Divide and Conquer(분할 정복)

그대로 해결할 수 없는 문제를 작은 문제로 분할하여 문제를 해결하는 방법
Quick Sort 나 Merge Sort 로 대표되는 정렬 알고리즘 문제가 대표적이다.

분할 정복 알고리즘은 보통 재귀 함수(recursive function)를 통해 자연스럽게 구현된다. 예를 들면,

```
function F(x):
  if F(x)의 문제가 간단 then:
    return F(x)을 직접 계산한 값
  else:
    x 를 y1, y2 로 분할
    F(y1)과 F(y2)를 호출
    return F(y1), F(y2)로부터 F(x)를 구한 값
```

전략

1. 문제 사례를 하나 이상의 작은 사례로 분할(Divide)한다.
2. 작은 사례들을 각각 정복(Conquer)한다. 작은 사례가 충분히 작지 않은 이상 재귀를 사용한다.
3. 필요하다면, 작은 사례에 대한 해답을 통한(Combine)하여 원래 사례의 해답을 구한다.

장점

: 문제를 나누어 해결한다는 문제를 해결하는 데 큰 강점이 있다.

단점

: 함수를 재귀적으로 호출한다는 점에서 함수 호출로 인한 오버헤드가 발생하며, 스택에 다양한 데이터를 보관하고 있어야 하므로 스택 오버플로우가 발생하거나 과도한 메모리 사용을 하게 되는 단점 (결과값을 얻기 위한 시간, 메모리 등 자원 사용이 지나치게 많이 소요되는 문제)

1 부터 n 까지의 합 (1 + 2 + 3 + ... +n)

```
int sum(start, end){
        if(start == end) return start;

    mid = (start + end) / 2;

    return consecutive_sum(start, mid) + consecutive_sum(mid + 1, end);
}
```

출력

```
System.out.println(consecutive_sum(1, 100))
```

결과 값

5050

1) Divide: 1 ~ n 까지의 합을 절반 씩 나눈다.
2) Conquer: 절반씩 나눈 것들의 합을 구한다.
3) Combine: Conquer 에서 구한 값들을 모두 합친다.

이분 탐색

1) 정렬된 배열 안에서 특정 원소를 찾을 때 인덱스 i 부터 j 의 중간값과 비교

2) 중간값이 찾는 원소가 아니라면 인덱스 i 와 j 다시 정해줌

3) 인덱스 i 와 j 의 정할 때 마다 탐색 범위 반으로 줄어듦

1)처음 범위는 인덱스 0 부터 끝까지이다. 이 때 중간 인덱스를 mid 로 한다.

2) mid 의 값와 찾는 원소를 비교한다.
2-1)찾는 원소와 mid 의 값이 같다면 탐색 종료한다.
2-2) mid 의 값 < 찾는 원소 일 때 left 는 mid + 1 로 하여 2)를 다시 반복한다.
2-3) mid 의 값 > 찾는 원소 일 때 right 는 mid - 1 로 하여 2)를 다시 반복한다.

2-4) 만약 right > left 가 된다면 해당 배열에 찾는 원소가 없는 것이다.

시간복잡도

순차적 탐색: 최악의 경우 배열의 끝까지 탐색해야한다. -> O(n)

이분 탐색: 범위를 새로 정할 때 마다 탐색 범위는 1/2 씩 줄어든다. -> O(log n)

탐색 기법	시간복잡도
순차적 탐색	O(n)
이분 탐색	O(log n)