1 분할 정복

1 분할 정복(Divide and Conquer)

- 주어진 문제의 입력을 다루기 쉽게 부분으로 **분할하여 문제를 해결(정복)하는 방식**의 알고리즘
- 문제를 더 이상 나눌 수 없을 때까지 나누고 이렇게 나누어진 문제들을 각각 풂으로써 전체 문제의 답을 얻는 알고리즘
- 문제를 두 단계인 ①분할과 ②정복으로 나눠서 해결하는 것

1 분할 정복(Divide and Conquer)

- 분할한 입력에 대하여 동일한 알고리즘을 적용해 해를 계산하며 이들의 해를 취합하여 원래 문제의 해를 얻음
- 엄청나게 크고 방대한 문제를 조금씩 나눠가면서 용이하게 풀 수 있는 문제 단위로 나눈 다음 그것들을 다시 합쳐서 해결하자는 개념

1 분할 정복(Divide and Conquer)

- 분할된 입력에 대한 문제를 **부분 문제**(Subproblem) 라고 하고, 부분 문제의 해를 **부분해**라고 함
- 부분 문제는 더 이상 분할할 수 없을 때까지 계속 분할함
- 점진적 개발에서 모듈 하나씩 개발하는 것도 분할 정복 개념임
- 재귀적 구조를 가진 알고리즘에서는 대개 분할 정복 접근법을 따름
- 문제를 쪼개는 요령이나 규칙은 없으며 개발자의 창의에 달려 있음

1 분할 정복(Divide and Conquer)

- 분할 정복은 동적 계획법과는 다른 방식임
- 분할 정복의 대표적인 예
  - 합병 정렬
  - 퀵 정렬
  - 최대값 찾기
  - 임계값의 결정
  - 이진 탐색
  - 거듭제곱 연산(a^b) 등

1 분할 정복

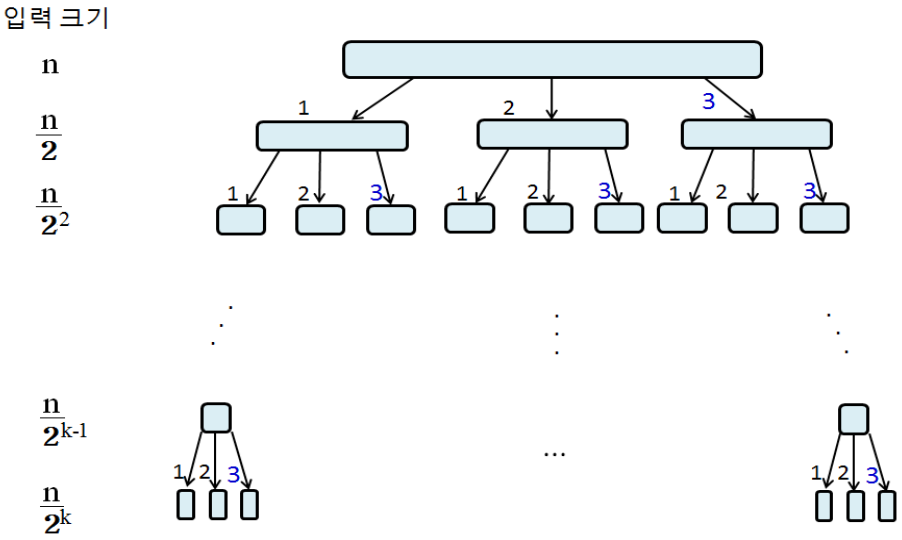
2 분할(Divide)

- 분할 단계에서는 주어진 문제를 여러 개의 부분 문제들로 나눔
- 문제가 작아지면 작아질수록 풀기 쉬워지는 성질을 이용함
- 문제의 크기가 엄청나게 줄어든다면(n=1 or n=2 정도) 바로 답을 구할 수 있는 수준이 됨
- 예)
  - n개의 원소를 정렬하는 문제를 2개의 n/2 원소를 정렬하는 문제로 분할하는 것

1 분할 정복

2 분할(Divide)

- 크기가 n인 입력을 3개로 분할하고 각각 분할된 부분 문제의 크기가 n/2이면 다음과 같이 문제가 분할됨



※ 출처 : 알기 쉬운 알고리즘, 양성봉, 생능출판사

1 분할 정복

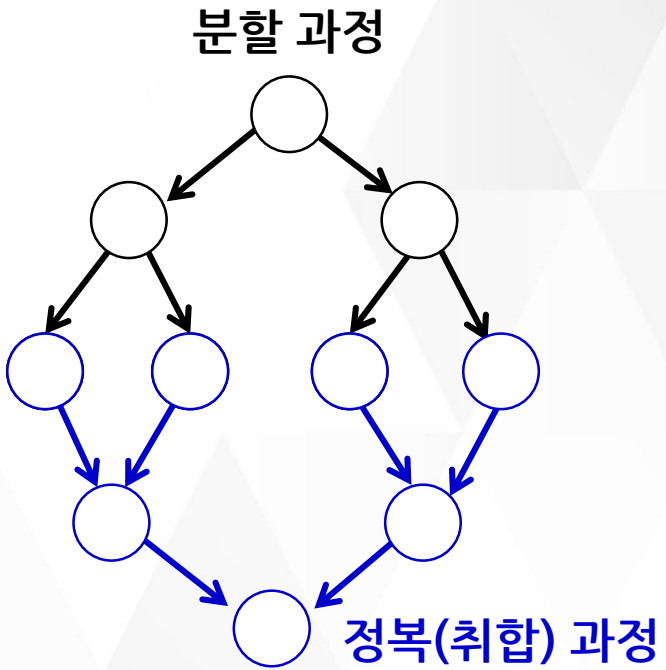
3 정복(Conquer)

- 대부분의 분할 정복 알고리즘은 문제의 입력을 단순히 분할만 해서는 해를 구할 수 없음
- 분할된 부분 문제들을 정복해야 하는데 정복이란 부분해를 찾는 것을 의미함
- 정복하는 방법은 문제에 따라 다르나 일반적으로 부분 문제들의 해를 취합하여 보다 큰 부분 문제의 해를 구함
- 부분 문제를 재귀적으로 풀어서 정복함

1 분할 정복

3 정복(Conquer)

- 부분 문제의 크기가 충분히 작으면 직접적인 방법으로 품
- 예)
  - 2개의 n/2 원소를 정렬하는 문제를 각각 재귀 알고리즘으로 해결하는 것



※ 출처 : 알기 쉬운 알고리즘, 양성봉, 생능출판사

1 분할 정복

4 결합(Combine)

- 부분 문제의 해를 결합하여
  원래 문제의 해가 되도록 만듦
- 분할되어 해결된 문제를 합치면
  커다란 문제의 답이 나옴
- 예)
  - 2개의 정렬된 원소를 병합하는 것

1 분할 정복

5 알고리즘을 설계하는 요령

① 분할(Divide)
: 문제가 분할이 가능한 경우 2개 이상의
하위 문제로 나눔

② 정복(Conquer)
: 하위 문제가 여전히 분할이 가능한 상태라면
하위 집합에 대해 ①을 수행하고 그렇지 않다면
하위 문제를 품

③ 결합(Combine)
: ② 과정에서 정복된 답을 취합함

5 분할 정복과 동적 계획법

공통점

- 문제를 더 작은 부분 문제들로 쪼개어 나가고, 그 답들로부터 원래 문제의 답을 도출해 내는 방식이 같음
- 분할 정복과 동적 계획법의 차이가 발생하는 부분은 부분 문제들의 관계와 문제를 나누는 방식임

5 분할 정복과 동적 계획법

차이점

- 분할 정복의 부분 문제들은 서로 독립적인 관계를 유지하고 동적 계획법의 부분 문제들은 의존적 관계가 존재함

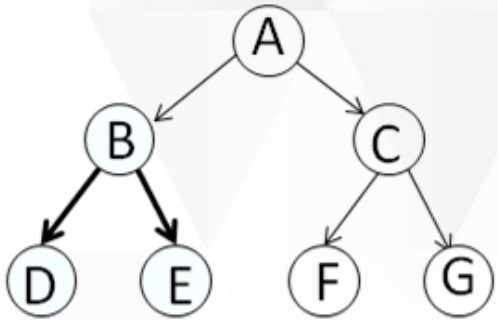
- 어떤 문제들은 부분 문제들로 쪼개어 풀 때 같은 답 하나를 구하는데 두번 쪼개어 풀 때가 있음

  즉, 중복되는 계산 과정이 존재하는데 동적 계획법은 이러한 중복을 개선하여 속도 향상을 꾀하는 알고리즘

5 분할 정복과 동적 계획법

분할 정복 알고리즘

- 하향식(top-down) 접근 방법으로 최상위 사례의 해답은 아래로 내려가면서 작은 사례에 대한 해답을 구함으로써 구함
- A는 B와 C로 분할되고, B는 D와 E로 분할되는데, D와 E의 해를 취합하여 B의 해를 구함
- D, E, F, G는 각각 더 이상 분할할 수 없는 (또는 가장 작은 크기의) 부분 문제들
- F와 G의 해를 취합하여 C의 해를 구하고, 마지막으로 B와 C의 해를 취합하여 A의 해를 구함



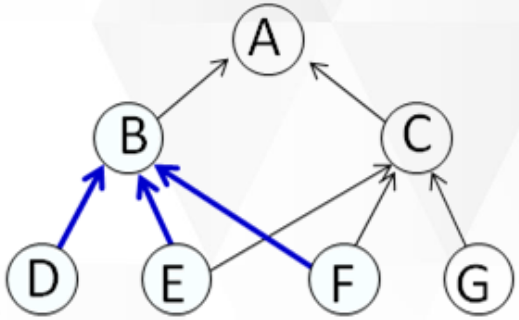
[분할 정복 알고리즘]

※ 출처 : 알기 쉬운 알고리즘, 양성봉, 생능출판사

5 분할 정복과 동적 계획법

동적 계획법 알고리즘

- 어떤 부분 문제는 두 개 이상의 문제를 푸는데 사용될 수 있기 때문에, 이 문제에 답을 여러 번 계산하는 대신 한 번만 계산하고 계산 결과를 재활용함으로써 속도의 향상을 꾀함
- 먼저 최소 단위의 부분 문제 D, E, F, G의 해를 각각 구함
- 그 다음에 D, E, F의 해를 이용하여 B의 해를 구함



[동적 계획법 알고리즘]

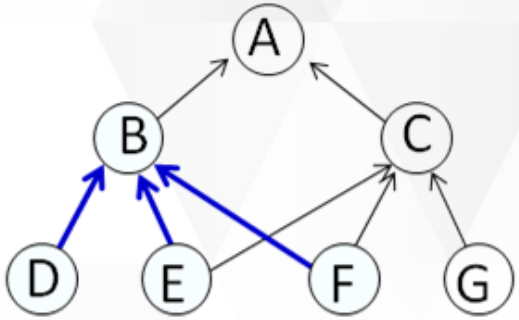
※ 출처 : 알기 쉬운 알고리즘, 양성봉, 생능출판사

5 분할 정복과 동적 계획법

동적 계획법 알고리즘

- E, F, G의 해를 이용하여 C의 해를 구함
- B와 C의 해를 구하는데 E와 F의 해 모두를 중복 이용
- 동적 계획법은 부분 문제들 사이에 의존적 관계가 존재

→ **분할 정복 알고리즘은**
부분 문제의 해를 중복 사용하지 않음



[동적 계획법 알고리즘]

※ 출처 : 알기 쉬운 알고리즘, 양성봉, 생능출판사

6 분할 정복 예

◆ 예)
입력 크기가 n일 때 총 몇 번 분할하여야
더 이상 분할할 수 없는 크기인 1이 될까?

(풀이)
답을 계산하기 위해서 총 분할한 횟수= k라고 하면

1번 분할 후 각각의 입력 크기는 $n/2$
2번 분할 후 각각의 입력 크기는 $n/2^2$
…
k번 분할 후 각각의 입력 크기는 $n/2^k$

→ 따라서 $n/2^k = 1$일 때 더 이상 분할할 수 없으므로 $k = \log_2 n$ 임

2 분할 정복의 적용

2 분할 정복의 적용

1 병합 정렬 (Merge Sort)

- 병합 정렬은 입력이 2개의 부분 문제로 분할하여 부분 문제의 크기가 1/2로 감소하는 분할 정복 알고리즘
- n개의 숫자들을 n/2개씩 2개의 부분 문제로 분할하고, 각각의 부분 문제를 재귀적으로 병합 정렬한 후, 2개의 정렬된 부분을 병합하여 정렬(정복)함
- 병합 과정이 문제를 정복하는 것임

2 분할 정복의 적용

2 병합(Merge)

- 병합이란 2개의 각각 정렬된 숫자들을
  1개의 정렬된 숫자들로 합치는 것

배열 A : 6 14 18 20 29

→ 배열 C : 1 2 6 14 15 18 20 25 29 30 45

배열 B : 1 2 15 25 30 45

2 분할 정복의 적용

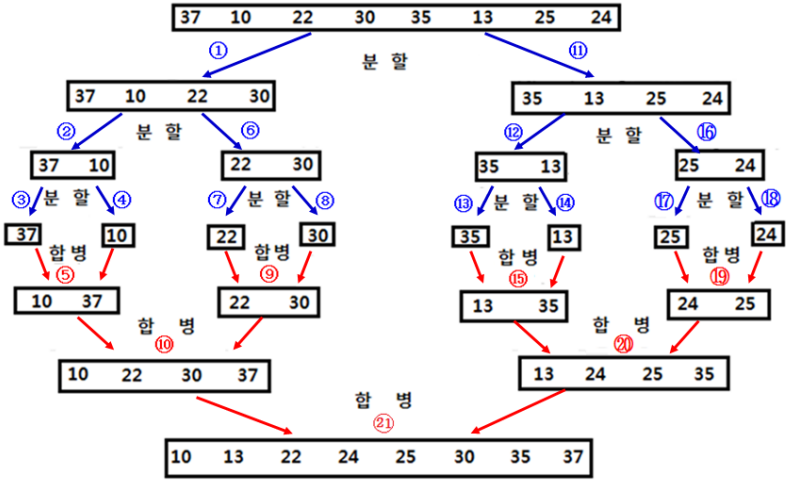
3 병합 정렬 알고리즘

```
MergeSort(A, p, q)
▷ 입력: A[p]~A[q]
▷ 출력: 정렬된 A[p]~A[q]
{
 if ( p < q ) {                ▷ 배열의 원소의 수가 2개 이상이면
      k = ⌊(p+q)/2⌋            ▷ k=반으로 나누기 위한 중간 원소의 인덱스
      MergeSort(A, p, k)       ▷ 앞부분 재귀 호출
      MergeSort(A, k+1, q)     ▷ 뒷부분 재귀 호출
      A[p]~A[k]와 A[k+1]~A[q]를 병합한다.
   }
}
```

2 분할 정복의 적용

3 병합 정렬 알고리즘

- 예)
  - 입력 크기 n=8인 배열 A=[37, 10, 22, 30, 35, 13, 25, 24]에 대하여 병합 정렬되는 과정



※ 출처 : 알기 쉬운 알고리즘, 양성봉, 생능출판사

2 분할 정복의 적용

4 병합 정렬의 시간복잡도

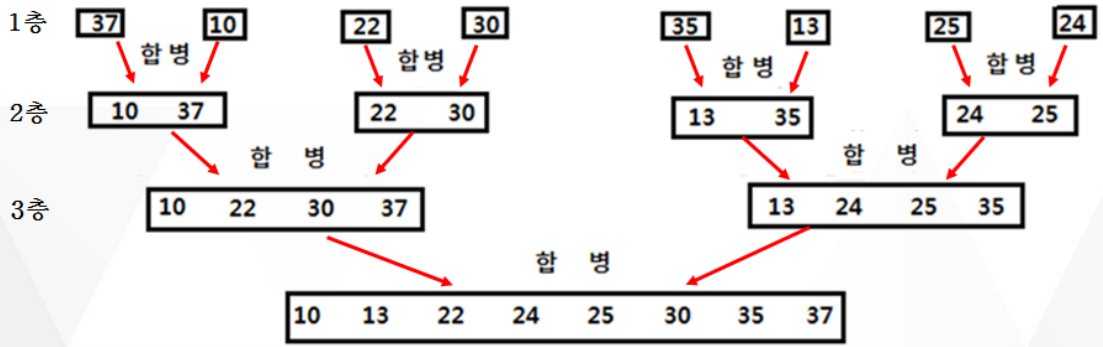
- 분할하는 부분은 배열의 중간 인덱스 계산과 2번의 재귀 호출이므로 O(1) 시간 소요
- 병합의 수행 시간은 입력의 크기에 비례하고 만약 2개의 정렬된 배열 A와 B의 크기가 각각 n과 m이라면 최대 비교 횟수는 **(n+m-1)**임

※ 출처 : 알기 쉬운 알고리즘, 양성봉, 생능출판사

2 분할 정복의 적용

4 병합 정렬의 시간복잡도

- 각 층을 살펴보면 모든 숫자(즉, n=8개의 숫자)가 합병에 참여
- 합병은 입력 크기에 비례하므로 각 층에서 수행된 비교 횟수는 **O(n)**



※ 출처 : 알기 쉬운 알고리즘, 양성봉, 생능출판사

2 분할 정복의 적용

4 병합 정렬의 시간복잡도

- 층수를 세어보면, 8개의 숫자를 반으로, 반의 반으로, 반의 반의 반으로 나눔
- 이 과정을 통하여 3층이 만들어짐

입력 크기	예	층
n	8	
n/2	4	1층
$n/4 = n/2^2$	2	2층
$n/8 = n/2^3$	1	3층

※ 출처 : 알기 쉬운 알고리즘, 양성봉, 생능출판사

2 분할 정복의 적용

4 병합 정렬의 시간복잡도

- 입력의 크기가 n일 때 몇 개의 층이 만들어질까?
- n을 계속하여 1/2로 나누다가, 더 이상 나눌 수 없는 크기인 1이 될 때 분할을 중단함
- 따라서 k번 1/2로 분할했으면 k개의 층이 생기는 것이고, k는 $n=2^k$으로 부터 **$\log_2 n$**임을 알 수 있음

※ 병합 정렬의 시간복잡도

(층수) x O(n) = $\log_2 n$ x O(n) = **O(n logn)**

2 분할 정복의 적용

5 병합 정렬의 단점

- 병합 정렬의 **공간 복잡도 : O(n)**
- 입력을 위한 메모리 공간(입력 배열)외에 추가로
  입력과 같은 크기의 공간 (임시 배열)이 별도로 필요
- 2개의 정렬된 부분을 하나로 합병하기 위해
  병합된 결과를 저장할 곳이 필요하기 때문

2 분할 정복의 적용

6 병합 정렬 응용

- 병합 정렬은 **외부 정렬의 기본**이 되는 정렬 알고리즘
- **연결 리스트에 있는 데이터를 정렬**할 때에도 퀵 정렬이나 힙 정렬보다 훨씬 효율적임
- 멀티코어 CPU와 다수의 프로세서로 구성된 그래픽 처리 장치의 등장으로 정렬 알고리즘을 **병렬화**하는 데에 병합 정렬 알고리즘이 활용

3 분할 정복의 장단점

3 분할 정복의 장단점

1 분할 정복의 장단점

분할 정복의 장점

- 문제를 나눔으로써 어려운 문제를 해결할 수 있음
- 문제를 나누어 해결한다는 특징상
  병렬적으로 문제 해결 가능

3 분할 정복의 장단점

1 분할 정복의 장단점

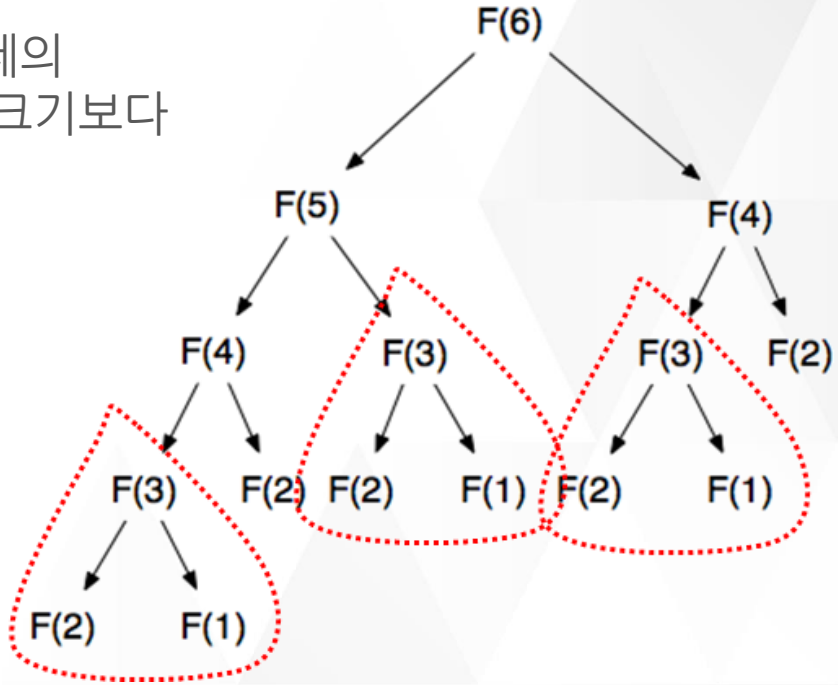
분할 정복의 단점

- 함수를 재귀적으로 호출함으로 인해
  함수 호출로 인한 오버헤드가 발생함
- 스택에 다양한 데이터를 보관하고 있어야 하므로
  스택 오버플로우가 발생하거나 과도한 메모리
  사용을 하게 될 수도 있음

3 분할 정복의 장단점

2 분할 정복이 부적절한 경우

- 입력이 분할될 때마다 분할된 부분 문제의 입력 크기의 합이 분할되기 전의 입력 크기보다 매우 커지는 경우
- 예) 피보나치 수열
  - 피보나치 수 F(6)을 구하기 위해 F(2)를 5번이나 중복하여 계산해야 하고, F(3)은 3번 계산됨



※ 출처 : 알기 쉬운 알고리즘, 양성봉, 생능출판사

3 분할 정복의 장단점

2 분할 정복이 부적절한 경우

- n 번째의 피보나치 수를 구하는데
  F(n) = F(n-1) + F(n-2)로 정의
- 재귀 호출을 사용하는 것이 자연스러워 보이나 이 경우의 입력은 1개이지만 사실상 **n의 값 자체가 입력 크기**임
- n이라는 숫자로 인해 2개의 부분 문제인 F(n-1)과 F(n-2)가 만들어짐
- 2개의 입력 크기의 합이 (n-1) + (n-2) = (2n-3)이 되어서, **분할 후 입력 크기가 거의 2배로 늘어남**

  → 이 경우 분할 정복 알고리즘을 사용하는 것은 매우 부적절하며 다른 방법을 찾아야 함

3 피보나치 수 계산을 위한 O(n) 시간 알고리즘

- for 루프를 사용하여 해결

```
FibNumber(n)
{
   F[0]=0
   F[1]=1
   for i=2 to n
      F[i] = F[i-1]+ F[i-2]
}
```

→ 피보나치 수 F(n)을 중복된 계산 없이 구할 수 있음
이 알고리즘의 복잡도는 O(n)임

3 분할 정복의 장단점

4 분할 정복 적용시 주의할 점

- 규모가 큰 문제를 작은 규모의 모듈로 분할하여 문제를 해결하는 방법은 매우 유용하나 모듈로 나누면 모듈끼리 서로 통신하는 방법이 필요함

- 무작정 작게만 쪼개면 통신으로 인해 복잡도가 오히려 증가될 수 있으므로 설계자는 어느 수준으로 쪼갤지를 결정해야 함

  즉, 복잡도로 인한 증가 비용과 처리의 용이성을 고려하여 결정해야 함

3 분할 정복의 장단점

4 분할 정복 적용시 주의할 점

- 주어진 문제를 분할 정복으로 해결하려고 할 때에 주의해야 하는 또 하나의 요소는 취합(정복) 과정임
- 입력을 분할만 한다고 해서 효율적인 알고리즘이 만들어지는 것은 아니며 취합 과정이 문제 해결에 잘 부합되어야 함
- 취합이 간단하거나 필요 없거나 조금 복잡한 편인 경우 효율적인 분할 정복 알고리즘으로 해결 가능