1. **분할정복 알고리즘** (divide and conquer Algorithm)

문제를 **나눌 수 없을 때까지 나누어서 각각 풀면서 다시 병합**하여

문제의 답을 얻는 [알고리즘](https://namu.wiki/w/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98) 설계 기법

2. 특징

가. **분할, 정복, 조합**

* 분할: 문제를 더이상 분할할 수 없을 때까지 동일한 유형의 여러 하위 문제로 분할
* 정복: 가장 작은 단위의 하위 문제를 해결하여 정복
* 조합: 하위 문제에 대한 결과를 원래 문제에 대한 결과로 조합

나. 장점 : 특징상 병렬적으로 문제를 해결하는데 큰 장점

다. 단점

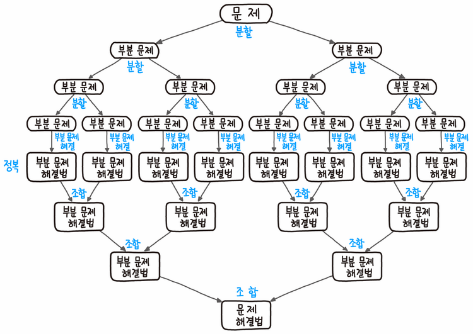
* 함수를 재귀적으로 호출 → 오버헤드 발생
* 스택에 다양한 데이터 보관 → 스택 오버플로우 발생, 과도한 메모리 (메모이제이션으로 해결)

3. 동작에 Fit한 경우

가. **최적 부분 구조** **(optimal substructure)**

1) **매 순간의 최적해가 문제에 대한 최적해여야 한다.**

나. **각 문제들이 독립적**

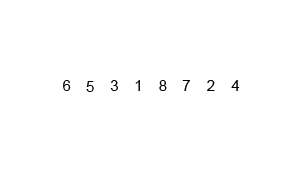


4. 어디에 쓸까?

가. 퀵소트



나. 머지소트



5. 그리디 vs 동적 계획법(DP) vs 분할정복

가. 분할 정복 : 직접 해결할 수 있을 정도로 간단한 문제가 될 때까지 문제를 재귀적으로 나누고,

그 하위 문제의 결과를 조합하여 원래 문제의 결과로 만들어 낸다.

**그리디**

탐욕 선택 속성

최적 부분 구조

다익스트라

허프만 코딩 알고리즘

ID3 알고리즘 (머신러닝)

**동적 계획법**

점화식 (중복된 하위 문제)

최적 부분 구조

DP 테이블, 캐싱 기법

배낭 문제

동전 바꾸기 문제

가장 큰 합

피보나치

다익스트라

**분할 정복**

최적 부분 구조

동일한 부분 문제 반복 계산X

병합정렬, 퀵정렬