완전탐색과 동적 계획법 알고리즘은 궤를 같이했다. 그도 그럴 것이 완전탐색의 중복 문제를 해결하는 방법이 곧 동적 계획법 알고리즘이었기 때문이다. 핵심 아이디어는 변하지 않았다. 이번에는 문제 자체를 다르게 접근하자.

자, **우리는 수열**arrarr**을 입력받았다. 이제 이 미지의 수열을 하나씩 건너가며 lis를 만들어나간다고 하자. 이때 우리는 이 수열이 정수 수열이라는 것만 알고 그 크기 등은 모르는 상태다.**

어떤 수열의 원소를 4개를 먼저 살피니 [5,6,1,2][5,6,1,2]였다고 하자. 현재까지의 lis는 [5,6][5,6], [1,2][1,2]다. 뒤에 더 많은 숫자가 있지만 일단은 지금의 정보가 최선이다.

다음엔 어떤 수열의 첫 5개의 숫자가 다음과 같다고 하자: [5,6,7,1,2,...][5,6,7,1,2,...]. 현재까지의 lis는 [5,6,7][5,6,7]이다. lis를 위한 우리의 위대한 여정을 계속한다고 할 때 현재까지의 lis 정보가 의미있을까? 그건 상황에 따라 다른데 크게 두 가지 경우로 나눌 수 있다.

* 의미없다. [1,2][1,2]가 최종적인 lis의 시초가 될 수 있다. 알고 보니 원 수열이 [5,6,7,1,2,3,4][5,6,7,1,2,3,4]일 수도 있으니까. 이때는 중간 수열 [5,6,7][5,6,7] 정보가 무의미한 정보였다.
* 의미있다. 원수열이 [5,6,7,1,2][5,6,7,1,2]에서 끝나버린다면 [5,6,7][5,6,7]의 크기 3이 곧 답이 된다.

다시 말하지만 우리는 현재까지의 수 이상의 미지의 수는 아직 모르는 상태다. 그렇기 때문에 중간에 생기는 lis가 무의미할 수도, 의미있을 수도 있는 상황이기에 애매하다. 결국 우리의 인생처럼 무의미와 의미의 경계에서 헤매이게 되는걸까…?

잠깐, 불확실의 살얼음을 걷는 나에게도 아직 확실한 것은 남아 있다. 마찬가지로 **여기까지의 불완전한 수열(**[5,6,7,1,2][5,6,7,1,2]**)까지만 봤을 때도 확실한 것들이 있다.** 길이가 1인 증가 부분수열들의([5],[6],...[2][5],[6],...[2]) 마지막 값 중 최소의 값은 1이고, 길이가 2인 증가 부분수열들의([5,6],[1,2][5,6],[1,2]) 마지막 값 중 최소의 값은 2이다. 길이가 3인 증가 부분수열의([5,6,7][5,6,7]) 마지막 값 중 최소의 값은 3이다.

그리고 **증가 부분수열의 크기가 같다면, 이때 마지막 값의 크기가 작은 것의 정보를 유지하는 것이 유리하다.** 가령 배열의 첫 네 원소가 [5,6,1,2][5,6,1,2]일 때 크기가 2인 증가 부분수열은 [5,6][5,6]과 [1,2][1,2]이 있다. 그 다음 수가 무엇이 될지는 모르겠지만 작은 정보를 유지할 때는 lis를 문제없이 구할 수 있다. 가령 원 수열의 바로 뒤의 수이자 마지막 수가 3일 때나 11111일 때 모두 [1,2][1,2]는 lis를 만들어낼 수 있다.([1,2,3],[1,2,11111][1,2,3],[1,2,11111]) 하지만 [5,6][5,6]은 3일 때 lis를 이어가지 못한다.([5,6,3][5,6,3]) 즉, **같은 크기의 증가수열에서 최소의 마지막 값만 기억하면 문제를 풀어낼 단서를 찾을 수 있다.**

앞선 *lis(i)* 는 *i* 인덱스부터 시작하는 부분배열의 lis의 길이를 구하는 것이었다면 이번에는 CC라는 배열을 관리하는데 C[i]C[i]라는 문제는 다음과 같이 정의된다.

C[i] = 길이가 i인 증가수열들 중에서 최소의 마지막 값

입력이 빈 배열이 아닐 경우 lis의 길이의 최소값은 1이고, 최대값은 배열의 크기 자체다. 그래서 우리는 **|arr| + 1** 크기의 배열을 준비한다. | |은 집합의 크기를 산정하는 수학기호다. 크기를 1 늘리는 이유는 배열의 크기 자체를 인덱싱 가능하게 하기 위해서다. 우리는 원 배열을 순회해 나가면서 C 배열의 각 값을 최소값으로 다듬을 예정이기 때문에 일단 무한으로 초기화해놓자. 그리고 크기가 0인 증가 부분수열은 다루지 않기 때문에 음의 무한으로 초기화해놓는다.

원 배열을 [5,6,7,1,2][5,6,7,1,2]까지 진행했을 때 C 배열은 다음과 같을 것이다:

C = [-inf, 1, 2, 7, Inf, Inf]

이것이 이해되어야 식을 유도할 수 있다. 여기서C[i]여기서C[i]**가 무한이 아닌 i의 최대값이 lis의 길이가 됨을 알 수 있다.** *i* 가 3일 때가 7인데 현재까지의 lis의 크기는 3이 된다. C의 원소들의 정확한 값은 중요하지 않다. 각 크기의 증가 부분수열은 매우 많을 수 있는데 우리는 각 크기의 증가 부분수열의 마지막 값 중 최소값만 계속 다듬어 나간다. **이렇게 다듬어 나가는 과정을 거치다보면 어느 순간 원소의 끝에 도달할 수 있을테고 그때 우리는**C[i]C[i]**가 무한이 아닌 마지막 i를 통해 lis의 길이를 반환할 수 있게 된다.**

자, 이제 다듬는 과정을 조금만 더 살펴보자. C[i]C[i]은 처음에 모두 양의 무한으로 시작할텐데 어떻게 최소값으로 깎아나가는가?(0번째 인덱스는 무시한다.) 앞선 중간 탐색과정의 CC 배열을 더 활용하자. 현재 원 배열을 순회 중에 있고 다음 원소를 살펴본다. 이때는 다음과 같은 경우의 수가 있겠다.

이것이 이해되어야 식을 유도할 수 있다. 여기서C[i]여기서C[i]**가 무한이 아닌 i의 최대값이 lis의 길이가 됨을 알 수 있다.** *i* 가 3일 때가 7인데 현재까지의 lis의 크기는 3이 된다. C의 원소들의 정확한 값은 중요하지 않다. 각 크기의 증가 부분수열은 매우 많을 수 있는데 우리는 각 크기의 증가 부분수열의 마지막 값 중 최소값만 계속 다듬어 나간다. **이렇게 다듬어 나가는 과정을 거치다보면 어느 순간 원소의 끝에 도달할 수 있을테고 그때 우리는**C[i]C[i]**가 무한이 아닌 마지막 i를 통해 lis의 길이를 반환할 수 있게 된다.**

자, 이제 다듬는 과정을 조금만 더 살펴보자. C[i]C[i]은 처음에 모두 양의 무한으로 시작할텐데 어떻게 최소값으로 깎아나가는가?(0번째 인덱스는 무시한다.) 앞선 중간 탐색과정의 CC 배열을 더 활용하자. 현재 원 배열을 순회 중에 있고 다음 원소를 살펴본다. 이때는 다음과 같은 경우의 수가 있겠다.