

# 분할정복

## 분할정복 (Divide and Conquer) 알고리즘

### 📚 개념과 이해

분할정복(Divide and Conquer)은 문제를 더 작은 부분으로 나누어 각각 해결한 후, 그 결과를 조합하여 원래 문제의 해를 도출하는 알고리즘 설계 기법입니다. 이 방법은 복잡한 문제를 간단한 문제로 변환하여 효율적으로 해결할 수 있도록 도와줍니다.

### 🔍 핵심 원리

분할정복은 다음과 같은 세 단계로 구성됩니다:

1. **분할(Divide)**: 문제를 더 작은 하위 문제들로 나눈다.
2. **정복(Conquer)**: 각 하위 문제를 재귀적으로 해결한다.
3. **결합(Combine)**: 하위 문제의 결과를 조합하여 원래 문제의 해를 만든다.

이 과정은 재귀적 구조를 가지며, 종료 조건(기저 조건)이 있어야 합니다.

### ✓ 예시: 병합정렬 (Merge Sort)

병합정렬은 분할정복의 대표적인 예입니다.

- 배열을 두 개의 절반으로 나눈다.
- 각 절반을 재귀적으로 정렬한다.
- 정렬된 두 배열을 병합하여 최종 정렬된 배열을 만든다.

```
#include <bits/stdc++.h>
using namespace std;

void merge(vector<int>& arr, int left, int mid, int right) {
    vector<int> leftArr, rightArr;

    // 왼쪽 부분 복사
    for (int i = left; i <= mid; i++) {
        leftArr.push_back(arr[i]);
    }

    // 오른쪽 부분 복사
    for (int i = mid + 1; i <= right; i++) {
```

```

        rightArr.push_back(arr[i]);
    }

    // 병합 (두 배열을 정렬된 순서로 합침)
    int i = 0, j = 0, k = left;
    while (i < leftArr.size() && j < rightArr.size()) {
        if (leftArr[i] <= rightArr[j]) {
            arr[k++] = leftArr[i++];
        } else {
            arr[k++] = rightArr[j++];
        }
    }

    // 남은 원소 복사
    while (i < leftArr.size()) arr[k++] = leftArr[i++];
    while (j < rightArr.size()) arr[k++] = rightArr[j++];
}

void mergeSort(vector<int>& arr, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        // 왼쪽 부분 정렬
        mergeSort(arr, left, mid);
        // 오른쪽 부분 정렬
        mergeSort(arr, mid + 1, right);

        // 두 부분 병합
        merge(arr, left, mid, right);
    }
}

// 테스트 코드
int main() {
    vector<int> arr = {5, 2, 8, 1, 9, 3};
    int n = arr.size();

    mergeSort(arr, 0, n - 1);

    // 출력
    for (int x : arr) {
        cout << x << " ";
    }
    cout << endl;

    return 0;
}

```

 출력: 1 2 3 5 8 9

## 분할정복의 장점

장점	설명
효율성	$O(n \log n)$ 복잡도로 큰 데이터도 빠르게 정렬 가능
안정성	동일한 값이 있을 경우 원래 순서를 유지 (안정 정렬)
병렬 처리 가능	하위 문제는 별도로 처리 가능하여 멀티코어 활용 가능

## 분할정복의 한계

한계	설명
메모리 사용	병합 과정에서 추가 배열을 사용하므로 $O(n)$ 공간 필요
작은 데이터에 비효율	작은 배열에서는 더 간단한 정렬(예: 삽입정렬)이 더 빠를 수 있음
재귀 깊이 제한	매우 큰 데이터에서는 스택 오버플로우 발생 가능

## 다른 분할정복 예제

알고리즘	설명
빠른 정렬 (Quick Sort)	피봇 기반으로 분할, 재귀적으로 정렬. 평균 $O(n \log n)$ , 최악 $O(n^2)$
피보나치 수 계산	재귀적으로 피보나치 수를 계산 (하지만 효율성 문제 있음)
최대 공약수 (GCD)	Euclidean 알고리즘: $a = b*q + r \rightarrow GCD(a,b) = GCD(b,r)$
최소 공배수 (LCM)	$LCM(a,b) = (a*b) / GCD(a,b)$

## 분할정복의 응용

분야	응용 예시
수학	소인수 분해, 최대공약수, 최소공배수
컴퓨터 과학	정렬, 탐색, 최적화 문제 해결
그래프 이론	최단 경로 문제 (예: 분할 정복 기반 최단 경로 탐색)
암호학	RSA 등에서 소수 분해 문제 활용

## 📌 핵심 요약 (A4 10장 기준)

- 분할정복은 문제를 작은 부분으로 나누고 해결하는 전략.
- 재귀적 구조를 가지며, 기저 조건이 반드시 필요.
- 병합정렬은 가장 대표적인 예로,  $O(n \log n)$  성능을 보임.
- 메모리 사용과 재귀 깊이는 주의할 점.
- 다른 분할정복 알고리즘: 빠른 정렬, GCD, LCM 등도 활용 가능.



## 실무에서의 활용 팁

1. 문제 분석: 문제를 "작은 부분으로 나눌 수 있는가?"를 먼저 생각.
2. 재귀 구조 설계: 함수가 자신을 다시 호출할 수 있도록 구현.
3. 기저 조건 설정: 재귀가 끝나는 조건을 명확히 정의.
4. 결합 과정 최적화: 하위 문제의 결과를 조합할 때 효율적으로 처리.
5. 테스트: 작은 입력부터 시작하여 정확성과 성능을 검증.



## 예시 문제 (문제 해결을 위한 분할정복 적용)

문제: 1~n 사이의 정수 중 3개를 고르는 조합을 모두 구하라.

(예:  $n=4 \rightarrow \{1,2,3\}, \{1,2,4\}, \{1,3,4\}, \{2,3,4\}$ )

이 문제는 백트래킹과 유사하지만, 분할정복으로 접근할 수 있습니다.

- 전체 수를 1~n으로 나누고, 3개를 선택하는 문제 → **분할** ( $1 \sim n/2$ ,  $n/2 + 1 \sim n$ )
- 각 부분에서 3개를 선택하거나, 전체에서 선택

하지만 이 경우 백트래킹이 더 자연스럽습니다.

분할정복은 결합 가능한 문제에 더 적합합니다.

## ✓ 결론

- 분할정복은 복잡한 문제를 간단히 해결할 수 있는 강력한 기법입니다.
- 특히 정렬, 수학 문제, 최적화 문제 등에 효과적입니다.
- 실무에서는 병합정렬, 빠른 정렬, **GCD/LCM** 등에 자주 활용됩니다.
- 재귀 구조를 잘 이해하고, 기저 조건과 결합 과정을 정확히 구현해야 합니다.

 **Tip:** 분할정복을 사용할 때는 항상 문제의 구조를 분석하고, "이 문제를 어떻게 나누면 더 쉽게 해결할 수 있을까?"를 질문하세요.

---

 이제 분할정복을 이해하고, 실무에서 어떻게 활용할 수 있는지에 대한 기초 지식을 갖출 수 있습니다. 다음 단계는 동적 계획법(DP)과 백트래킹과의 비교를 통해 더 깊이 이해할 수 있습니다.