정복을 푸는 순서는

- (1) Divide : 문제가 분할이 가능한 경우, 2개 이상의 문제로 나눈다.
- (2) Conquer : 나누어진 문제가 여전히 분할이 가능하면, 또 다시 Divide 를 수행한다.
분할이 안 될 시 그 문제를 푼다.
- (3) Combine : Conquer 한 문제들을 통합하여 문제의 답을 얻는다.

1	2
3	4

다음과 같이
네 부분으로 나누고 안 나누어질때까지
나눈 다음에 다시 합치면 끝!

분할정복
문제풀기(2630)
Code

```
7   int white, blue;
8   void divide_and_conquer(int start_row, int start_col, int end_row, int end_col) {
9
10      //하나의 정사각형이 되어 더이상 못자를때!
11      if (start_row == end_row && start_col == end_col) {
12          if (arr[start_row][start_col] == 0) white++;
13          else blue++;
14          return;
15      }
16      //만일 현재 확인하는 영역이 모두 같은색일때
17      bool all_same_color = true;
18      int before_color = arr[start_row][start_col];
19      //하나의 값을 기준으로 정해놓고 나머지값중 하나라도 다를시 false 로 표시
20      for (int i = start_row; i <= end_row; i++) {
21          for (int j = start_col; j <= end_col; j++) {
22              if (arr[i][j] != before_color) all_same_color = false;
23          }
24          if (!all_same_color) break;
25      }
26      if (all_same_color) {
27          if (before_color == 0) white++;
28          else blue++;
29          return;
30      }
31      //현재확인하는영역에 다른색깔의 색이 하나라도 있을시 나누어줍니다.
32      int rowmid = (start_row + end_row) / 2 , colmid = (start_col + end_col) / 2;
33      divide_and_conquer(start_row, start_col, rowmid, colmid);           //1 영역
34      divide_and_conquer(start_row, colmid+1, rowmid, end_col);           //2 영역
35      divide_and_conquer(rowmid+1, start_col, end_row, colmid);           //3 영역
36      divide_and_conquer(rowmid+1, colmid+1, end_row, end_col);           //4 영역
37      return;
```

이후에는 문제풀이만 있습니다 😊

이론은 여기 까지고
뒤에서 부터는
쭈욱 문제풀이 입니당

Brute Force Algorithm 문제풀이

충분한 시간을 가지고
머리를 끄끄 쓰매면서 고민하는게 좋을듯 ^__^
삼성 SW 역량테스트 기출문제입니다

백준 3190
뱀(snake)
(^--^)

◦ 말 그대로 문제에서 시키는 대로만 구현을 하면 되는데
문제에서 시키는게 좀 많아서 힘들었던 문제

◦Key idea 는 뱀의 몸을 배열에다가 저장을 해서 뱀이 움직일 시
머리와 꼬리만 움직이고 나머지 부분은 따로 신경을 안 써줘도 된다는 것을 알면 조
금이나마 쉽게 풀 수 있다.

◦배열에다가 그때당시 뱀의 머리가 보고있던 방향 역시 저장을 해서
꼬리가 이동할 시 조금 더 편리하게 이동할 수 있다!

■ 각 이동할 때마다 뱀의 몸통을 다 출력해 보는 것도 좋은
디버깅 방법이다!

백준 3190

전역변수 및 함수 소개

```
7 //함수
8 void snake_move(); //뱀의 머리가 보고있는 방향으로 한칸이동!
9 □ int change_direction(int now_dir, char change); //뱀의 머리가'C' 'D' 에따라서 방향바뀜
10   [ // ↑ snake_look 의 index 가 return 값이다!
11
12   //자료형
13   □ int snake_body[103][103]; //뱀의 몸전체를 2차원 배열로 표현
14   [ //int 형으로 한이유는 각각의 몸부위에 보는방향 저장
15
16   pair<int, int> snake_look[4] = { { 1,0 }, { 0,1 }, { -1,0 }, { 0,-1 } };
17                                   //   ↓       →       ↑       ← //다음과 같이 보는방향
18
19   //머리와 꼬리 부분 전역변수로 따로저장
20   int snake_head_row = 1, snake_head_col = 1;
21   int snake_tail_row = 1, snake_tail_col = 1;
22
23   bool where_is_apple[103][103]; //사과위치 저장
24   int N, K,L,X;
25   char C;
26   queue<pair<int,char>> q; //input 큐에다가 저장할것임
27   int now_time; //정답으로 출력할값
28
```

백준 3190

Main 함수 흐름 소개

```
30 int main() {
31     //뱀 초기화
32     memset(snake_body, -1, sizeof(snake_body)); snake_body[1][1] = 1;
33     cin >> N >> K;
34     while (K--) {
35         int apple_row, apple_col;
36         cin >> apple_row >> apple_col;
37         where_is_apple[apple_row][apple_col] = true;
38     }
39     cin >> L;
40     while (L--) {
41         cin >> X >> C;
42         q.push({ X,C });
43     }
44     now_time = 0;
45     while (!q.empty()) {
46         pair<int, char> now_order = q.front(); q.pop();
47         int change_time = now_order.first;
48         char direction = now_order.second;
49         //change_time 이라는 시간 전까지 뱀은 머리가 향하는 방향으로 계속 움직여야한다!
50         while (now_time < change_time) {
51             now_time++;
52             snake_move();
53         }
54         //change_time 시간이 끝나면 오른쪽 혹은 왼쪽으로 뱀의 머리가 방향을 90도 튼다.
55         int snake_head_look = snake_body[snake_head_row][snake_head_col];
56         snake_body[snake_head_row][snake_head_col] = change_direction(snake_head_look, direction);
57     }
58     //만일 명령을 다 수행하였는데도 게임이 안끝났으면 게임이 끝날때까지 움직이자!
59     while (true) {
60         now_time++;
61         snake_move();
62     }
63 }
```

백준 3190
함수 소개 1

```
65 int change_direction(int now_dir, char change) {
66
67     // pair<int, int> snake_look[4] = { { 1,0 }, { 0,1 }, { -1,0 }, { 0,-1 } };
68     //   ↓       →       ↑       ← //다음과 같이 보는방향
69     //전역변수에 다음과 같이 선언하였었습니다!
70
71     if (now_dir == 0) {
72         if (change == 'L') return 1; // ↓ + 'L' = → 반환
73         else if (change == 'D') return 3; // ↓ + 'D' = → 반환
74     }
75     else if (now_dir == 1) {
76         if (change == 'L') return 2; // → + 'L' = ↑ 반환
77         else if (change == 'D') return 0; // → + 'D' = ↓ 반환
78     }
79     else if (now_dir == 2) {
80         if (change == 'L') return 3; // ↑ + 'L' = ← 반환
81         else if (change == 'D') return 1; // ↑ + 'D' = → 반환
82     }
83     else if (now_dir == 3) {
84         if (change == 'L') return 0; // ← + 'L' = ↓ 반환
85         else if (change == 'D') return 2; // ← + 'D' = ↑ 반환
86     }
87 }
88
```

백준 3190

함수 소개 2-1

```
89 void snake_move() {
90     int now_look = snake_body[snake_head_row][snake_head_col];
91
92     //1. 뱀의 몸길이를 늘려서 머리를 다음칸에 위치시키는 과정
93     int next_head_row = snake_head_row + snake_look[now_look].first;
94     int next_head_col = snake_head_col + snake_look[now_look].second;
95
96     //만일 뱀의 머리가 맵밖으로 나가게 된다면 프로그램종료
97     if ( !(next_head_row >= 1 && next_head_row <= N && next_head_col >= 1 && next_head_col <= N) ) {
98         cout << now_time << 'Wn';
99         exit(0);
100     }
101     //만일 뱀의 머리가 자신의 몸과 부딪히게 된다면 프로그램 종료
102     if ( snake_body[next_head_row][next_head_col] != -1) {
103         cout << now_time << 'Wn';
104         exit(0);
105     }
106 }
```

백준 3190 함수 소개 2-2

```
106
107     if (where_is_apple[next_head_row][next_head_col]==true) {
108         //2.사과가 있을시 사과 없애고 꼬리는 움직이지 말자
109         where_is_apple[next_head_row][next_head_col] = false;
110     }
111     else {
112         //3.사과가 없을시 몸길이를 줄여서 꼬리가 위치한 칸을 비워준다.
113         //이때 꼬리는 snake_body 에 저장되어있는 tail 이 보고있는방향으로 이동해준다
114         int tail_look      = snake_body[snake_tail_row][snake_tail_col];
115         int next_tail_row = snake_tail_row + snake_look[tail_look].first;
116         int next_tail_col = snake_tail_col + snake_look[tail_look].second;
117
118         snake_body[snake_tail_row][snake_tail_col] = -1;
119         //뱀 꼬리 다시 세팅 + 원래꼬리 -1로 재배치!
120         snake_tail_row = next_tail_row;
121         snake_tail_col = next_tail_col;
122     }
123
124     //뱀 머리 다시 세팅
125     snake_body[next_head_row][next_head_col] = now_look;
126     snake_head_row = next_head_row;
127     snake_head_col = next_head_col;
128 }
```


백준 1541
잃어버린 괄호

분할정복 - 문제풀기(2630)

■ 처음 마이너스 나온 값을 기준으로

$(a_1 + a_2 + \dots + a_n) \text{ " - " } (b + c + d - e - g + f \dots)$

다음과 같이 첫번째 부분 두번째 부분으로 나눌 수 있다. 이때 n 의 값이 1이상이다 .

결론적으로 말하자면

이 두번째 부분은 우리가 괄호를 적절히 쳤을 때
 $-b-c-d -e-g-f$ 로 만들 수 있습니다!

이후에는 문자열처리와 다름없는 문제가 되므로 주욱 풀면 됩니다!

1. 두번째 괄호 부분에 plus 가 들어가 있으면 $A+B$

이런 형태일 텐데 $-(A+B)$ 이렇게 해주시면 되고

2. 괄호부분에 minus 가 들어가 있으면 $(a_1+a_2..) - (A-B)$

이런 형태일 텐데 이때는 그냥 순차적으로 마이너스 해주시면 됩니다

즉, 두번째 괄호부분부터 나오는 숫자는 모두 빼기 해주면 된다!

Greedy Algorithm

백준 1541
잃어버린 괄호

“그게 돼?”

```
6   string str;
7   int minResult()
8   {
9       int result = 0;
10      string temp = "";
11      bool minus = false;
12      for (int i = 0; i <= str.size(); i++)
13      {
14          //연산자일 경우
15          if (str[i] == '+' || str[i] == '-' || str[i] == 'W0')
16          {
17              if (minus) result -= stoi(temp); //첫번째 마이너스가 나온이후
18              else result += stoi(temp); //첫번째 마이너스가 나오기이전
19
20              temp = ""; //초기화
21
22              if (str[i] == '-') minus = true;
23              //첫번째 마이너스가 나오면 이후에 계속 마이너스 해줘도 된다
24              continue;
25          }
26          //피연산자일 경우
27          temp += str[i]; //temp 에 입력받은 문자열저장(곧숫자가될예정)
28      }
29      return result;
30  }
```


Divide and conquer

백준 6549
히스토그램에서
가장 큰 정사각형

Divide and conquer 백준6549

1. 0~N 까지의 구간에서 최댓값을 구한다고 가정하면
2. start~ mid-1 까지의 최댓값과
3. mid 가 포함되어 있을 때의 최댓값과
4. mid+1 부터 end 까지의 최댓값을 구하고

2,3,4 중에서 최댓값을 구해봅시다!

추가문제

1107 리모컨
14502 연구소
1946 신입사원
2352 반도체설계
1992 퀘드트리