Quest

우아한테크코스 알고리즘 스터디

Week 3. 재귀와 분할 정복

이번 주차의 목표

- ✓ 이번 주차에서는 여러분이 알고 계신 재귀를 이용해 분할 정복이라는 묘기를 부려보는 시간을 가질 겁니다!
- ✓ 여러분이 분할 정복이라는 알고리즘을 새롭게 학습하는 것도 중요하지만, 가장 중요한 목표는 재귀에 익숙해지는 것입니다.
- ✓ 여러분에게 정해 드리는 목표는 **A, B, C번 문제의 해결**입니다. 더 연습을 하고 싶으시다면 시간이 나면 풀어보세요.
- ✓ 엄밀하게 시간복잡도를 증명하느라 설명이 어려울 것이라 생각합니다. 한 번에 모든 내용을 이해하지 않아도 좋습니다. 지금 **당장은 분할 정복의 아이디어**만 가져가신다면 충분하다고 생각합니다.
- ✓ 내용이 어렵기는 하지만 조금이라도 부담없이 연습을 진행하셨으면 좋겠습니다. 저도 가능한 한 상세히 설명해 보도록 하겠습니다.

재귀란

- 재귀는 여러분들도 이미 아시다시피, **자기 자신을 호출하는 것**을 의미합니다. 어떤 함수 내부에 자기 자신을 실행시키는 로직이 포함되어 있다면 그 함수를 재귀함수라 부릅니다.
- 자기 자신이 내부에서 호출될 수 있고, 파라미터가 같으면 대부분 같은 일을 수행하는 재귀함수의 특성상, 가만히 내버려 두면 무한 루프에 빠질 수도 있습니다. 따라서 재귀함수의 경우 특정 조건에서는 더 이상 실행되지 않도록 **종료 조건**이 요구됩니다.
- 비슷한 연산을 반복하고 종료조건이 있다는 특징을 떠올린다면 반복문을 떠올리실 수 있을 것입니다. 실제로 재귀와 반복문은 닮은 점이 정말로 많으며, 반복문으로 풀리는 문제는 대개 재귀로 풀 수 있고, 재귀로 풀리는 문제는 반복문으로 풀 수 있는 경우가 많습니다.

재귀를 사용하는 이유

- ✓ 그렇다면, 알고리즘에서 "반복문을 사용하면 그만인데 왜 재귀함수를 사용하느냐?" 라고 의문이 드실 수도 있다고 생각합니다.
- ✓ 그 이유는, 몇몇 문제의 경우 재귀를 사용했을 때 반복문을 사용했을 때보다 더 직관적으로 풀 수 있는 있는 문제들이 많기 때문입니다.
- ✓ 재귀함수가 무슨 일을 하게 되는지는 파라미터의 값에 달렸습니다.이를 이용하면 문제풀이에 필요한 모든 상태와 값을 관리해야 하고, 반복문의 중첩이 많아 복잡해질 수 있는 반복문을 이용한 해결 방법과 비교했을 때 재귀함수의 경우 파라미터만 신경쓰면 되므로 문제가 간단하고 직관적이게 되며, **복잡한 문제를 여러 재귀함수로 나누어 해결할 수 있게 됩니다.**
- ✓ 알고리즘에서 재귀를 사용하기 편한 상황 중 하나는, **하나의 복잡한 문제를 여러 개의 작은 문제로 나누어 해결**하기 좋은 때라 할 수 있겠습니다.

재귀를 사용하는 이유

- ✓ 반복문과 재귀 중 무엇을 사용해야 하는지에 대한 명확한 정답은 정해져 있지 않습니다. 여러분이 편하다고 생각하시는 방법을 사용하시기 바랍니다.
- ✓ 다음 페이지에서 본격적으로 분할 정복을 들어가 봅시다!

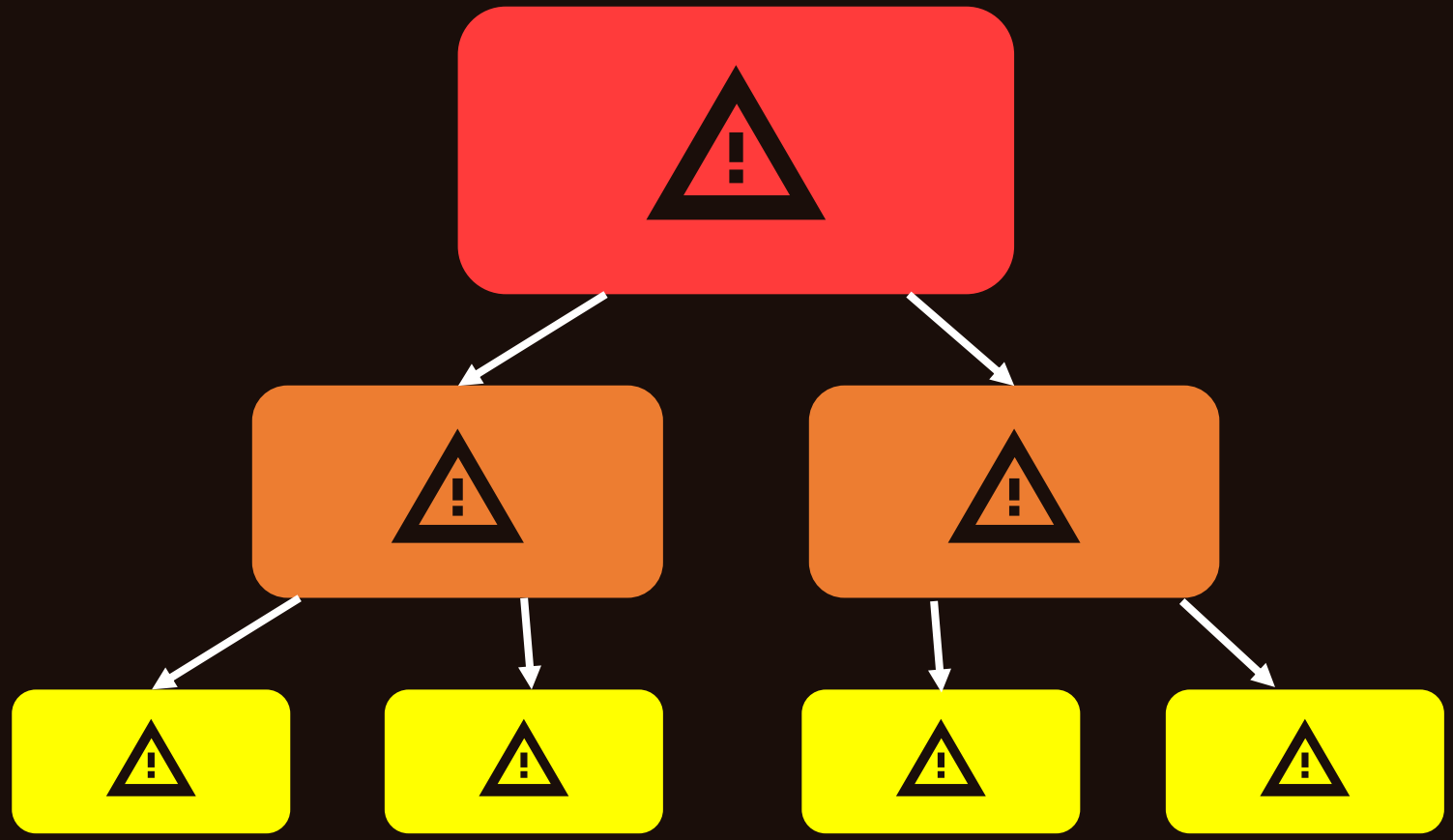
분할 정복이란

- 분할 정복이란, **해결하기 힘든 거대한 문제를 해결하기 쉬운 작은 문제로 먼저 쪼개고, 해결한 작은 문제들을 합쳐 나가며 점차 큰 문제를 해결해나가는 방식**입니다.
- 여기 마침 매우 해결하기 힘든 큰 문제가 있습니다.

분할 정복이란

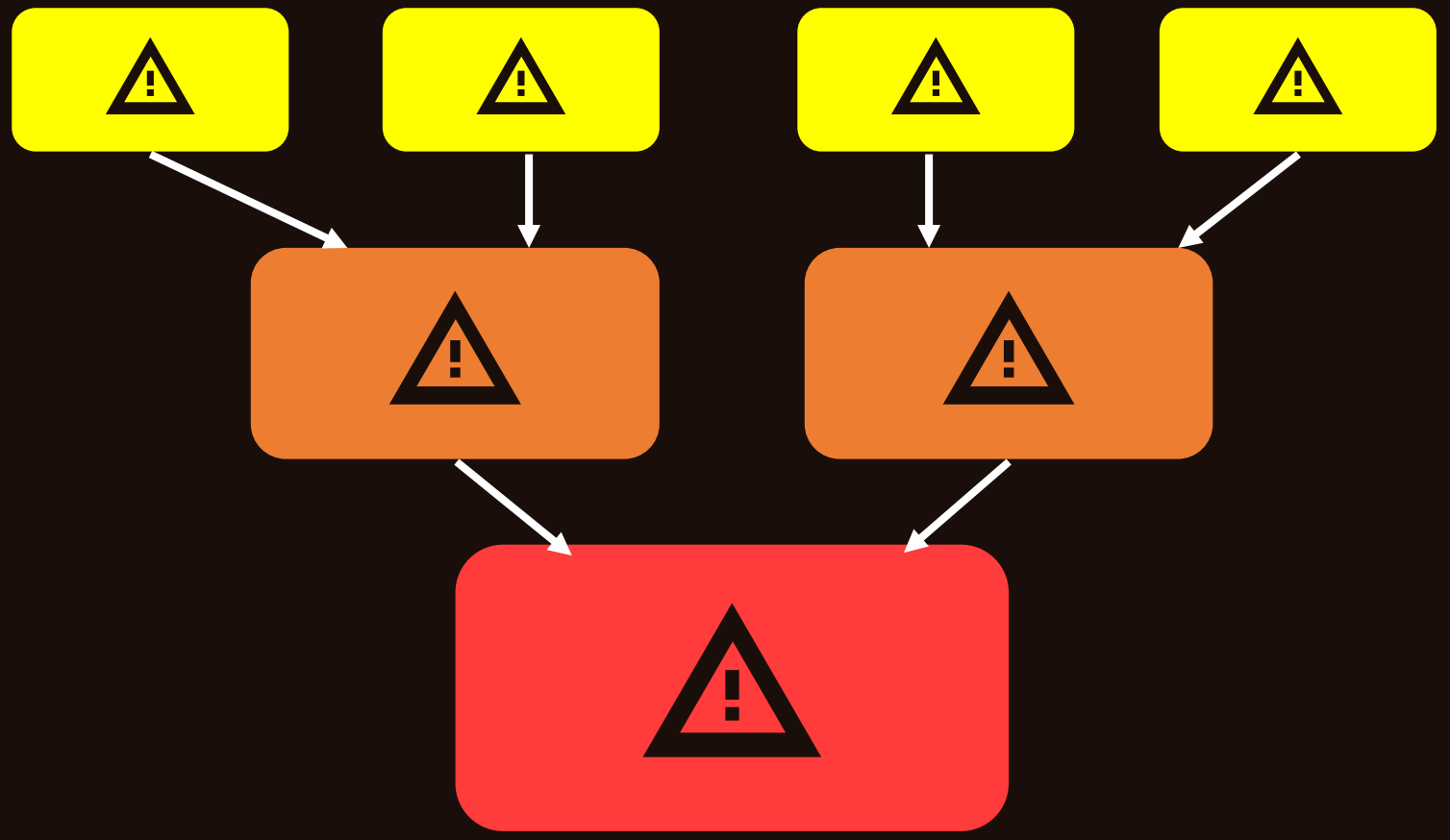
- ✓ 이 문제는 도저히 당장 해결할 기미가 보이지 않습니다. 대신, **이 문제를 작은 문제로 쪼개 볼 수는 있어 보입니다.**

"분할"



분할 정복이란

- 충분히 **작게 쪼개진 문제는 해결할 수 있습니다**. 그리고 해결한 **작은 문제를 합치면 점차 큰 문제를 해결할 수 있게 되며**, 이런 식으로 해결하다 보면 가장 큰 문제를 해결할 수 있게 됩니다. 이것이 분할 정복입니다.



"정복"

분할 정복이란

- ✓ 분할 정복을 사용하기 위해서는 재귀를 주력으로 사용하게 됩니다.
- ✓ 이렇게 설명만 했을 때는 와닿지 않으실 것이라 생각됩니다. 다음 페이지에서부터는 본격적으로 문제를 풀면서 분할 정복에 대한 감을 잡고, 재귀에 익숙해져 봅시다.

2 A. 피보나치 수 5

- ✓ 우리는 이미 다이나믹 프로그래밍 주차에서 이 문제를 풀었습니다. 해결 방식을 아시겠지만, 이번에는 재귀로 풀어 봅시다.
- ✓ 먼저, 해결하고자 하는 **큰 문제**를 떠올려 봅시다. 이 문제에서는 **n번째 피보나치 수를 구하는 것**이 큰 문제가 되겠군요.
- ✓ 이 큰 문제는 당장 해결할 수는 없습니다. 20번째 피보나치 수가 무엇이냐고 묻는다면 대답하실 수 없을 것입니다. 대답할 수 있다면 그건 변태입니다.

2 A. 피보나치 수 5

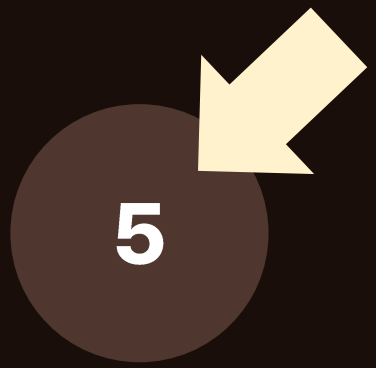
- ✓ 우리는 이 **큰 문제**를 당장 해결할 수는 없지만, 우리가 알고 있는 정보를 이용해 **더 작은 정보**로 나눌 수 있습니다.
- ✓ 우리가 알고 있는 정보는 아래의 두 가지입니다.
  1) i번째 피보나치 수는 i – 1번째 피보나치 수와 i – 2번째 피보나치 수의 합과 같다.
  2) 0번째 피보나치 수의 값은 0이고, 1번째 피보나치 수의 값은 1이다.
- ✓ 이 정보를 토대로 큰 문제를 쪼개봅시다. 우리는 **n번째 피보나치 수의 값**이 뭔지는 모르지만 어쨌든 그것이 **n – 1번째 피보나치 수**와 **n – 2번째 피보나치 수**의 합임은 알고 있습니다. 그러면 저 두 값을 구하면 큰 문제를 해결 것과 동일하겠지요?
- ✓ **하나의 큰 문제를 해결해야 하는 것을, 두 개의 조금 더 작은 문제를 해결하는 것으로 쪼갰습니다. (분할)**

2 A. 피보나치 수 5

- ✓ 이제 이 쪼갠 문제들을 더 작은 문제들로 쪼갤 수 있습니다.
- ✓ 4번째 피보나치 수는 2번째 피보나치 수와 3번째 피보나치 수의 합과 같습니다.
  3번째 피보나치 수는 1번째 피보나치 수와 2번째 피보나치 수의 합과 같습니다.
  2번째 피보나치 수는 0번째 피보나치 수와 1번째 피보나치 수의 합과 같습니다.
- ✓ 이런 식으로, 점점 문제를 쪼개다 보면 이 문제는 **0번째 피보나치 수, 또는 1번째 피보나치 수**를 구해야 하는 문제까지 쪼개집니다.
- ✓ 위의 정보는 우리가 알고 있으니, 문제를 해결할 수 있게 됩니다, 그리고...
  0번째 수와 1번째 수를 알게 되니 2번째 피보나치 수가 무엇인지 알 수 있고,
  1번째 수와 2번째 수를 알게 되니 3번째 피보나치 수가 무엇인지 알 수 있고, ...
  이런 식으로 결합해 나가면, 결국 가장 큰 문제인 n번째 피보나치 수의 값을 구하는 것을 해결할 수 있게 됩니다!
- ✓ **가장 작은 단위로 쪼개진 문제를 해결하고, 이를 이용해 점차 더 큰 문제를 해결하는 것으로 가장 큰 문제를 해결할 수 있습니다. (정복)**

2 A. 피보나치 수 5

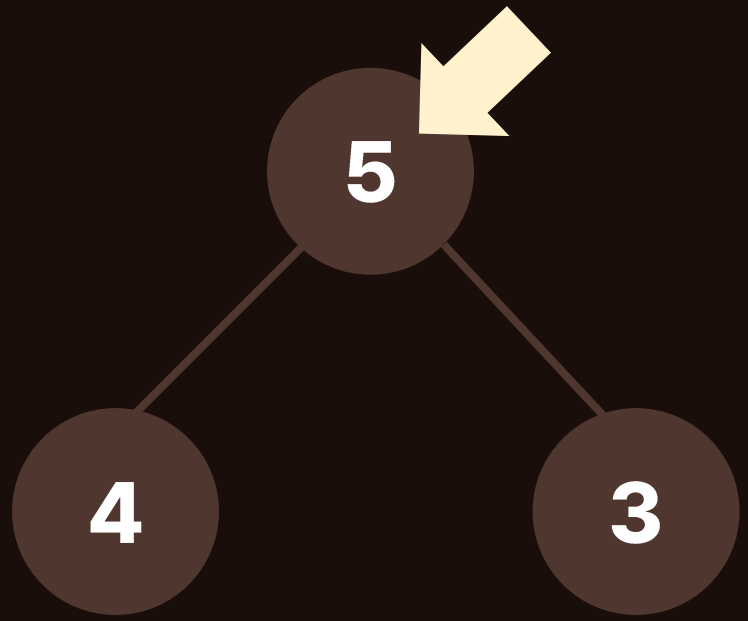
✓ 지금까지 설명한 과정을 그림으로 나타내 보겠습니다. 여기서는 5번째 피보나치 수를 구한다 가정해 보겠습니다. **화살표가 가리키는 부분은 현재 재귀함수가 진행중인 부분입니다.**



▶ 5번째 피보나치 수를 구하고 싶다.

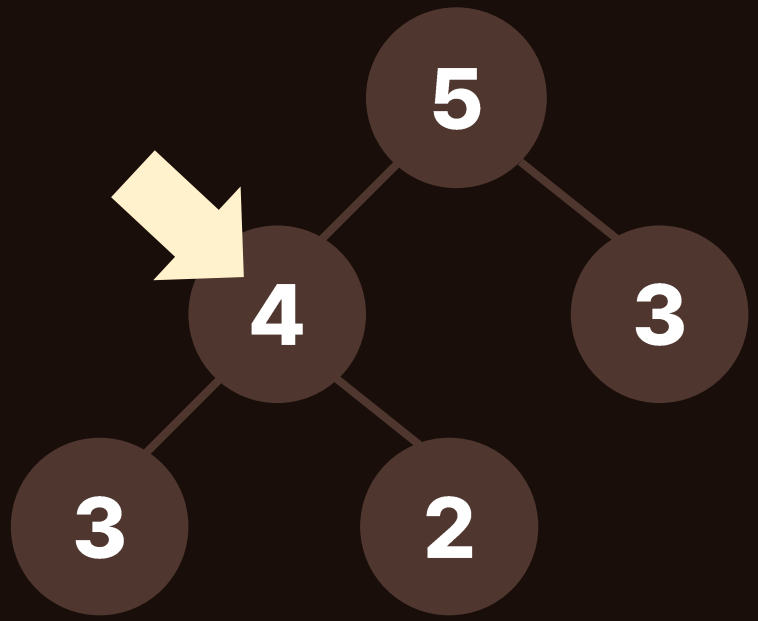
2 A. 피보나치 수 5

✓ 지금까지 설명한 과정을 그림으로 나타내 보겠습니다. 여기서는 5번째 피보나치 수를 구한다 가정해 보겠습니다.



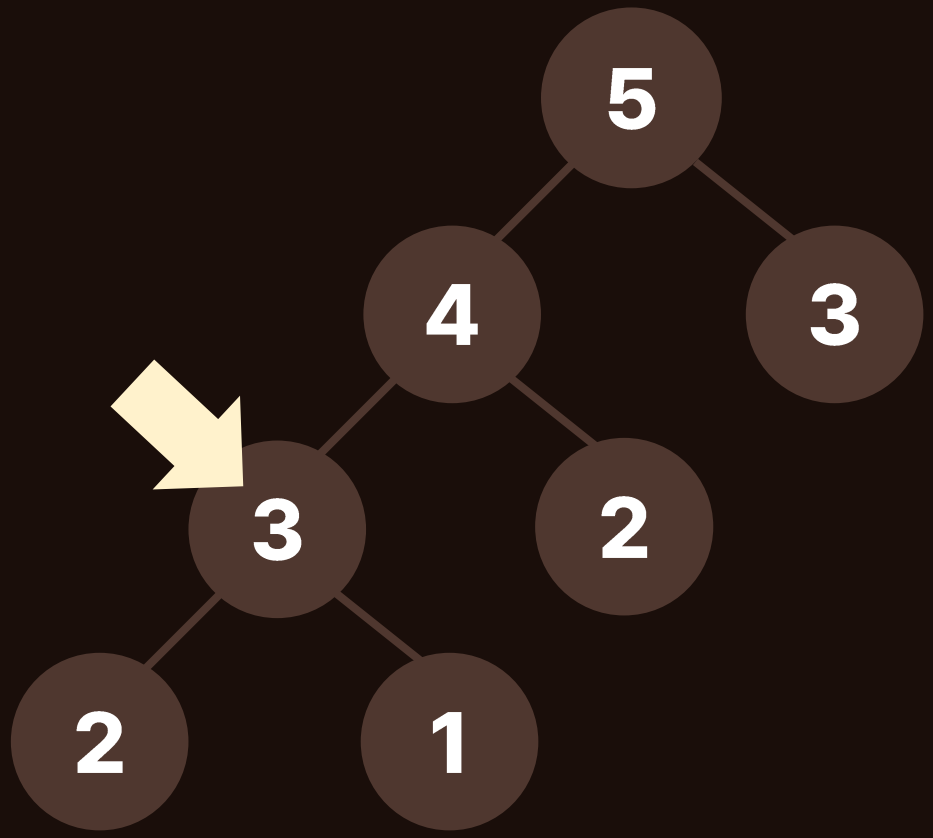
▶ 5번째 피보나치 수가 무엇인지는 모르겠지만,
3번째 피보나치 수와 4번째 피보나치 수의 합이라는 것은 알고 있다.

2 A. 피보나치 수 5



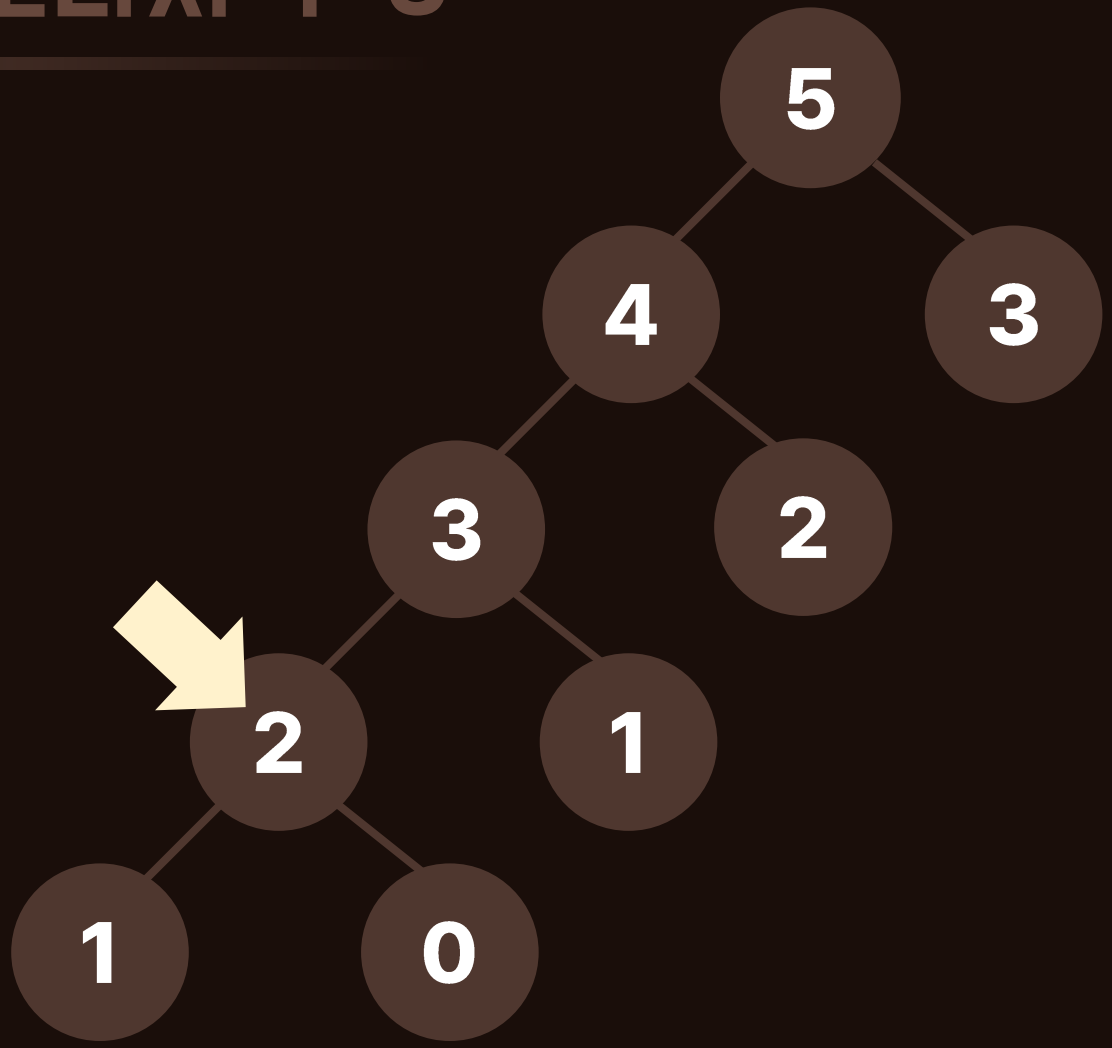
▶ 4번째 피보나치 수부터 먼저 구해보자. 역시 당장은 구할 수 없지만,
그것이 2번째 피보나치 수와 3번째 피보나치 수의 합임은 알고 있다.

2 A. 피보나치 수 5



▶ 3번째 피보나치 수부터 먼저 구해보자. 역시 당장은 구할 수 없지만,
그것이 1번째 피보나치 수와 2번째 피보나치 수의 합임은 알고 있다.

2 A. 피보나치 수 5



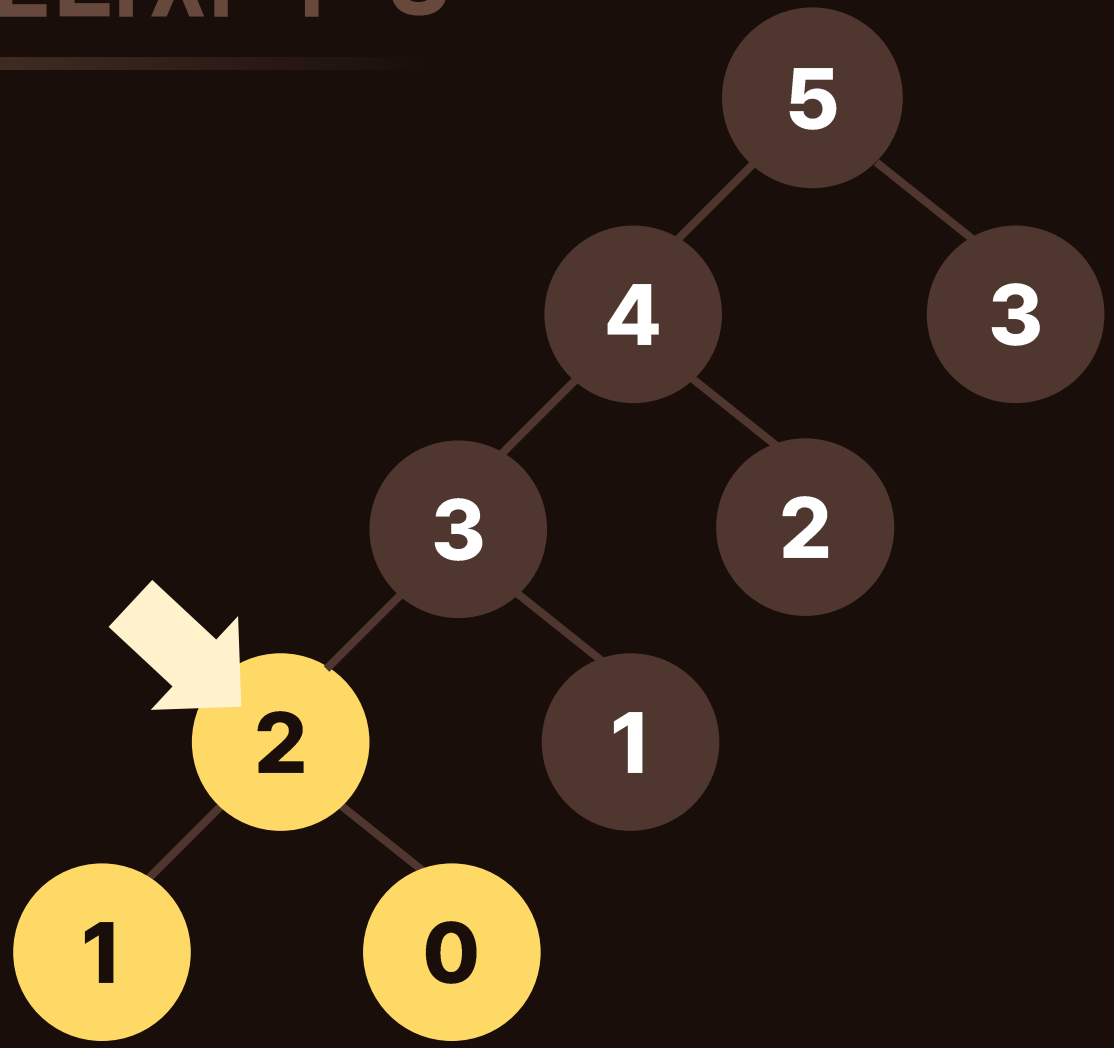
▶ 2번째 피보나치 수부터 먼저 구해보자. 역시 당장은 구할 수 없지만,
그것이 0번째 피보나치 수와 1번째 피보나치 수의 합임은 알고 있다.

2 A. 피보나치 수 5

화살표가 두 개지만, 두 함수를 동시에 재귀하고 있는 것은 아닙니다.
재귀 함수의 이름을 f라 한다면 f(1)을 한 후 f(0)을 하고 있는 것입니다.
앞으로 화살표가 여러 개 나올 때도 이렇게 이해해 주세요.

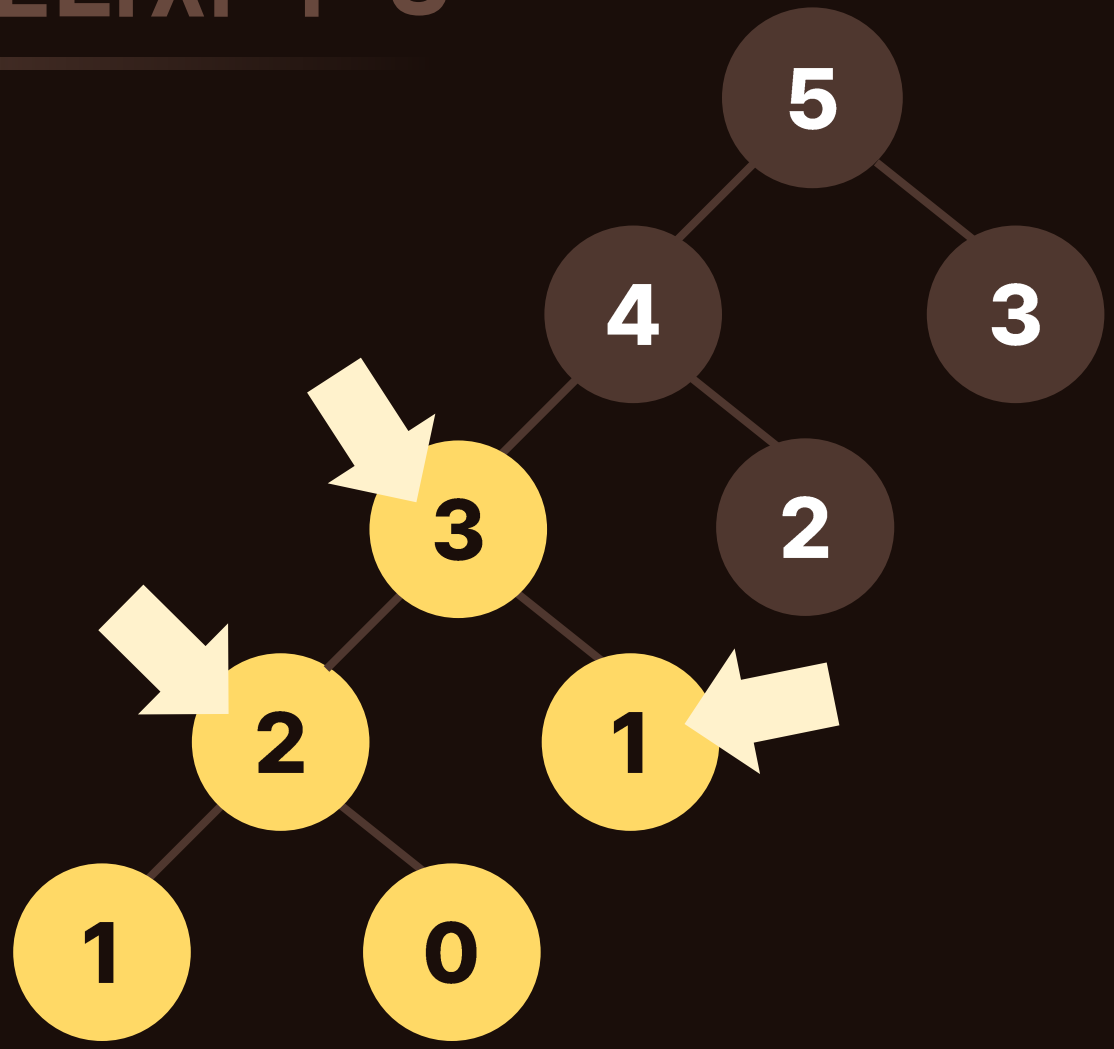
▶ 0번째 수와 1번째 수가 무엇인지는 우리가 알고 있다.
이 정보를 활용하면 2번째 수를 구할 수 있을 것이다.

2 A. 피보나치 수 5



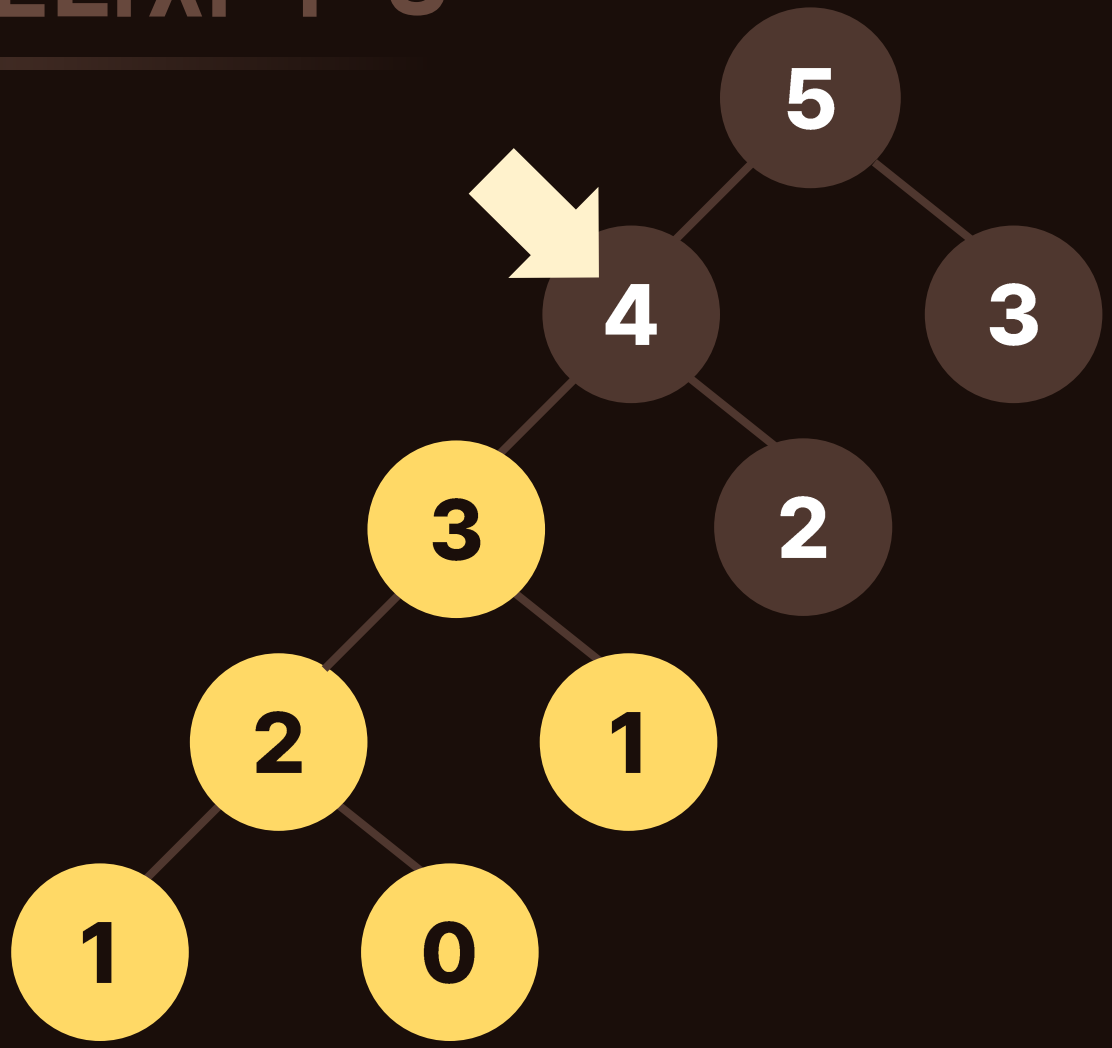
▶ 이제 2번째 피보나치 수가 무엇인지 알아냈다.

2 A. 피보나치 수 5



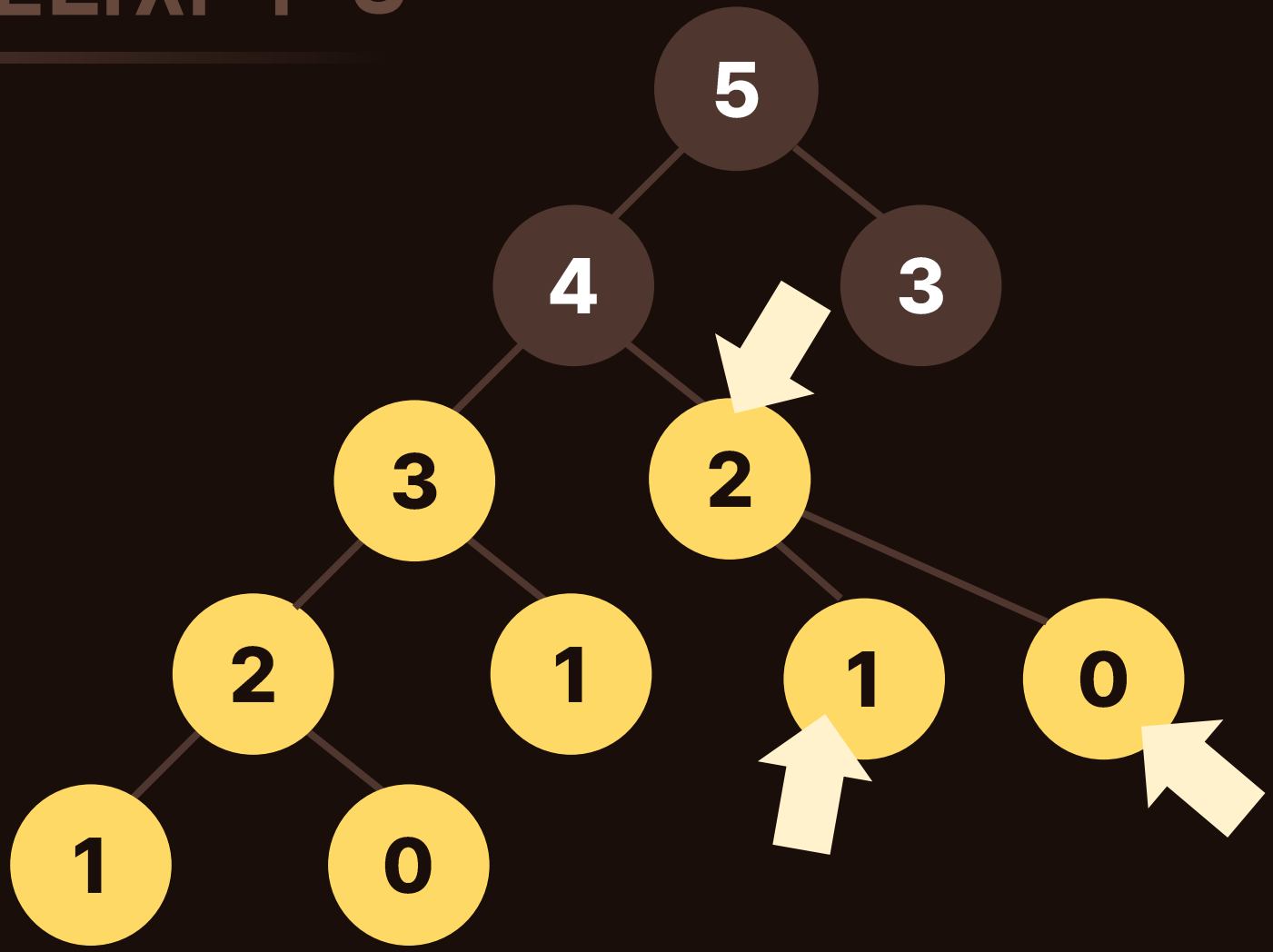
▶ 3번째 피보나치 수 역시 알아낼 수 있었다.
이를 구하기 위한 1번째 피보나치 수와 2번째 피보나치 수를 모두 알기에.

2 A. 피보나치 수 5



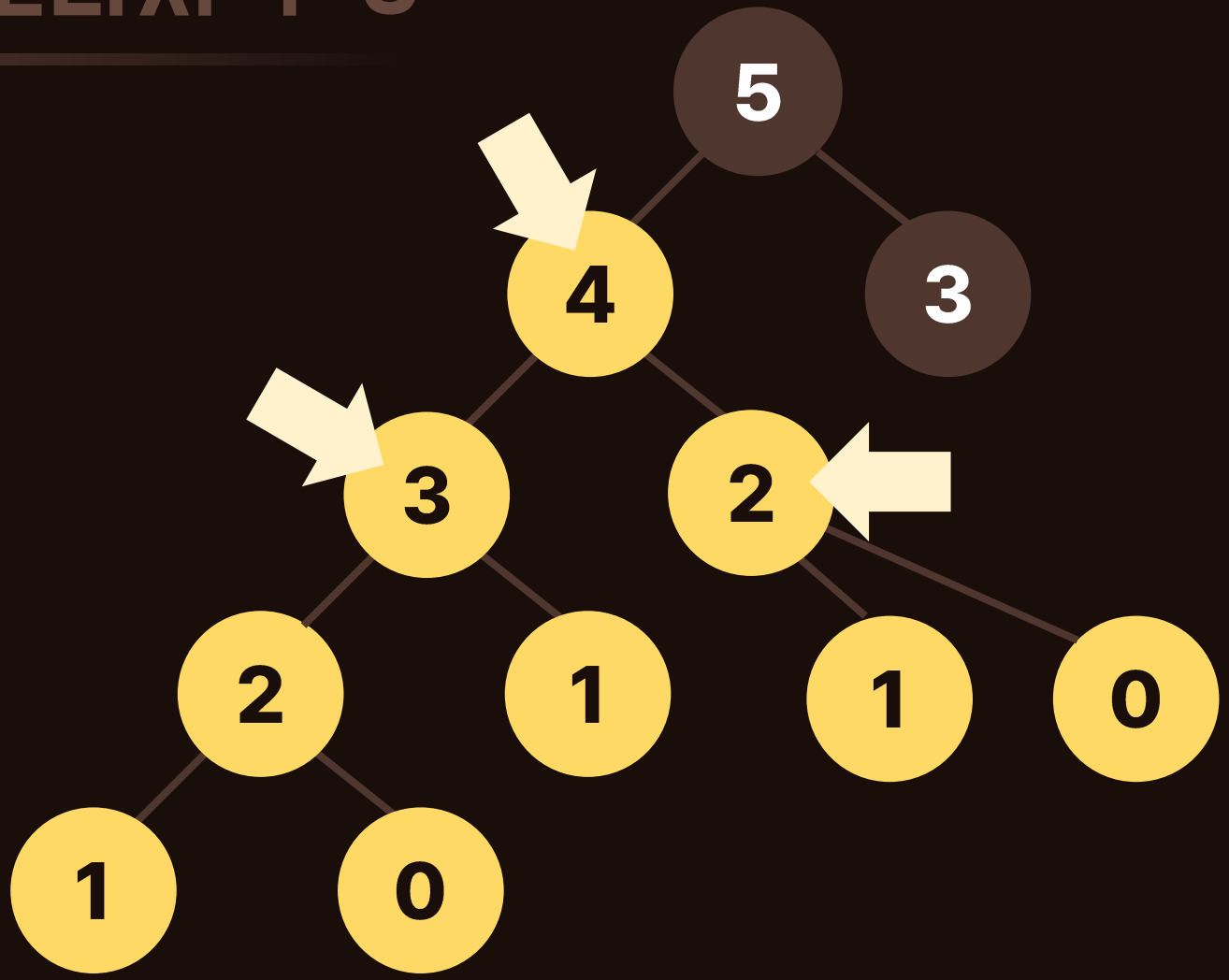
▶ 이제 4번째 피보나치 수를 구하기 위해서는 2번째 피보나치 수만 구하면 된다.
이 경우, 이전에 했던 것과 같은 방식으로, 계속해서 문제를 쪼개 답을 구하게 된다.

2 A. 피보나치 수 5



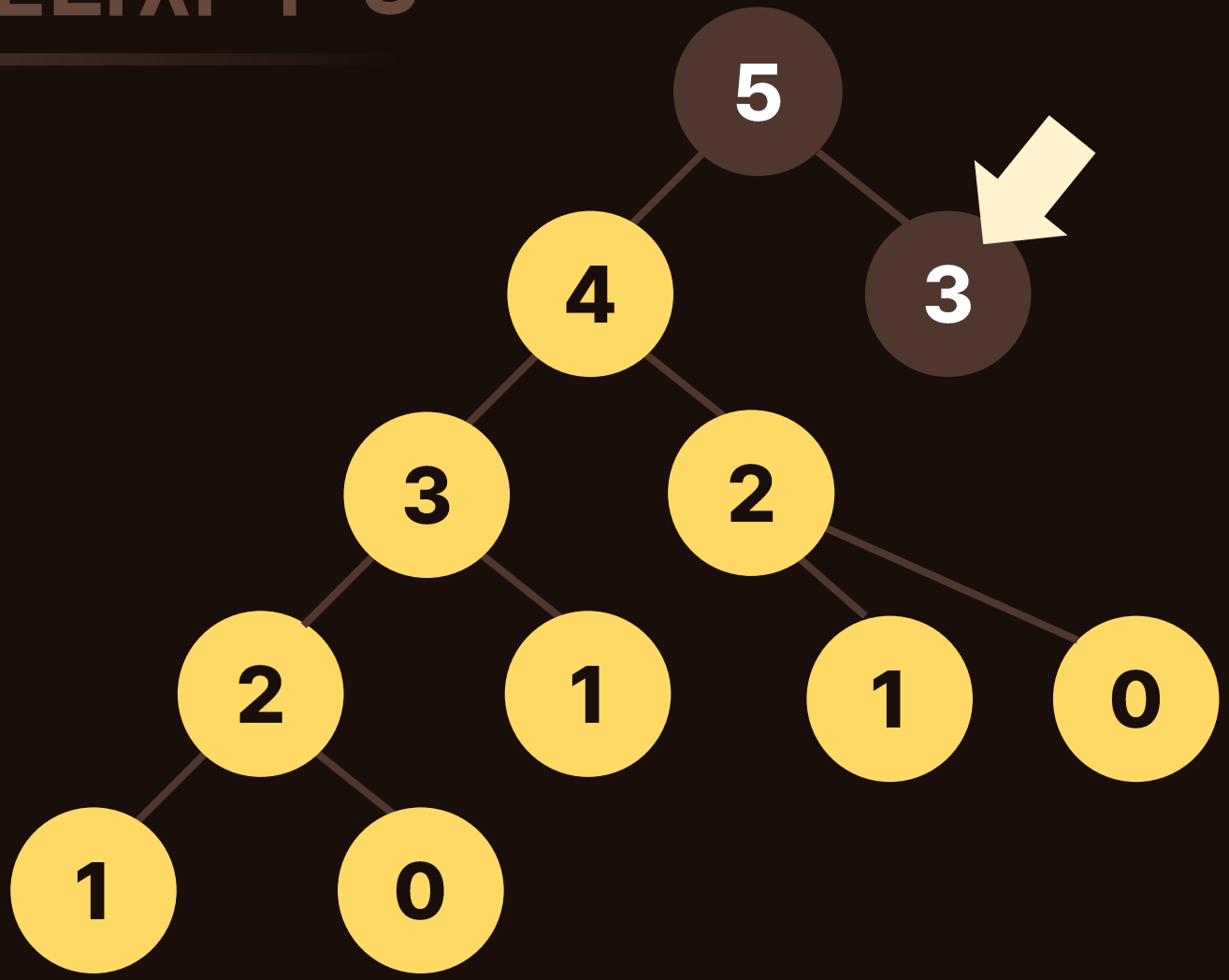
▶ 아마, 이런 과정을 거쳐 다시 2번째 피보나치 수를 구하게 될 것이다.
이미 다루었던 과정이므로 편의상 생략한다.

2 A. 피보나치 수 5



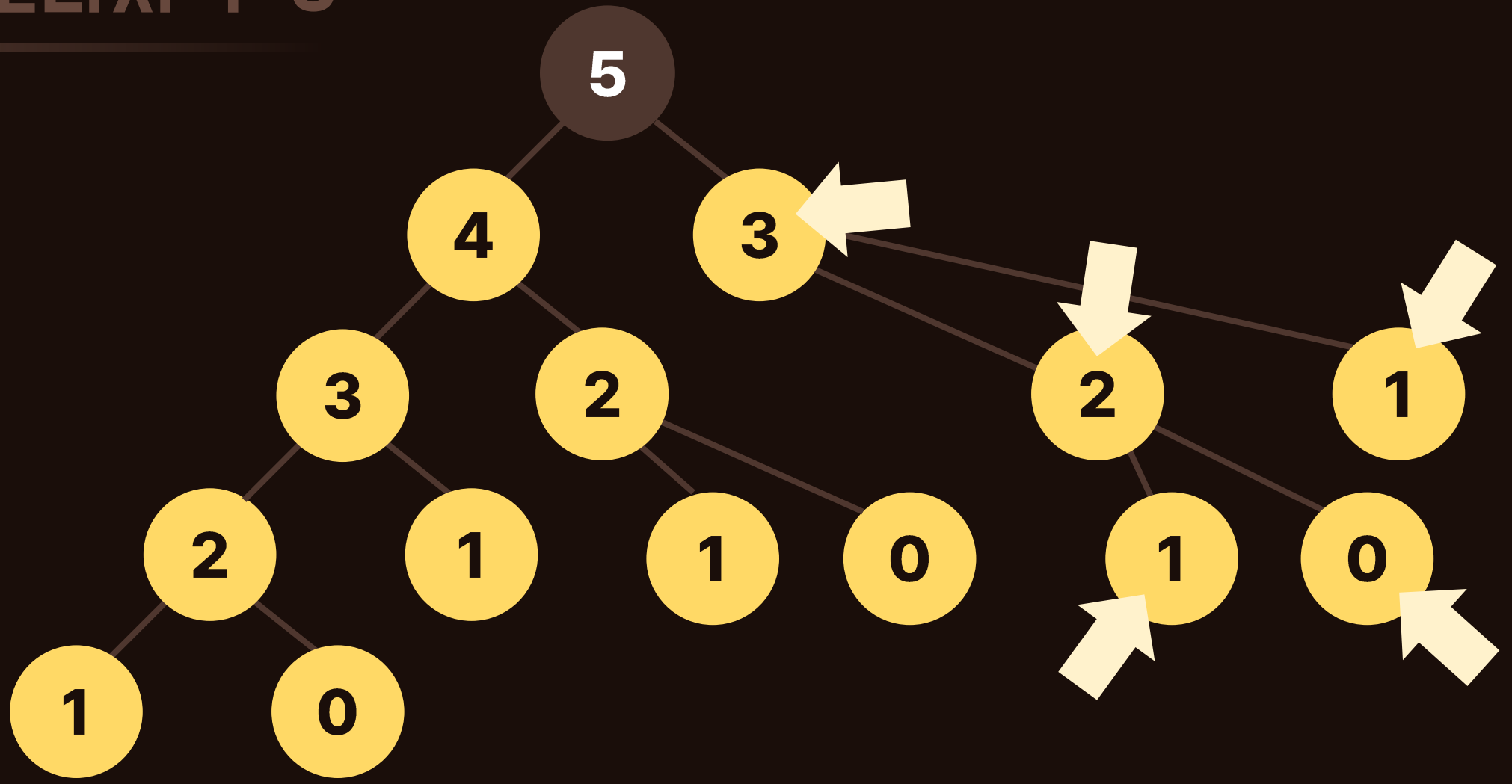
▶ 이제 우리는 2번째 피보나치 수와 3번째 피보나치 수가 무엇인지 알기에
4번째 피보나치 수를 구할 수 있게 된다.

2 A. 피보나치 수 5



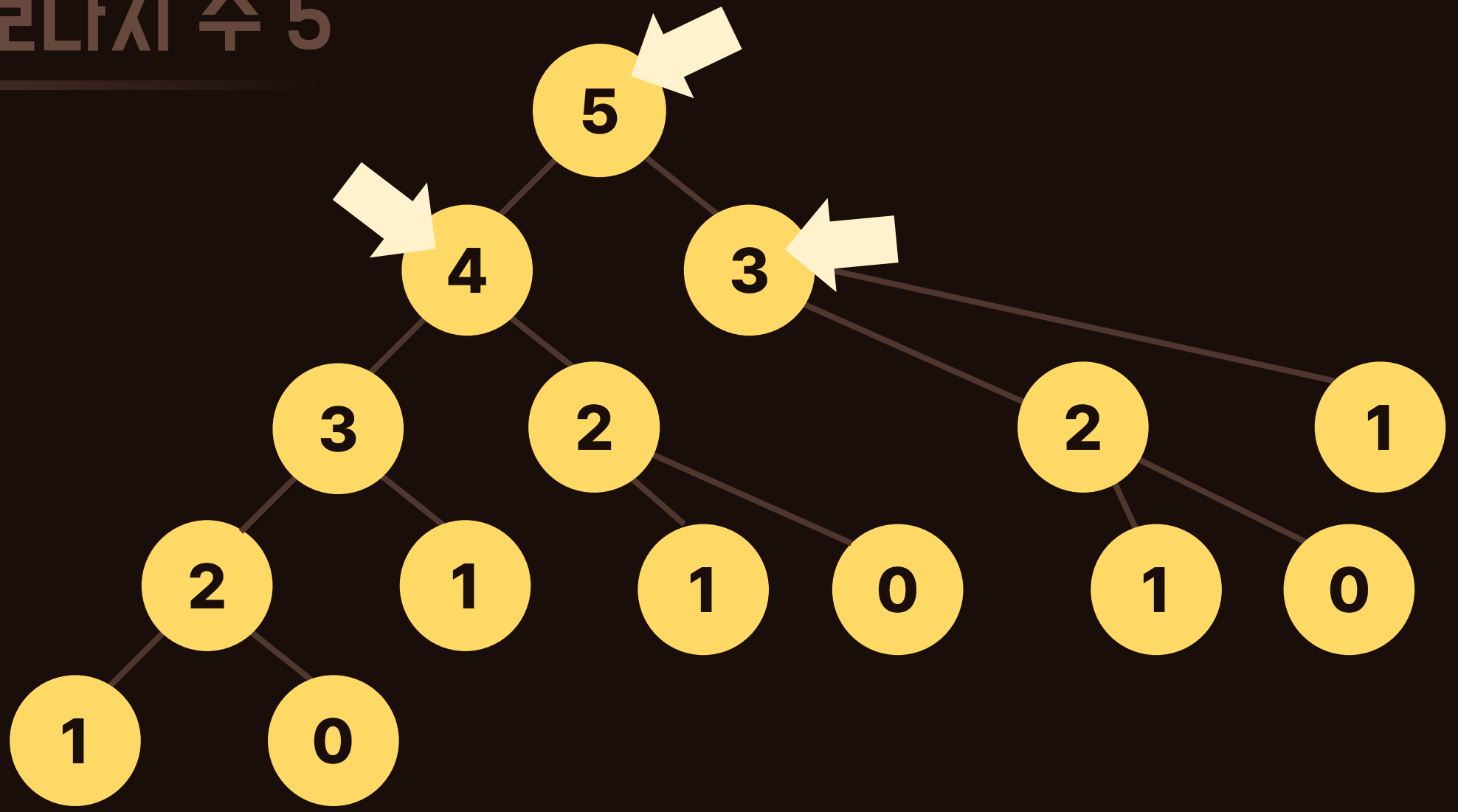
▶ 이제 최종 목표인 5번째 피보나치 수를 구하기 위해서는
3번째 피보나치 수가 무엇인지만 알면 된다.

2 A. 피보나치 수 5



▶ 이전과 같은 방법으로 재귀를 뻗어나가, 3번째 피보나치 수가 무엇인지 구하였다. 과정은 생략한다.

2 A. 피보나치 수 5



▶ 이제 3번째 피보나치 수와 4번째 피보나치 수가 무엇인지 모두 알기에
5번째 피보나치 수가 무엇인지 구할 수 있게 되었다. 문제 해결!

2 A. 피보나치 수 5

✓ 코드도 첨부해 드리겠습니다. **어떻게 더 작은 문제로 쪼개는지, 어느 조건에서 더 이상 재귀를 뻗어나가지 않고 값을 반환하는지(재귀의 종료조건)**에 중점을 두고 봐 주세요.

재귀의 종료조건

더 작은 문제로 쪼개는 부분

```
const N = Number(require('fs').readFileSync(0, 'utf-8'));

const getNthFibonacciNumber = (n) => {
  if (n === 0) {
    return 0;
  }

  if (n === 1) {
    return 1;
  }

  return getNthFibonacciNumber(n - 1) + getNthFibonacciNumber(n - 2);
}

console.log(getNthFibonacciNumber(N));
```

그 다음 문제들은요...

- ✓ 그 다음 문제들의 경우 난이도가 높아졌지만 근본적인 접근 방식은 같습니다.
- ✓ 피보나치 수의 경우 n번째 수는 n – 1번째 수와 n – 2번째 수의 합이라는 규칙을 여러분에게 대놓고 알려주지만, 그 다음 문제부터는 그 규칙을 대놓고 알려주지는 않을 것입니다. 직접 규칙을 찾아보시기 바랍니다.
- ✓ 그 다음 문제부터는 여러분들께 “규칙” 에 대한 단서를 설명하여 여러분의 문제풀이를 조금이나마 도와드리고자 합니다.

B. 재귀함수가 뭔가요?

- ✓ B번은 분할 정복 문제는 아니지만, 재귀를 연습해 볼 수 있는 문제입니다.
- ✓ "재귀함수가 뭔가요?" 와 "라고 답변하였지" 사이의 내용에 규칙성이 보이며, 그 사이의 내용을 출력하기 위해 재귀함수를 사용할 수 있습니다.
- ✓ 다음 페이지에서 좀 더 많은 예시를 보여드리도록 하겠습니다. 규칙을 유추해 보세요!

B. 재귀함수가 뭔가요?

✓ n = 1

어느 한 컴퓨터공학과 학생이 유명한 교수님을 찾아가 물었다.
"재귀함수가 뭔가요?"
"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을 통달한 선인이 있었어.
마을 사람들은 모두 그 선인에게 수많은 질문을 했고, 모두 지혜롭게 대답해 주었지.
그의 답은 대부분 옳았다고 하네. 그런데 어느 날, 그 선인에게 한 선비가 찾아와서 물었어."
____"재귀함수가 뭔가요?"
____"재귀함수는 자기 자신을 호출하는 함수라네"
____라고 답변하였지.
라고 답변하였지.

B. 재귀함수가 뭔가요?

✓ n = 2

어느 한 컴퓨터공학과 학생이 유명한 교수님을 찾아가 물었다.
"재귀함수가 뭔가요?"
"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을 통달한 선인이 있었어.
마을 사람들은 모두 그 선인에게 수많은 질문을 했고, 모두 지혜롭게 대답해 주었지.
그의 답은 대부분 옳았다고 하네. 그런데 어느 날, 그 선인에게 한 선비가 찾아와서 물었어."
____"재귀함수가 뭔가요?"
____"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을 통달한 선인이 있었어.
____마을 사람들은 모두 그 선인에게 수많은 질문을 했고, 모두 지혜롭게 대답해 주었지.
____그의 답은 대부분 옳았다고 하네. 그런데 어느 날, 그 선인에게 한 선비가 찾아와서 물었어."
________"재귀함수가 뭔가요?"
________"재귀함수는 자기 자신을 호출하는 함수라네"
________라고 답변하였지.
____라고 답변하였지.
라고 답변하였지.

B. 재귀함수가 뭔가요?

✓ n = 3

어느 한 컴퓨터공학과 학생이 유명한 교수님을 찾아가 물었다.
"재귀함수가 뭔가요?"
"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을 통달한 선인이 있었어.
마을 사람들은 모두 그 선인에게 수많은 질문을 했고, 모두 지혜롭게 대답해 주었지.
그의 답은 대부분 옳았다고 하네. 그런데 어느 날, 그 선인에게 한 선비가 찾아와서 물었어."
____"재귀함수가 뭔가요?"
____"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을 통달한 선인이 있었어.
____마을 사람들은 모두 그 선인에게 수많은 질문을 했고, 모두 지혜롭게 대답해 주었지.
____그의 답은 대부분 옳았다고 하네. 그런데 어느 날, 그 선인에게 한 선비가 찾아와서 물었어."
_____"재귀함수가 뭔가요?"
_____"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을 통달한 선인이 있었어.
_____마을 사람들은 모두 그 선인에게 수많은 질문을 했고, 모두 지혜롭게 대답해 주었지.
_____그의 답은 대부분 옳았다고 하네. 그런데 어느 날, 그 선인에게 한 선비가 찾아와서 물었어."
_____"재귀함수가 뭔가요?"
_____"재귀함수는 자기 자신을 호출하는 함수라네"
_____라고 답변하였지.
_____라고 답변하였지.
____라고 답변하였지.
라고 답변하였지.

5 B. 재귀함수가 뭔가요?

- ✓ 회색 글씨로 표시된 첫 번째 줄을 제외하면, 같은 색깔의 글씨끼리는 규칙이 보이시나요?
  항상 일정한 문장에서 시작해 일정한 문장으로 끝나고 있습니다.
- ✓ 그리고 두 일정한 문장 사이에는 바로 다음 단계(다른 색)에 해당하는, 역시 똑같은 규칙의 문장이 출력되고 있습니다.
- ✓ 가장 높은 단계로 들어갔을 때에는, 더 이상 더 높은 단계로 들어가지 않고 아래의 문장을 출력하는 것으로 끝이 납니다.

```
_____"재귀함수가 뭔가요?"
_____"재귀함수는 자기 자신을 호출하는 함수라네"
_____라고 답변하였지.
```

B. 재귀함수가 뭔가요?

- ✓ 규칙에 따라 재귀함수를 아래의 의사 코드를 사용하여 나타내는 것으로 설명을 마무리하겠습니다.
- ✓ 재귀함수에 단계라는 변수를 별도로 둔 이유는, 가장 높은 단계에 도달했을 경우 노란색 텍스트를 출력하고 재귀함수를 빠져나와야 하는 것도 있고, 단계에 따라 출력해야 하는 "_" 의 개수가 다르기 때문입니다.

어느 한 컴퓨터공학과 학생이 유명한 교수님을 찾아가 물었다.
"재귀함수가 뭔가요?"
"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을 통달한 선인이 있었어.
마을 사람들은 모두 그 선인에게 수많은 질문을 했고, 모두 지혜롭게 대답해 주었지.
그의 답은 대부분 옳았다고 하네. 그런데 어느 날, 그 선인에게 한 선비가 찾아와서 물었어."
____"재귀함수가 뭔가요?"
____"잘 들어보게. 옛날옛날 한 산 꼭대기에 이세상 모든 지식을 통달한 선인이 있었어.
____마을 사람들은 모두 그 선인에게 수많은 질문을 했고, 모두 지혜롭게 대답해 주었지.
____그의 답은 대부분 옳았다고 하네. 그런데 어느 날, 그 선인에게 한 선비가 찾아와서 물었어."
________"재귀함수가 뭔가요?"
________"재귀함수는 자기 자신을 호출하는 함수라네"
________라고 답변하였지.
____라고 답변하였지.
라고 답변하였지.

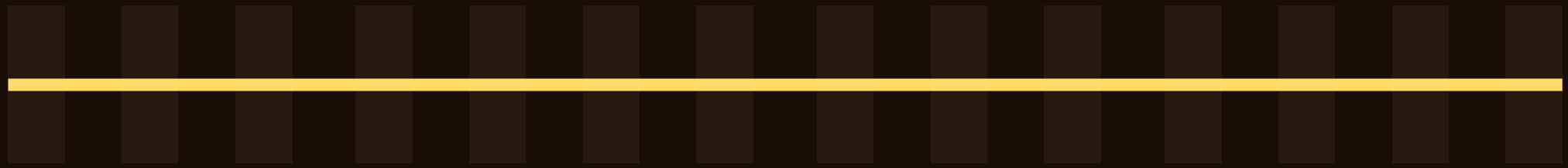
```
재귀함수(단계: X) {
    출력
    재귀함수(단계: X + 1)
    출력
}
```

3 C. 칸토어 집합

- ✓ C번은 분할 정복을 이용하여 해결할 수 있습니다.
- ✓ **당장 해결할 수는 없는 큰 문제**, 그리고 **큰 문제를 나누기 위한 규칙**을 생각해 봅시다.
- ✓ **당장 해결할 수는 없는 큰 문제** 는 출력해야 하는 문자열 전체가 될 것입니다.
- ✓ **큰 문제를 나누기 위한 규칙**은 바로 단계가 진행될 때마다 각 선을 3등분한 후, 가운데 선을 비운다는 점에서 찾아볼 수 있습니다.
- ✓ 이를 이용하여 문제를 해결해 봅시다.

3 C. 칸토어 집합

- ✓ 입력이 3이 주어졌다고 생각해 봅시다. 이 경우 출력해야 할 문자열의 길이는 27이 되겠군요. 우선은, “-” 로만 구성된 문자열을 저장해 둡시다.



3 C. 칸토어 집합

- ✓ 이제 문제를 풀기 위한 재귀함수를 구상해야 합니다. 문제에서는 각 단계가 진행될 때마다 선을 3등분하고 그 중 가운데 선을 없앤 뒤, **새롭게 생긴 두 선에 대해 다시 같은 과정을 반복**한다고 설명하고 있습니다.
- ✓ 즉, 기존의 선과 자른 후 생성된 새로운 선에 대해 실행해야 하는 연산이 같습니다. 그러므로, 재귀함수는 아래와 같이 설계하겠습니다.

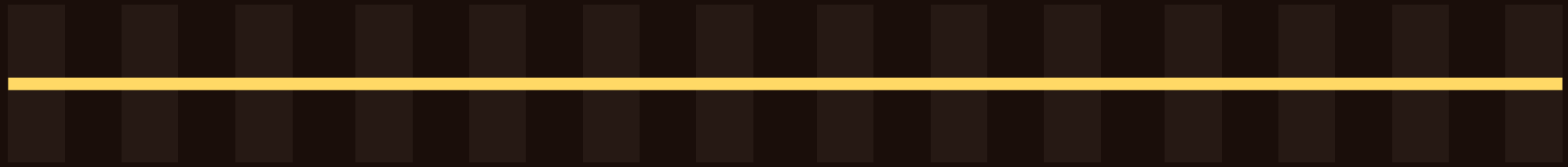
```
재귀함수(자르고자 하는 선의 범위) {
    선의 범위가 한 칸이면 더 못자르니 더 이상 안자르고 리턴

    재귀함수(선을 3등분한 후 첫 번째 선에 대한 범위)
    지우기(두 번째 선에 대한 범위)
    재귀함수(선을 3등분한 후 세 번째 선에 대한 범위)
}
```

3 C. 칸토어 집합

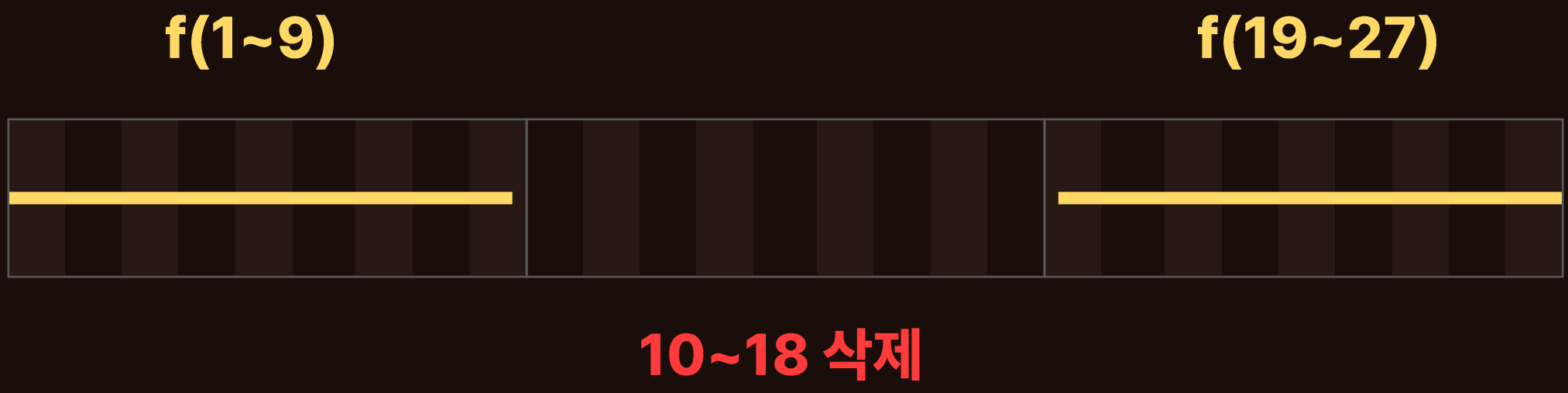
- ✓ 이제 시뮬레이션을 해 봅시다. 편의상 함수 이름은 f로 하겠습니다.
- ✓ 가장 첫 번째로 실행된 재귀함수의 선의 경우 전체 범위가 해당됩니다.

f(1~27)



3 C. 칸토어 집합

- ✓ 이 경우 재귀함수는 이렇게 실행됩니다.
- ✓ 첫 번째 범위와 세 번째 범위의 경우 다시 해당 선에 대해 똑같은 규칙을 적용하기 위해 범위만 바뀐채로 재귀 함수가 호출됩니다.
- ✓ 두 번째 범위의 경우 해당하는 영역의 선이 지워집니다.

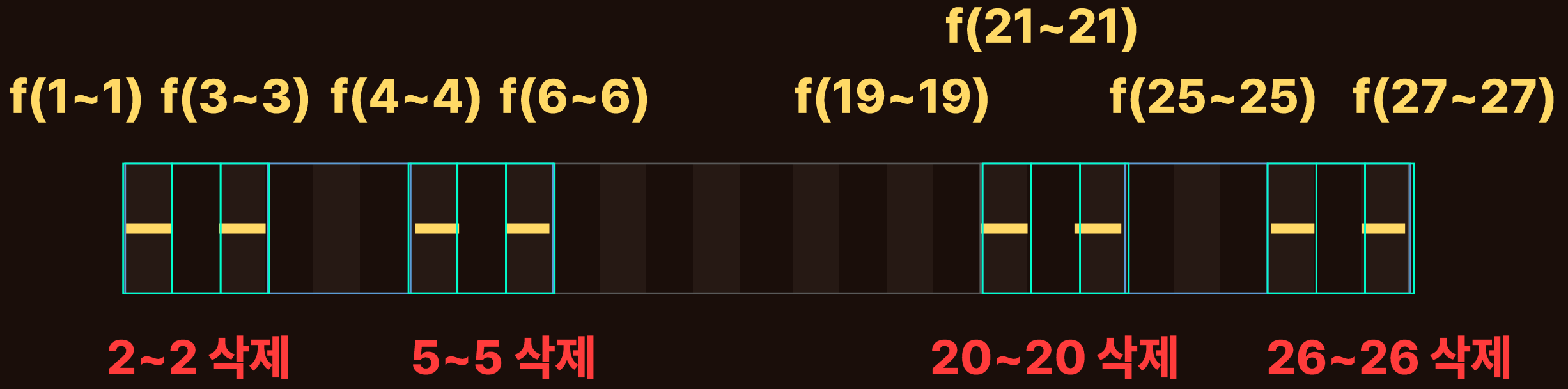


3 C. 칸토어 집합

✓ 다시 실행된 재귀함수에도 똑같은 행동을 계속해서 반복하시면 됩니다.

3 C. 칸토어 집합

- ✓ 한 번만 더 반복해 줍시다.
- ✓ 이 과정을 거치고 나면, 모든 재귀함수의 범위는 이제 1이 됩니다. 길이가 1인 선은 더 이상 쪼갤 수 없으므로, 재귀함수에 범위가 1인 입력이 들어오면 아무것도 하지 않고 리턴해주시면 되겠습니다(재귀의 종료조건).



3 C. 칸토어 집합

- ✓ 모든 과정을 마친 문자열을 출력하시면 정답이 됩니다.
- ✓ 각 단계마다 재귀함수가 하는 연산 중 시간이 드는 연산은 역시 가운데의 칸을 지우는 작업입니다. 양쪽의 선에서 재귀함수를 호출하는 작업은 호출 자체만을 하는 작업은 $O(1)$로 생각할게요.
- ✓ 결국 모든 연산이 진행된 후 지워진 칸을 확인하는 것으로 모든 재귀함수를 통틀어 연산이 몇 회 실행되었는지를 짐작해 볼 수 있는데, 초기 문자열의 길이는 아무리 길어봐야 3의 12제곱 = $531\ 441$이며, 지워지는 칸의 개수는 이 수보다 작습니다. 또한, 초기 문자열의 길이가 길어질 수록 지워지는 칸의 개수 또한 선형적으로 늘어납니다.
- ✓ 따라서 시간복잡도는 $O(3^N)$ 이며, 최악의 경우 $N = 12$이므로 문제를 충분히 해결할 수 있습니다.
- ✓ 시간복잡도에 대한 설명이 어렵죠 ☹ 지워지는 칸의 개수가 몇 칸인지만 생각하고 넘어가셔도 충분하다고 생각합니다. 결국 칸을 지우는 것이 핵심 연산이 되니까요.

3 C. 칸토어 집합

- ✓ 이 문제는 테스트 케이스의 수가 주어지지 않습니다. 따라서 Node.js로 문제를 푸시는 분이 아니라면 문제 해결이 난감할 것입니다.
- ✓ 대부분의 언어는 입력 파일의 끝을 검사하는 기능이 있습니다. **EOF 처리 방법**에 대해 검색해 보시면 해결 방법을 찾으실 수 있을 겁니다.

2 D. 종이의 개수

- 칸토어 집합과 비슷합니다만 이번에는 2차원 형태의 범위를 다뤄 주셔야 합니다.
- **각 재귀함수에서 해야 하는 일은 특정 범위에서의 종이의 개수를 반환하는 것입니다. 가장 큰 범위의 종이의 개수는 더 작은 범위에서 호출한 재귀함수가 반환한 종이의 개수를 모두 합치는 식으로 구해나가시면 됩니다.**
- **범위를 정하고 재귀 함수를 실행시켰을 때, 먼저 범위에 해당하는 모든 영역이 같은 색인지를 검사합니다. 모두 같은 색이라면 종이의 개수는 1이므로 1을 반환하면 되며, 그렇지 않을 경우 다시 네 조각으로 종이를 나누어 나눠진 범위에 대해 재귀를 돌려주시면 됩니다. 그리고 반환된 값을 모두 합친 값을 반환하면 되겠죠.**

2 D. 종이의 개수

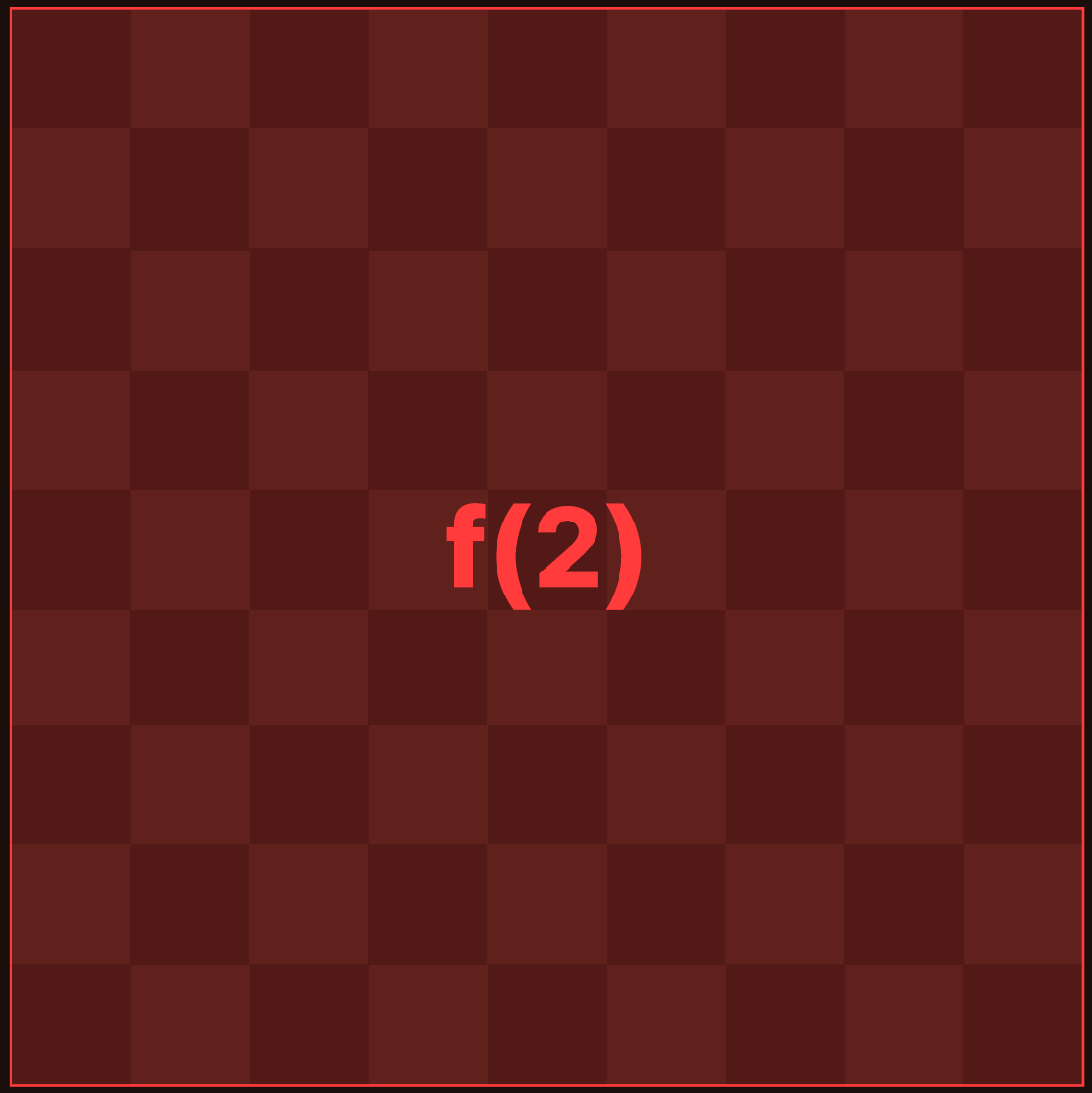
- ✓ 각 단계에서 최악의 경우 종이 전체 크기만큼의 종이를 탐색하게 되며, 재귀함수가 한 단계씩 깊게 진행될 때마다 검사하는 종이의 가로 길이는 절반으로 줄어듭니다.
- ✓ 따라서 단계 수는 총 $\log N$ 개가 되며, 이에 따라 시간복잡도는 $O(N^2 \log N)$ 가 됩니다.
- ✓ 문제해결에서 $\log N$ 의 경우 보통 한 자리 수~두 자리 수가 나오는 경우가 많습니다. 그렇기에 시간복잡도에 $\log N$ 가 붙은 경우에는 단지 연산의 횟수에 몇 배를 해 준것에 불과하므로, 시간 제한이 정말 빡빡한 상황이 아니라면 별로 풀이에 영향을 주지 않는다~ 라고 생각하셔도 무방하겠습니다.

E. 별 찍기 - 10

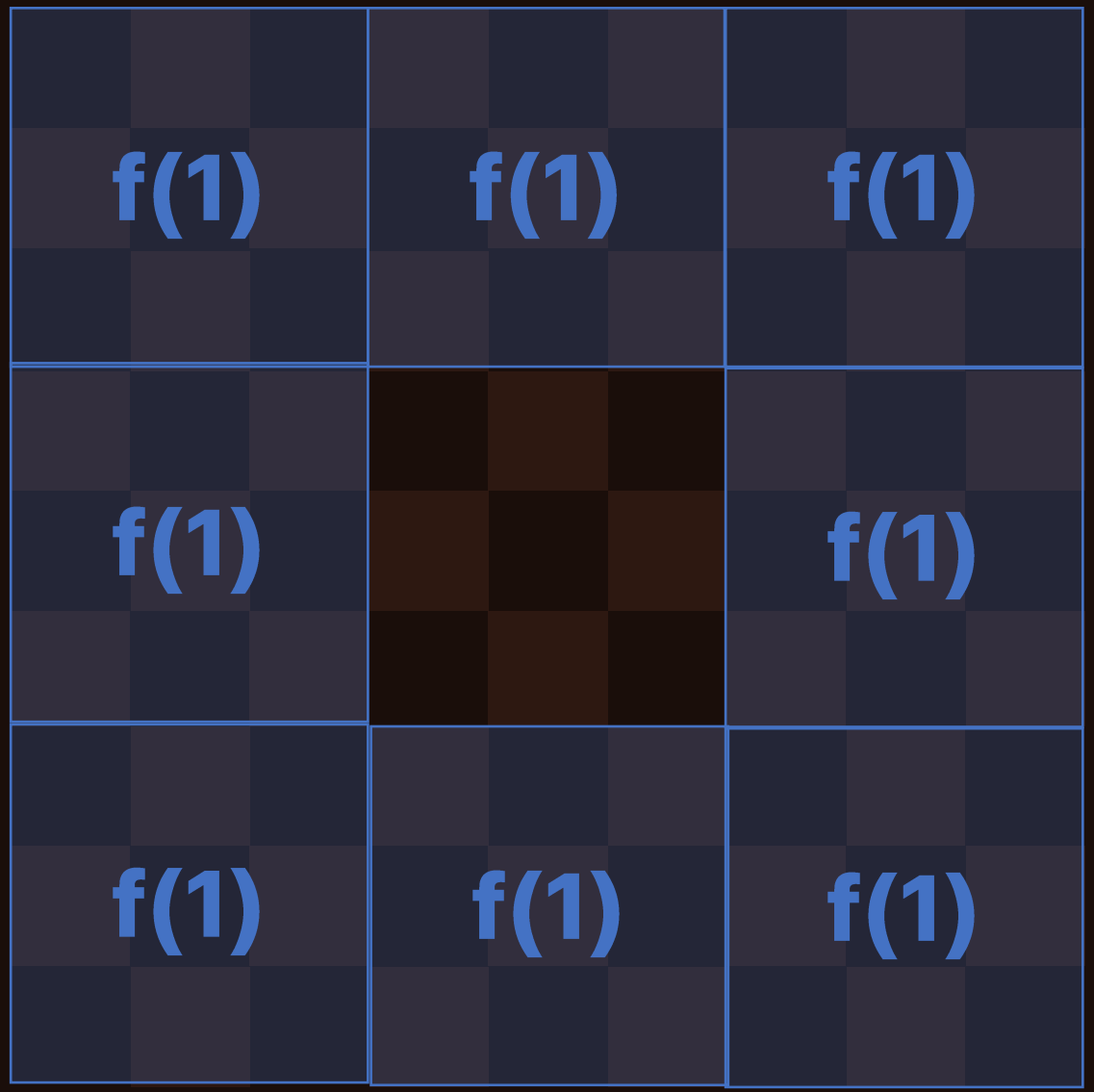
- ✓ 예제 출력에 나온 그림을 한 번에 출력하기는 힘듭니다. 그 대신, 문제를 작은 범위로 쪼개 해결한다면 문제를 해결하실 수 있을 것입니다.
- ✓ 규칙을 열심히 관찰해 보시면 좋은 연습이 될 것이라고 생각합니다! 다음 페이지에서는 **9×9** 크기의 그림을 출력하는 시뮬레이션을 해 보겠습니다. 규칙 유추와 재귀함수 구상에 도움이 되셨으면 좋겠습니다.

E. 별 찍기 - 10

E. 별 찍기 - 10

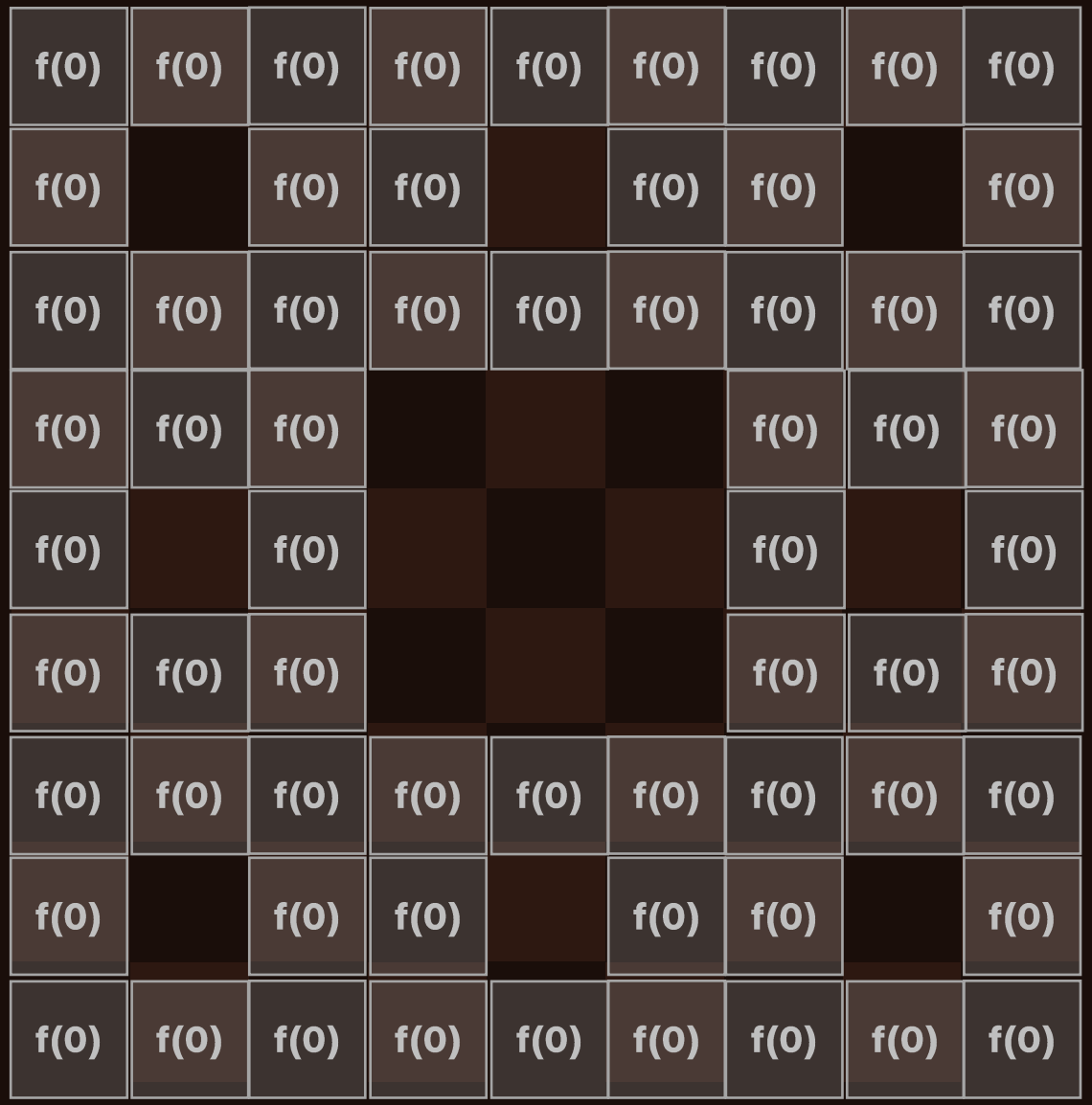


E. 별 찍기 - 10

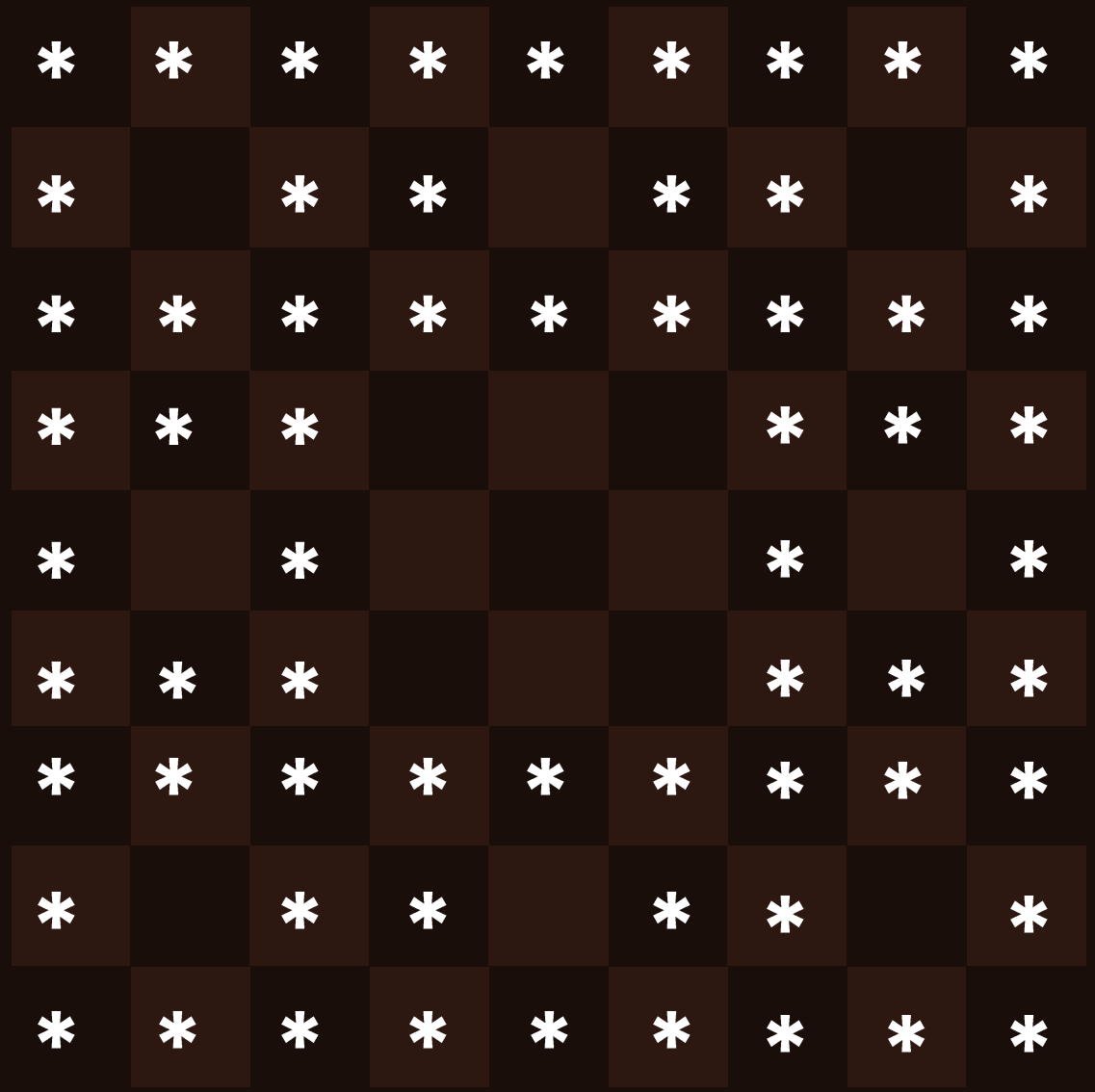


E. 별 찍기 - 10

f(0) = *


f(0) f(0) f(0) f(0) f(0) f(0) f(0) f(0) f(0)
f(0) f(0) f(0) f(0) f(0) f(0) f(0)
f(0) f(0) f(0) f(0) f(0) f(0) f(0) f(0) f(0)
f(0) f(0) f(0) f(0) f(0) f(0) f(0)
f(0) f(0) f(0) f(0) f(0) f(0) f(0)
f(0) f(0) f(0) f(0) f(0) f(0) f(0) f(0) f(0)
f(0) f(0) f(0) f(0) f(0) f(0) f(0) f(0) f(0)
f(0) f(0) f(0) f(0) f(0) f(0) f(0)
f(0) f(0) f(0) f(0) f(0) f(0) f(0) f(0) f(0)

E. 별 찍기 - 10



The End