Editorial for 1st hjroh cup

by hjroh0315

목차

A1. 스네이크 **Easy**	A2. 스네이크 그리기 **Medium**	B1. 1차원 2048 **Medium**	B2. 1차원 2048과 쿼리 **Medium**	C. 과녁 맞추기 **Medium**
D1. 구슬 정렬 **Hard**	D2. 구슬 정렬 (Hard) **Hard**	E. 소수의 합 **Medium**	F. 경찰서 **Challenging**	G. 공약수 (More Huge) **Challenging**

A1.
스네이크

graphs, math, arithmetic

출제자 의도: **Easy**

- 제출 422회, 정답 237명 (정답률 58.294%)
- First Solve: ai4youej, 0분

A1.
스네이크

graphs, math, arithmetic

출제자 의도: **Easy**

- n과 m의 기우성에 따라 정답이 다릅니다.
- n과 m이 모두 홀수인 경우 정답은 $nm - 1$입니다.
- 이외의 경우 정답은 언제나 nm입니다.
- 증명은 **A2. 스네이크 그리기**의 풀이에서 하겠습니다.

A2.
스네이크 그리기

graphs, constructive

출제자 의도: **Medium**

- 제출 224회, 정답 76명 (정답률 34.821%)
- First Solve: kdh9949, 7분

A2.
스네이크 그리기

graphs, constructive

출제자 의도: **Medium**

- 격자를 체스보드 형태로 칠하면, 인접한 두 칸은 색이 서로 다릅니다.
- 또한 “최적의 전략”에 따를 때 뱀의 길이는 홀수일 수 없습니다.
- 뱀의 길이가 홀수인 경우 뱀의 머리와 뱀의 꼬리에 해당하는 칸의 색이 같아야 합니다. 이는 “뱀의 머리와 꼬리가 인접해 있다”라는 조건에 위배됩니다.

A2.
스네이크 그리기

graphs, constructive

출제자 의도: **Medium**

- 따라서 n과 m이 모두 홀수일 때 뱀의 길이는 nm일 수 없습니다.
- 이제 n과 m이 모두 홀수일 때 길이가 $nm-1$인 "최적의 전략"이 존재함을 보이면 증명이 완료됩니다.

A2.
스네이크 그리기

graphs, constructive

출제자 의도: **Medium**

- n과 m이 모두 홀수인 경우를 증명하기 전, 우선 하나 이상이 짝수인 경우를 증명합니다.
- n이 짝수라면, 아래와 같은 구성이 가능합니다.

1	12	11	10
2	5	6	9
3	4	7	8

- n이 짝수가 아니며 m이 짝수인 경우, 대칭성을 통해 증명됩니다.

A2.
스네이크 그리기

graphs, constructive

출제자 의도: **Medium**

- n과 m이 모두 홀수일 때, x축 길이가 n이 아닌 $n-1$인 경우는 이미 증명되었습니다.
- 해당 경우에서, 맨 오른쪽 열에서 수직으로 올라가는 부분을 변형합시다.
- 수직으로 올라가는 것 대신, 위-오른쪽-위-왼쪽으로 이동하는 것을 반복합니다.
- 이 경우 최종적으로 뱀은 오른쪽 아래 모서리를 제외한 모든 칸을 사용하며, 길이는 $nm-1$이 됩니다.
- 이로써 증명을 마칩니다.

B1.
1차원 2048

greedy, priority_queue

출제자 의도: **Medium**

- 제출 391회, 정답 139명 (정답률 37.084%)
- First Solve: ai4youej, 4분

B1.
1차원 2048

greedy, priority_queue

출제자 의도: **Medium**

- 0을 제외한 작은 숫자부터 더해 주면 됩니다.
- 어떠한 2의 거듭제곱 2^k가 수열에 2개 존재하면 이를 2^{k+1}로 합칠 수 있으며, 항상 합치는 것이 이득입니다.
- 단, 이 문제는 우선순위 큐를 통해서 수를 하나씩 합쳐도 시간초과가 발생하지 않도록 설계되었습니다.
- 우선순위 큐를 사용하는 경우 시간 복잡도는 대략 $O(n \log n \log M)$입니다.

B2.
1차원 2048과 쿼리

greedy, math

출제자 의도: **Medium**

- 제출 231회, 정답 73명 (정답률 32.035%)
- First Solve: kclee2172, 6분

B2.
1차원 2048과 쿼리

greedy, math

출제자 의도: **Medium**

- 우선순위 큐를 통해 매 쿼리마다 배열 전체를 다시 처리하면 약 $O(Q^2 \log Q \log M)$의 시간 복잡도가 됩니다. 이는 너무 느립니다.

B2.
1차원 2048과 쿼리

greedy, math

출제자 의도: **Medium**

- 각각의 2의 거듭제곱에 대해 개수를 관리합시다. 다시 말해서, $[1,2,4,2]$는 $1*1$, $2*2$, $4*1$이 됩니다.
- 2의 거듭제곱들의 개수에 대한 배열이 있을 때, 합치는 과정을 $O(\log M)$ 시간 복잡도에 수행할 수 있습니다.
- 이를 사용해 $O(Q \log M)$ 시간복잡도의 풀이를 짤 수 있으며, 이는 문제를 통과합니다.

B2.
1차원 2048과 쿼리

greedy, math

출제자 의도: **Medium**

- 더 빠르게 문제를 해결하는 방법도 존재합니다. 덧, 뺄셈과 비트 연산이 충분히 빠르다고 가정할 때, $O(Q)$에 문제를 해결할 수 있습니다.
- 2의 거듭제곱의 개수에 대한 배열이 있을 때, 숫자들을 합치는 과정을 덧셈의 받아올림에 대응해 생각합시다.
- 이렇게 생각할 때, 합친 후의 결과에 해당하는 배열은 모든 숫자를 더한 결과의 이진법 표기와 동일합니다.
- 따라서, 모든 숫자의 합을 관리하여 문제를 해결할 수 있습니다.
- 각 쿼리의 정답은 모든 숫자의 합의 MSB에 해당합니다. 단, 모든 숫자의 합이 0인 경우 0을 출력해야 함에 유의합니다.

C.
과녁 맞추기

geometry, physics,
hash_map/tree_map

출제자 의도: **Medium**

- 제출 137회, 정답 33명 (정답률 24.818%)
- First Solve: kclee2172, 10분

C.
과녁 맞추기

geometry, physics,
hash_map/tree_map

출제자 의도: **Medium**

- 문제에 주어진 조건 중 필요없는 것이 많습니다.
- 단위에 대한 설명은 가속도에 대한 설명이 잘 정의되도록 추가된 것입니다.
- 중력 가속도의 값은 필요가 없습니다. 해당하는 값이 바뀌어도 문제는 잘 정의되며 답이 바뀌지 않습니다.

C.
과녁 맞추기

geometry, physics,
hash_map/tree_map

출제자 의도: **Medium**

- 모든 포물선은 각각 대응되는 단 하나의 식을 가지며, 각각의 과녁 또한 단 하나의 포물선에 대응됩니다.
- 각각의 과녁에 대응되는 포물선의 계수를 해시맵 또는 트리맵으로 세어주면 됩니다.
- 이 때, 속도가 0인 경우, 속도가 ∞인 경우, 포물선이 아래로 볼록한 경우 등에 대해 처리해야 합니다.

D1.
구슬 정렬

math, sorting

출제자 의도: **Hard**

- 제출 73회, 정답 22명 (정답률 30.137%)
- First Solve: pichulia, 15분

D1.
구슬 정렬

math, sorting

출제자 의도: **Hard**

- 문제를 각 막대에 대해 나누어 생각합시다.
- 각 막대에서 구슬의 배치를 알고 있을 때, 해당 막대에서 구슬이 이동하는 거리는 쉽게 구할 수 있습니다.
- 해당 문제의 풀이 방법은 25379: 피하자에서도 활용된 바 있습니다.
- 그러나 모든 막대에 대해 일일이 답을 구하는 것은 $O(m)$ 시간복잡도로 비효율적입니다.

D1.
구슬 정렬

math, sorting

출제자 의도: **Hard**

- 값의 개수가 n개이므로, 서로 다른 값의 종류도 최대 n가지입니다.
- 따라서, 서로 다른 배치에 대해서만 정답을 구한 뒤 해당 배치가 등장하는 횟수로 곱해서 더하는 것을 생각해 볼 수 있습니다.
- 이러한 풀이에는 정렬이 요구되며, 시간복잡도는 $O(n \log n)$이 됩니다.

D2.
구슬 정렬 (Hard)

segment_tree,
value_compression

출제자 의도: **Hard**

- 제출 24회, 정답 13명 (정답률 50.000%)
- First Solve: kcm1700, 28분

D2.
구슬 정렬 (Hard)

segment_tree,
value_compression

출제자 의도: **Hard**

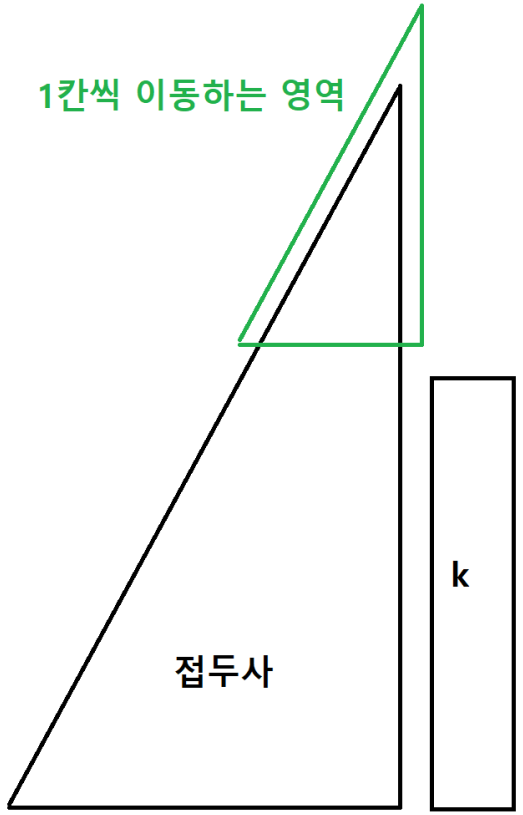
- 어떤 접두사에 해당하는 영역이 정렬되어 있다고 생각하고, 다음에 원소가 추가될 때 구슬이 얼마나 많이 이동하는지 생각합시다.

D2.
구슬 정렬 (Hard)

segment_tree,
value_compression

출제자 의도: **Hard**

- 어떤 접두사가 정렬되어 있고, 그 뒤에 원소 k가 추가된다면, k보다 높은 위치에 있는 구슬들이 모두 정확히 1칸씩 이동합니다.
- 그림으로 나타내면 다음과 같습니다.



D2.
구슬 정렬 (Hard)

segment_tree,
value_compression

출제자 의도: **Hard**

- "k보다 높은 위치에 있는 구슬들"의 개수를 다시 수식으로 표현하면 다음과 같습니다.
- $(k$보다 큰 값의 합$) - ($그 중 이동하지 못하는 개수$)$
- 이는 다시 정리하면 다음과 같습니다.
- $(k$보다 큰 값의 합$) - k(k$보다 큰 값의 개수$)$
- 값들의 합과 값들의 개수를 각각 Segment Tree로 관리하면 각 접두사에 대해 문제를 $O(\log n)$에 해결할 수 있습니다.
- 값의 범위가 크므로, 값 압축을 시행하거나 Dynamic Segment Tree를 사용해야 합니다.
- 값은 각각 매번 하나씩만 업데이트하므로 Lazy Propagation은 필요하지 않습니다.

E.
소수의 합

knapsack, dp,
number_theory

출제자 의도: **Medium**

- 제출 61회, 정답 26명 (정답률 42.623%)
- First Solve: hyjk826, 4분

E.
소수의 합

knapsack, dp,
number_theory

출제자 의도: **Medium**

- 3067: Coins와 전반적으로 동일한 풀이를 사용합니다.
- n의 값이 크므로, $O(n^2)$ 풀이는 통과하기 어렵습니다.
- 그러나, 소수 모두에 대해 Knapsack DP를 수행하면 실제 시간 복잡도는 $O(\frac{n^2}{\log n})$이 됩니다.
- 이는 소수 정리 때문입니다.
- 소수 정리에 따라, n 이하의 소수는 $O(\frac{n}{\log n})$개입니다.
- 따라서, 결과적으로 시간 복잡도는 $O\left(\frac{n}{\log n}n\right) = O(\frac{n^2}{\log n})$입니다.

F.
경찰서

출제자 의도: **Challenging**

- 제출 25회, 정답 4명 (정답률 16.000%)
- First Solve: kclee2172, 69분

F.
경찰서

출제자 의도: **Challenging**

- 2가지 풀이가 존재합니다. 이 슬라이드에서 2가지 풀이 모두를 소개하고자 합니다.

F.
경찰서 (풀이 1)

halfplane_intersection,
binary_search,
parametric_search

출제자 의도: **Challenging**

- 첫 번째 풀이는 반평면 교집합을 사용합니다.
- 어떤 직선에서 거리 r 이하의 영역이 있다고 합시다.
- 이 영역 안에는 반지름이 r이며 이 직선과 만나는 어떤 원의 중심이 존재합니다.
- 이를 모든 직선에 대해 확장하면, 이러한 영역 n개의 교집합 안에는 반지름이 r이며 모든 직선과 만나는 어떤 원의 중심이 존재합니다.

F.
경찰서 (풀이 1)

halfplane_intersection,
binary_search,
parametric_search

출제자 의도: **Challenging**

- 어떤 직선에서 거리 r 이하인 영역은 반평면 2개의 교집합으로 나타낼 수 있습니다.
- 정확히는, 직선에서 각 방향으로 r만큼 이동한 직선 2개로, 반평면 2개가 직선을 포함하게 하면 됩니다.
- 이에 따라 관리해야 하는 반평면의 개수는 $2n = O(n)$ 개가 됩니다.
- $O(n \log n)$ 시간복잡도의 반평면 교집합 구현을 사용하면 $O(n \log n \log X)$ 시간복잡도로 통과합니다.
- 무작위화를 사용하는 $O(n)$ 시간복잡도의 반평면 교집합 구현을 사용하면 더 빠르게 해결 가능합니다.

F.
경찰서 (풀이 2)

min_enclosing_circle, randomization

출제자 의도: **Challenging**

- 두 번째 풀이는 최소 외접원 문제를 해결하는 알고리즘을 응용합니다.
- 이 풀이를 이해하기 전 LP-type 문제에 대해 이해하고 있으면 좋습니다.

- LP-type 문제란 낮은 차원의 선형 계획법 문제와 유사한 성질을 가지는 어떤 형태의 문제들을 통칭하는 이름입니다.
- LP-type 문제는 다음과 같이 정의됩니다.
- 주어진 원소들의 집합 S가 존재할 때, 어떤 함수 f가 S의 부분집합을 어떤 순서가 잘 정의된 집합의 원소로 대응시킨다고 합시다.
- 이 때, f는 다음과 같은 성질을 만족합니다.
- $A \subseteq B \subseteq S$일 때, $f(A) \leq f(B) \leq f(S)$입니다.
- $A \subseteq B \subseteq S$이며, x가 S의 한 원소일 때, $f(A) = f(B) = f(A \cup \{x\})$이면 $f(A) = f(B \cup \{x\})$입니다.
- 이 때, $f(B) = f(S)$인 가장 작은 B를 이 LP-type 문제의 Basis라고 부르며, 이 Basis를 찾는 것이 LP-type 문제의 목표입니다.

F.
경찰서 (풀이 2)

min_enclosing_circle, randomization

출제자 의도: **Challenging**

F.
경찰서 (풀이 2)

min_enclosing_circle, randomization

출제자 의도: **Challenging**

- 이 문제가 LP-type 문제인지 엄밀하게 증명하는 것은 쉽지 않습니다.
- 본 에디토리얼에서는 다음과 같은 방법을 사용해 증명합니다.
- Quasi-convex한 함수들의 집합이 주어질 때, 함숫값들의 최댓값을 최소화하는 문제는 LP-type 문제로 알려져 있습니다.
- "어떤 점에서 직선까지의 거리"는 Quasi-convex하며, 이 문제에서는 이 거리들의 최댓값을 최소화할 것을 요구하고 있습니다.
- 따라서 이 문제는 위에서 설명한 문제의 특수한 경우이며, LP-type입니다.

F.
경찰서 (풀이 2)

min_enclosing_circle, randomization

출제자 의도: **Challenging**

- LP-type 문제를 푸는 방법으로는, 무작위화를 사용하는 $O(n)$ 시간 복잡도의 여러 알고리즘들이 알려져 있습니다.
- 그 중 대표적인 것이 "최소 외접원 문제"를 해결하는 알고리즘으로 잘 알려진 Welzl's Algorithm입니다.

F.
경찰서 (풀이 2)

min_enclosing_circle, randomization

출제자 의도: **Challenging**

- Welzl's Algorithm을 다음과 같이 변형하면 문제를 해결할 수 있습니다.
- $\emptyset$에 대응되는 f의 값은 0입니다. 이는 어떤 임의의 점을 중심으로 가지는 반지름이 0인 원에 대응됩니다.
- 크기가 1인 집합에 대응되는 f의 값은 0입니다. 이는 직선 위에 있는 어떤 임의의 점을 중심으로 가지는 반지름이 0인 원에 대응됩니다.
- 크기가 2인 집합에 대응되는 f의 값은 0입니다. 이는 두 직선의 교점을 중심으로 가지는 반지름이 0인 원에 대응됩니다.
- 크기가 3인 집합에 대응되는 f의 값은 0 이상입니다. 이는 세 직선으로 만들어지는 삼각형의 내접원에 대응됩니다.

F.
경찰서 (풀이 2)

min_enclosing_circle, randomization

출제자 의도: **Challenging**

- 시간 복잡도는 무작위화를 사용할 때 $O(n)$입니다. 무작위화를 사용하지 않으면 최악의 경우 $O(n^3)$의 시간 복잡도를 얻을 수도 있습니다.

F.
경찰서
(통과하지 않는 풀이들)

출제자 의도: **Challenging**

- 삼분탐색 2개를 사용하여 함수의 최솟값을 찾는 풀이는 대회 중에는 통과되었으나, 통과되면 안 됩니다. 직선의 식에 대응되는 값의 범위만으로는 답에 해당하는 원의 중심의 범위를 유추하기 어렵습니다. 실제로 답에 해당하는 원의 중심의 좌표가 x축, y축 모두 절댓값이 10^{11}을 초과하는 입력이 존재합니다.
- 흔히 "경사하강법"이라고 알려진 풀이는 통과하지 않습니다. 해당하는 풀이는 실제로는 경사하강법이 아니라 Subgradient Method라고 불러야 하는데, n과 $1/\varepsilon$ 둘 모두에 대해 다항 시간 복잡도를 가지므로 이 문제에서는 통과하지 않습니다.
- 이외에도 함수의 최적화에 사용하는 Line search, Nelder-Mead, Pattern search 등의 방법도 통과하지 않습니다.

G.
공약수
(More Huge)

number_theory,
meet_in_the_middle

출제자 의도: **Challenging**

- 제출 44회, 정답 6명 (정답률 13.636%)
- First Solve: wider93, 80분

G.
공약수
(More Huge)

number_theory,
meet_in_the_middle

출제자 의도: **Challenging**

- 이 문제는 $O(\sqrt{n}\log n)$의 잘 알려진 해법이 있습니다. 그러나, 이 문제에서는 쿼리 하나조차 제대로 답하지 못할 것입니다.
- 시간복잡도를 얼마나 개선할 수 있을까요? 한 번 생각해 봅시다.
- 이 에디토리얼은 여러분이 $O(\sqrt{n}\log n)$ 풀이를 알고 있다는 전제 하에 설명합니다.

G.
공약수
(More Huge)

number_theory,
meet_in_the_middle

출제자 의도: **Challenging**

- 우선, 어떤 수 n의 약수가 대략적으로 $O(n^{0.33})$개라는 사실을 생각해 볼 수 있습니다.
- $O(\sqrt{n}\log n)$ 풀이에서 약수만을 돌아보도록 하면 약 $O(n^{0.33}\log n)$이라는 시간 복잡도를 얻을 수 있습니다.
- 하지만 여전히 너무 느립니다.

G.
공약수
(More Huge)

number_theory,
meet_in_the_middle

출제자 의도: **Challenging**

- 반드시 약수를 모두 돌아볼 필요는 없습니다. 약수 중에는 절대로 답이 될 수 없는 것도 있습니다.
- 예를 들어, 8의 약수 쌍 중에는 2와 4도 있습니다.
- 이 두 수는 모두 8의 약수이지만, 최대 공약수가 1이 아니기 때문에 정답이 되는 쌍을 구성할 수 없습니다.

G.
공약수
(More Huge)

number_theory,
meet_in_the_middle

출제자 의도: **Challenging**

- 최대 공약수가 1인 쌍만을 탐색할 방법이 있을까요?
- 산술의 기본 정리에 따라 모든 양의 정수 n은 $p_1^{e_1}p_2^{e_2}p_3^{e_3}\dots p_k^{e_k}$꼴로 분해할 수 있습니다. 이렇게 어떤 정수를 분해한 것을 n의 소인수 분해라고 합니다.
- 이때, 두 양의 정수 $n = p_1^{e_1}p_2^{e_2}\ p_3^{e_3}\dots p_k^{e_k}$와 $m = p_1^{f_1}p_2^{f_2}p_3^{f_3}\dots p_k^{f_k}$의 최대 공약수는 다음과 같습니다.
- $p_1^{\min(e_1,f_1)}p_2^{\min(e_2,f_2)}\ p_3^{\min(e_3,f_3)}\dots p_k^{\min(e_k,f_k)}$

G.
공약수
(More Huge)

number_theory,
meet_in_the_middle

출제자 의도: **Challenging**

- 따라서 두 수 n, m의 최대공약수가 1이기 위해서는 각 소수 p_i에 대응되는 지수 e_i와 f_i 둘 중 하나가 0이어야 합니다.
- 여기서 n, m은 실제로는 L/G의 약수이므로, L/G의 각 소인수에 대응되는 지수를 모두 n 또는 m 중 하나에만 곱해줘야 한다는 의미가 됩니다.
- 이제 이것은 Subset Sum 문제의 변형인 Subset Product 문제와 같게 됩니다.

G.
공약수
(More Huge)

number_theory,
meet_in_the_middle

출제자 의도: **Challenging**

- 이렇게 찾아낸 규칙에 따라 백트래킹을 시행하면 시간 복잡도가 어떻게 될까요?
- 양의 정수 n의 서로 다른 소인수의 개수를 $\omega(n)$이라고 합시다.
- 이 때, $\omega(n)$의 최댓값이 k로 갱신되는 시점의 n은 첫 소수 k개의 곱임을 증명할 수 있습니다.
- 또한, 이 결과와 소수 정리를 사용하여, $\omega(n) = O(\frac{\log n}{\log \log n})$임이 증명된 바 있습니다.
- 정확한 값으로 보자면, 제한 내에서 $\omega(n)$의 최댓값은 15이며, 이 값은 $n = 614889782588491410$에서 최초로 등장합니다.

G.
공약수
(More Huge)

number_theory,
meet_in_the_middle

출제자 의도: **Challenging**

- 이에 따라서, 시간 복잡도는 $O(2^{\omega(n)})$이 됩니다.
- GCD 함수를 전혀 사용하지 않으므로, 이 풀이의 시간 복잡도에서는 log가 배제됩니다.
- 이는 그 자체로 상당히 빠른 시간 복잡도이지만, 여전히 시간 제한 내에 통과하지는 않습니다.
- 가지치기와 같은 여러 가지 휴리스틱을 통해 이 방법의 실행 시간을 개선하는 방법이 알려져 있지만, 여전히 50000개의 쿼리에 답하기에는 다소 어려움이 있습니다.

G.
공약수
(More Huge)

number_theory,
meet_in_the_middle

출제자 의도: **Challenging**

- Meet-in-the-middle 방법을 통해 시간 복잡도를 $O(2^{\omega(n)/2}\omega(n))$으로 개선합시다.
- 집합을 반으로 쪼개서 각각 백트래킹을 시행합니다.
- 그 결과로 얻은 Subset Product들의 배열 2개를 정렬해 줍시다.
- 그 뒤에는 이분탐색이나 투 포인터 알고리즘을 통해 최적의 해법을 찾을 수 있습니다.
- 이렇게 하면 약 500ms 정도에 문제를 해결할 수 있습니다.
- 750ms 정도의 시간 제한에 500ms의 실행 시간을 가지는 풀이라면 구현에 따라 시간 초과가 나기 쉽습니다. 다음 슬라이드부터는 이 문제를 해결하는 $O(2^{\omega(n)/2})$ 해법을 제시합니다.

G.
공약수
(More Huge)

number_theory,
meet_in_the_middle

출제자 의도: **Challenging**

- Subset Product들을 구하는 다음과 같은 알고리즘을 생각합시다.
- Subset Product들의 배열 P를 관리합시다. P의 초기값은 $[1]$로 둡니다.
- 각 원소 k에 대해, $P = [p_1, p_2, \ldots, p_v]$였던 P를 $P + kP$로 갱신합니다. 이 때 kP는 $[kp_1, kp_2, \ldots, kp_v]$를 의미합니다.
- 이 알고리즘의 시간 복잡도를 분석해 봅시다.
- $T(n) = T(n-1) + 2^n$이며, $T(1) = O(1)$입니다.
- 따라서 $T(n) = 2^n + 2^{n-1} + 2^{n-2} + \cdots + 1 = O(2^n)$입니다.
- 그러나, 이 알고리즘은 배열을 정렬된 상태로 반환하지 않기에, 배열을 정렬하는 데에는 여전히 $O(n2^n)$의 시간 복잡도가 요구됩니다.

G.
공약수
(More Huge)

number_theory,
meet_in_the_middle

출제자 의도: **Challenging**

- 선술한 알고리즘을 변형해 정렬된 상태의 배열을 반환하는 알고리즘을 만들 수 있습니다.
- 어떤 양의 정수 k에 대해, 길이 v의 배열 P가 정렬되어 있다면, kP 또한 정렬되어 있습니다.
- 따라서, 두 배열 P와 kP를 $O(v)$ 시간에 정렬된 상태로 합칠 수 있습니다.
- 이 방법을 사용해 선술한 알고리즘을 변형하면, 시간 복잡도는 동일하게 $O(2^n)$이면서 정렬된 상태의 배열을 반환하게 할 수 있습니다.

G.
공약수
(More Huge)

number_theory,
meet_in_the_middle

출제자 의도: **Challenging**

- 이 알고리즘을 선술한 Meet-in-the-middle 해법에 적용합시다.
- 이렇게 하면 $O(2^{\omega(n)/2})$ 시간 복잡도로 두 Subset Product 배열을 모두 정렬된 상태로 얻을 수 있습니다.
- 두 배열이 모두 정렬되어 있다면, 나머지 부분은 투 포인터 기법으로 $O(2^{\omega(n)/2})$ 시간 복잡도로 해결할 수 있습니다.

G.
공약수
(More Huge)

number_theory,
meet_in_the_middle

출제자 의도: **Challenging**

- 이외에도 자잘하게 고려해야 할 사항들이 있습니다. 이를테면, L/G를 소인수분해한 결과는 어떻게 찾아내야 할까요?
- 이는 각 소수에 대응되는 지수를 빼는 것으로 해결할 수 있습니다.
- C++에서는 std::set_difference나 std::map으로, Python에서는 Counter로 이를 수행할 수 있습니다.

참여해주셔서 감사합니다.

Thank **you** for participating in the contest!