알고리즘

01

알고리즘이란?

02

재귀함수

03

정렬 종류

04

동적 프로그래밍

알고리즘이란?

- 알고리즘
  - 자료구조를 이용하여 어떠한 문제를 해결하는 구체적인 방식을 말합니다

재귀 함수 공부 이유

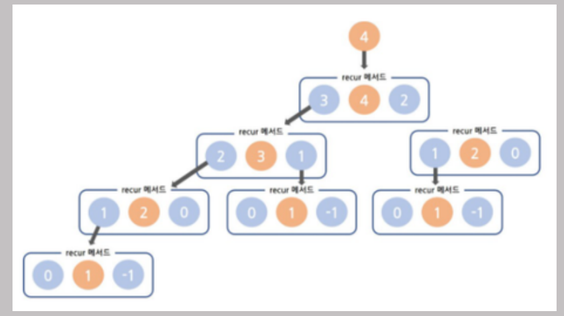
1. 변수 사용을 줄일 수 있다.
변수 사용이 줄어든다는 뜻은 프로그램에 오류가 생길 가능성이 줄어들고, 프로그램이 정상적으로 돌아가는지에 대한 증명이 쉬워진다는 뜻이다.

2. 알고리즘 자체가 재귀적인 표현이 자연스러운 경우 (+ 가독성)
재귀함수하면 너무나 유명한 예시인 피보나치 수열은 1, 1, 2, 3, 5, 8 ... 의 값을 가지는 데 이를 식으로 나타내보자면 다음과 같다.

하향식,상향식 접근법

하향식 분석 - top - down

출력 형태를 만들어 놓고 회수하는 형태
결과가 이전의 결과에 영향을 받는 것 -> 앞에서 보았던 재귀 메서드의 형태



상향식 분석 - bottom - up

가장 아래쪽부터 위로 쌓아 올리면서 분석하는 방법
하향식 분석과 분석 방식이 정반대

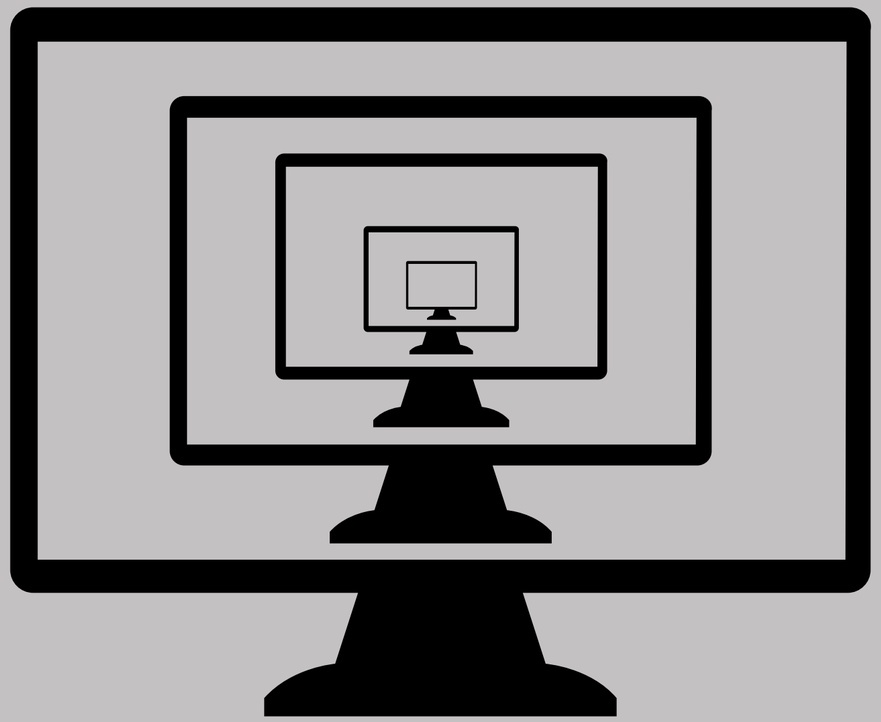
- f(n + 1) = f(n) + a

https://mintparc.tistory.com/121

재귀 사용 예시

사용 예시

- JSON(parser)등 또한 오브젝트(object)를 순회할 때 재귀를 사용
- 웹에서 DOM등을 순회할때도 사용. DOM Api query Selector등도 있다.
- 내부적으로 메서드로 사용하기도 한다.
- 알고리즘,코딩 테스트 등에도 사용함



재귀함수

- **정의**
  - 함수안에서 자기 자신을 참조하는것을 뜻한다.

- **재귀 장단점**
  - **장점**
    - 이해하기 쉽다.
    - 프로그래밍을 쉽게 할 수 있다.
    - 변수 사용을 줄여 준다.
  - **단점**
    - 반복문보다 메모리 사용량 많고 수행 시간이 더 길어질 수 있다
    - 무한반복이 일어나는 경우 에러가 발생한다.

기저조건(탈출 조건)

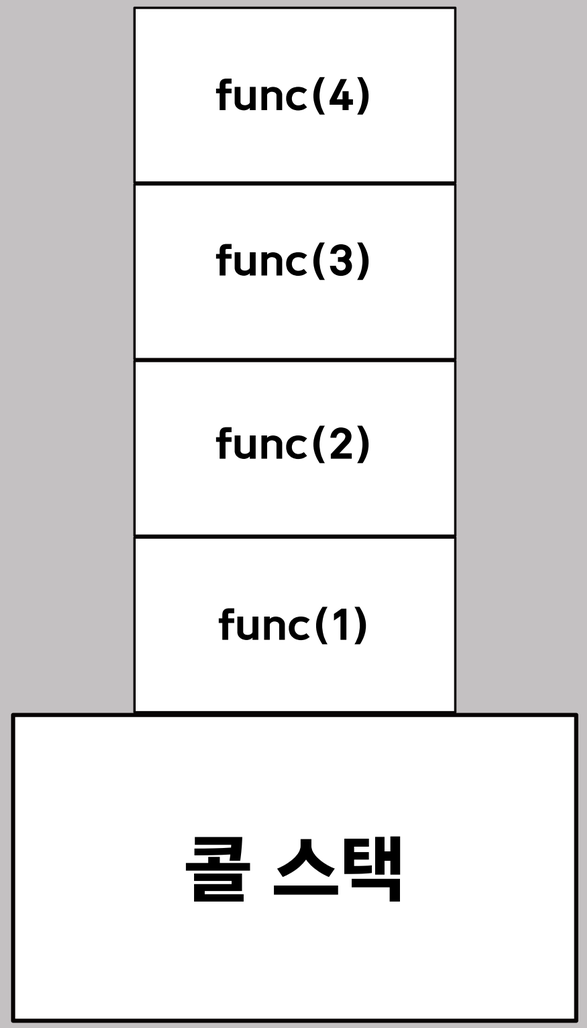
- **기저조건의 정의**
  - 재귀함수에서 **탈출하기 위한 조건**

- **기저조건의 특징**
  - 기져조건을 명시를 안할시 콜스택에 메모리가 넘쳐서 강제종료 된다.
  - 기저조건을 명시해야 탈출이 가능하다

```javascript
function recur(num) {
    // 기저조건
    if (num === 0) return num;
    return num + recur(num - 1);
}

const sum = recur(5);
console.log(sum)
```

콜스택



- 콜스택
  - 함수가 호출될때 그 함수들을 모아두는 자료구조

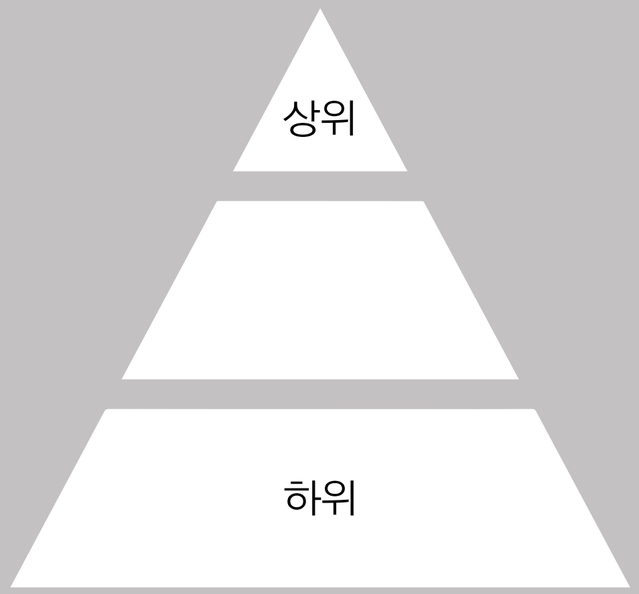
```
function fun1() {
  function fun2() {
    function fun3() {
      function fun4() {}
    }
  }
}
```

```
console.debug("example(2) 재귀함수의 기저조건");
// 10보다 크면 종료라는 기저조건을 명시함으로서 종료된다.
function func(number) {
  if (number > 10) return;
  console.log(number);
  func(number + 1);
}
func(1);
```

```
1
2
3
4
5
6
7
8
9
10
```

재귀 함수 예시

재귀 함수 vs 반복문



- **공통점**
  - 반복해서 값을 추출한다.

- **차이점**

함수	접근 방식	우위	예시
재귀 함수	상향식/하향식	하향식 접근	팩토리얼 구현
반복문	상향식	상향식 접근	일반적인 상황

- 상향식/하향식 이란?
  - 상향식은 상위문제를 해결하는데 하위 문제를 사용하는 것
  - 하향식은 하위문제를 해결하는데 상위 문제를 사용하는 것

재귀 함수 한계점

- 호출될 때마다 메모리에 스택이 쌓이는데, 한계치 이상으로 호출이 되어 스택이 넘쳐버리면 메모리 부족으로 에러가 발생 합니다.

- 반복문에 비해 시간이 오래 걸립니다.

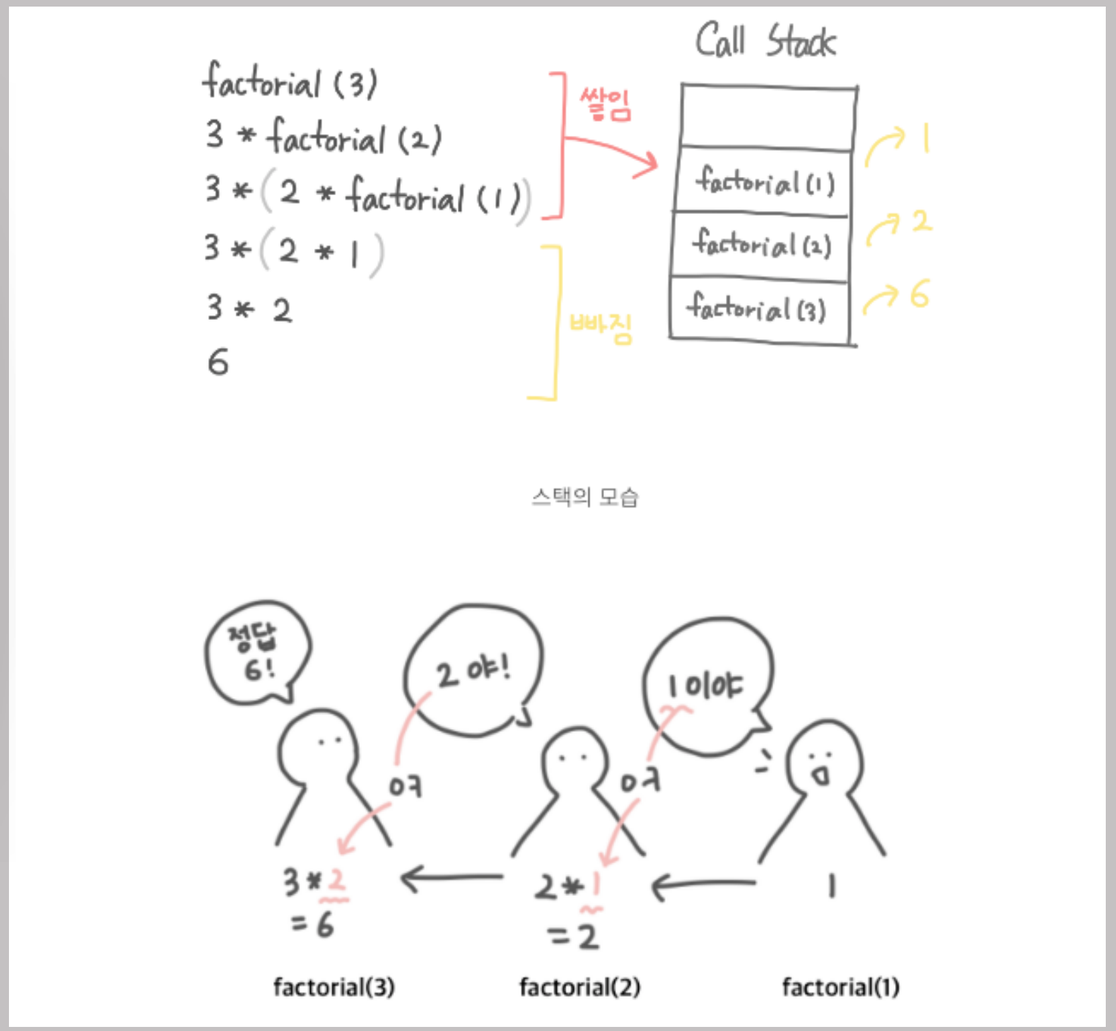
- 함수 호출과 복귀를 하기 위한 context switching 비용이 발생하기 때문에, 속도가 상대적으로 느립니다. 즉, 오버헤드가 발생하여 속도가 느리게 됩니다.

꼬리 재귀란?

재귀 호출이 끝난 후 현재 함수에서 추가 연산을 요구하지 않도록 구현하는 재귀의 형태입니다. 이를 이용하면 함수 호출이 반복되어 스택이 깊어지는 문제를 **컴파일러가 선형으로 처리하도록 알고리즘을 바꿔 스택을 재사용**할 수 있게 됩니다.

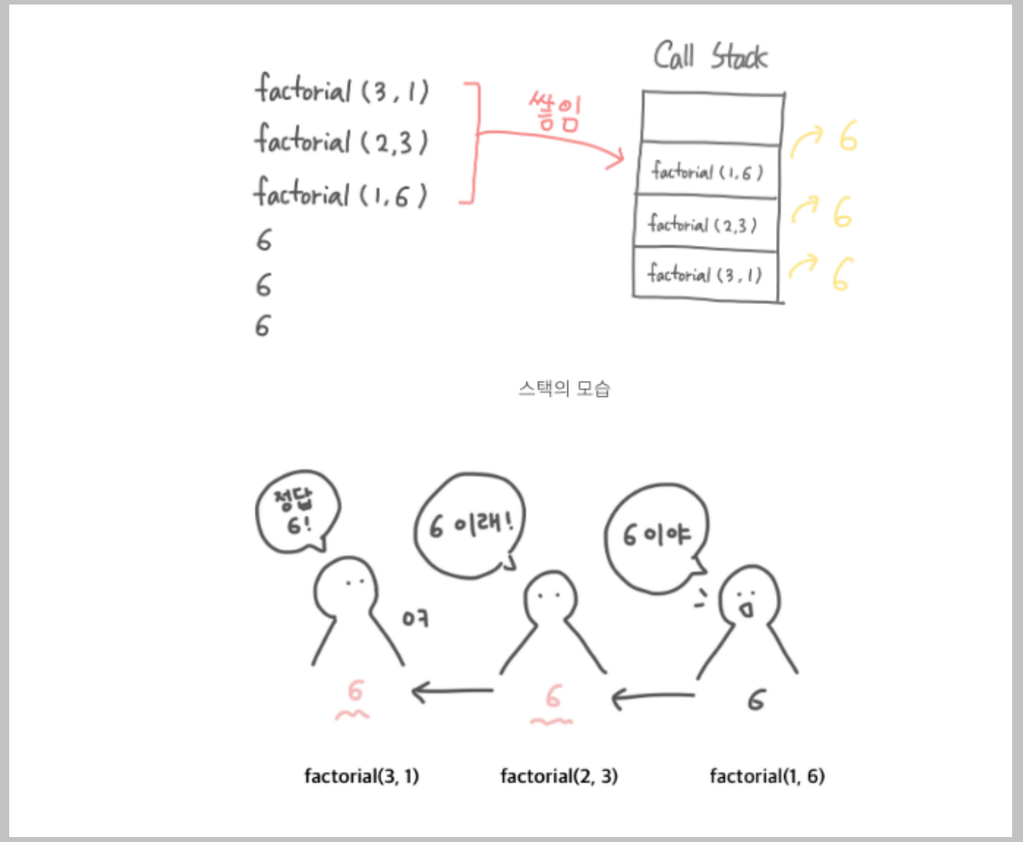
재귀 팩토리얼

```
function factorial(n) {
    if (n === 1) {
        return 1;
    }
    return n * factorial(n-1);
}
```



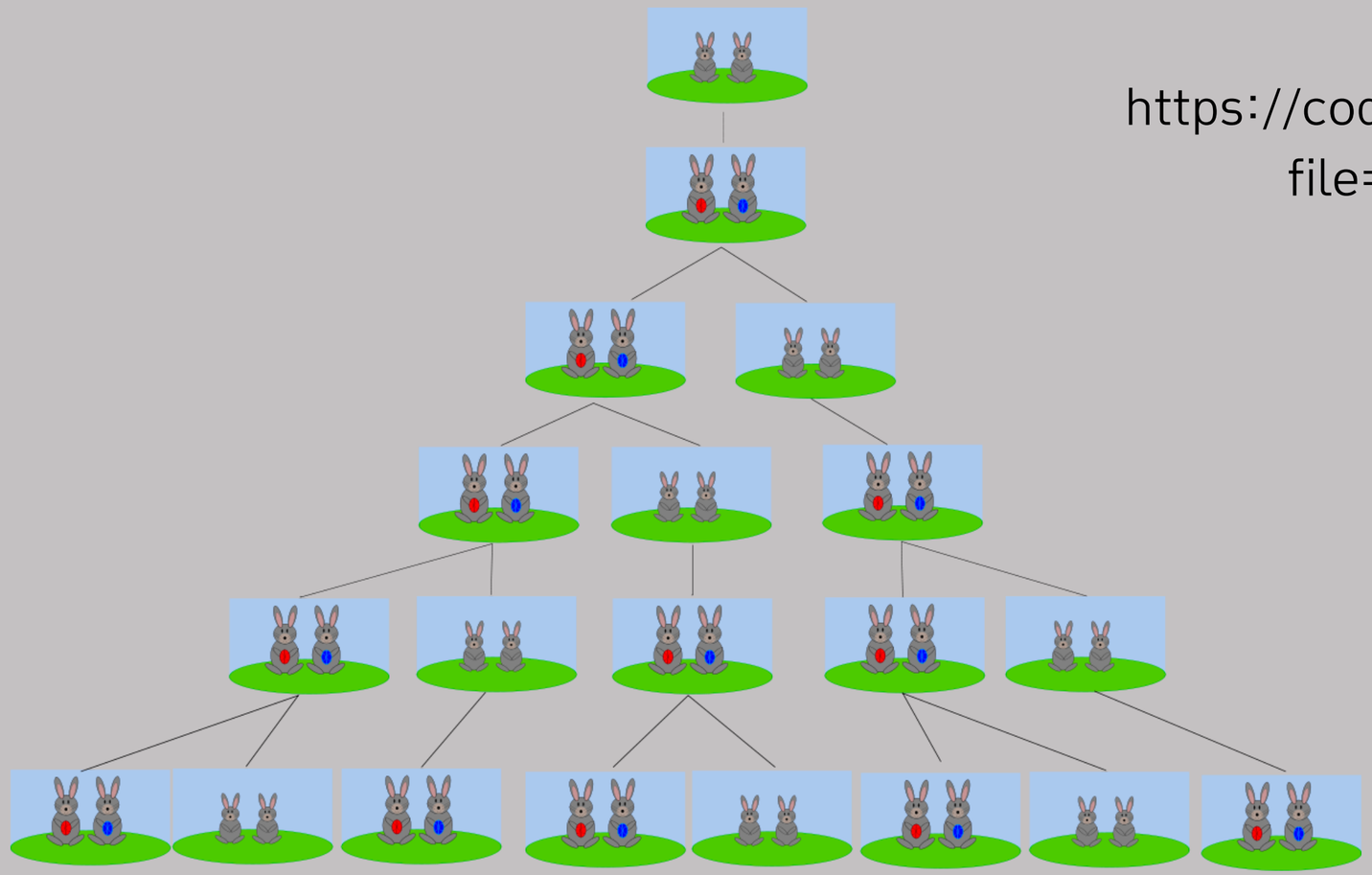
꼬리 재귀 팩토리얼

```
function factorial(n, total = 1){
    if(n === 1){
        return total;
    }
    return factorial(n - 1, n * total);
}
```



https://joooing.tistory.com/entry/%EC%9E%AC%EA%B7%80-%E2%86%92-%EA%BC%AC%EB%A6%AC-%EC%9E%AC%EA%B7%80-Tail-Recursion

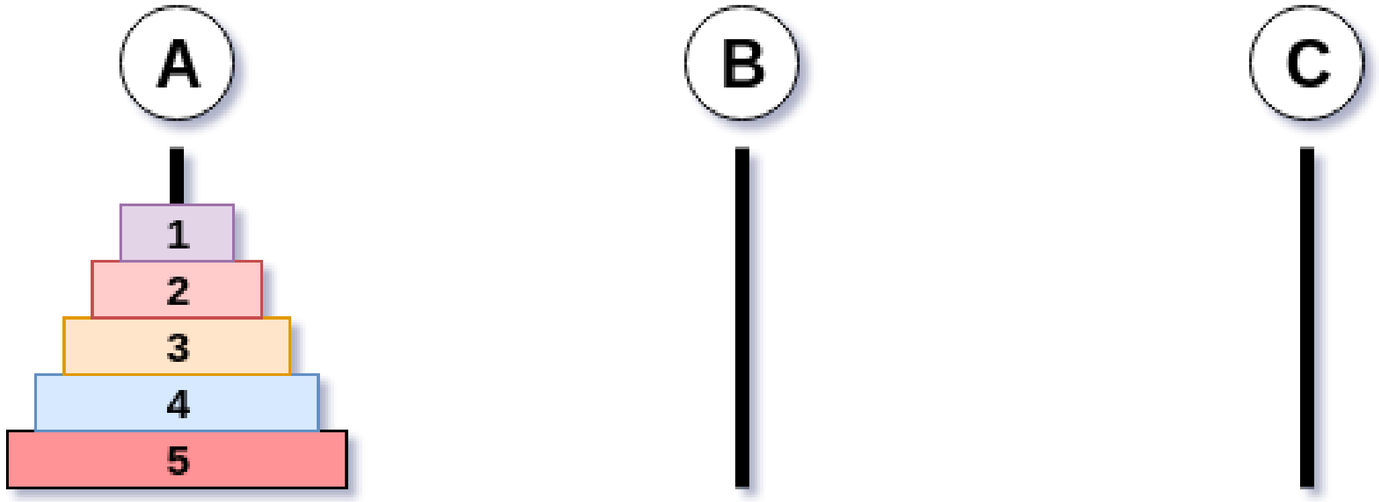
피보나치 수열



실습코드

https://codesandbox.io/s/algorithm-g4bmiu?file=/src/Recursive/fibonacci.js

하노이 탑



https://shoark7.github.io/programming/algorithm/tower-of-hanoi

실습코드

https://codesandbox.io/s/algorithm-g4bmiu?file=/src/Recursive/hanoi.js

재귀 활용 네비게이션