3장 분할 정복 알고리즘 (Divide-and-Conquer)

지난 시간에는…

- What is Algorithm?
  - strategy for solving a problem
- How to evaluate the algorithm?
  - correctness
  - runnability
  - finiteness
  - efficiency

지난 시간에는…

- Representations of Algorithm
  - Natural language
  - Flow chart
  - Pseudo code
  - Programming code

학습목표 및 내용

분할 정복 알고리즘의 이해 (Divide-and-Conquer)

- 합병 정렬 (Merge Sort)
- 퀵 정렬 (Quick Sort)
- 선택 문제

분할 정복 알고리즘

- 주어진 문제의 입력을 분할하여 문제를 해결(정복)하는 방식의 알고리즘
  - 문제를 나누어서 풀면 더 효율적인 경우 사용

1. Divide - Decomposite a problem to smaller problems (sub problems)
2. Conquer - Get answers of subproblems

분할 정복 알고리즘

- 부분문제와 부분해
  - 분할된 입력에 대한 문제를 부분문제 (subproblem)
  - 부분문제의 해를 부분해
  - 부분문제는 더 이상 분할할 수 없을 때가지 계속 분할
  - 분할한 입력에 대하여 동일한 알고리즘을 적용하여 해를 계산
  - 이들의 해를 취합하여 원래 문제의 해를 얻음

분할 정복 알고리즘의 예

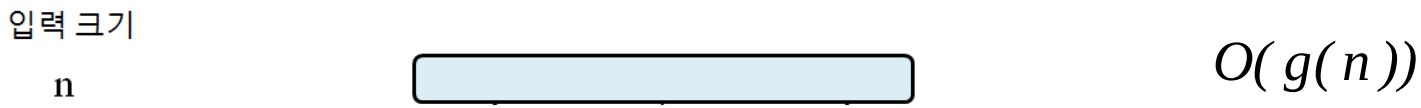
- 영어를 배운다…
  1. 알파벳
  2. 단어
  3. 문장
  4. 회화

왜 이 방식을 사용할까?

- 입력크기(사이즈)과 시간 복잡도와의 상관관계

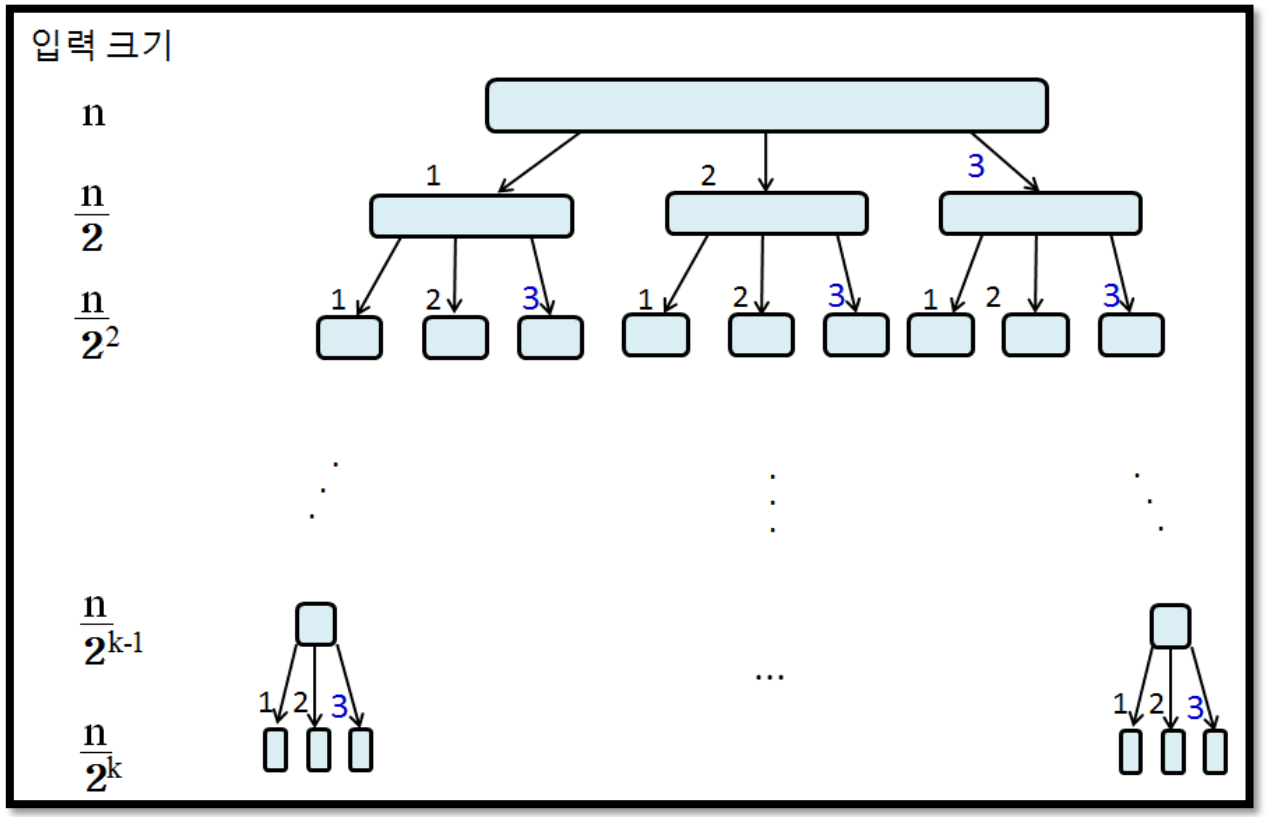
분할

- 크기가 n인 입력을 3개로 분할하고, 각각 분할된 부분 문제의 크기가 n/2일 경우의 분할 예



분할

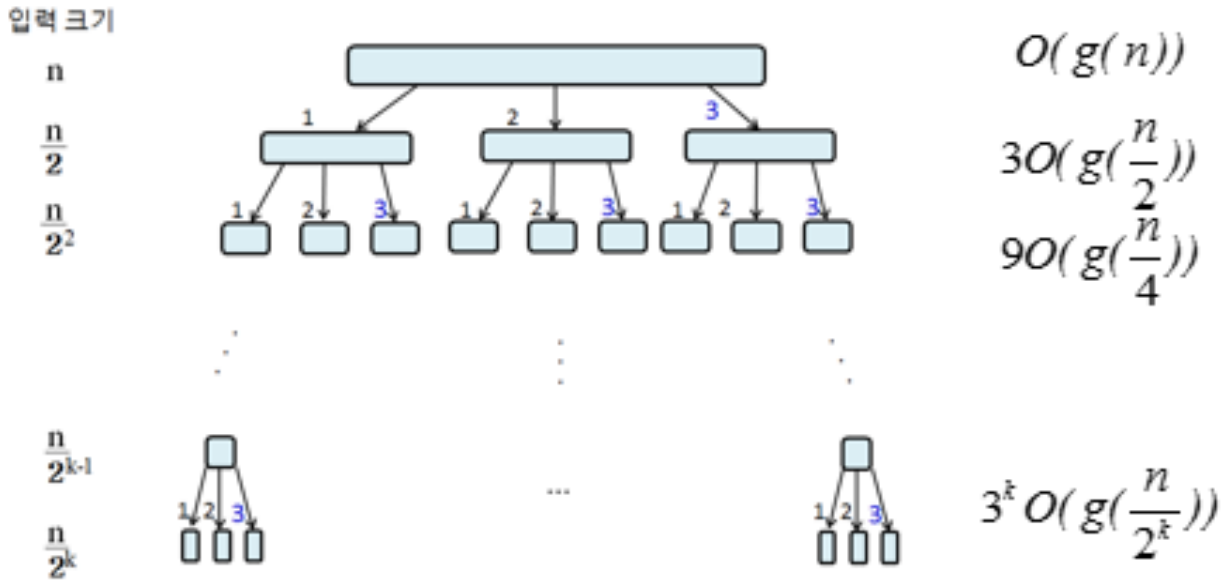
- 입력 크기가 n일 때 총 분할 횟수
  - 총 분할한 횟수 = k라고 놓는다.
  - 1번 분할 후 각각의 입력 크기 n/2
  - 2 번 분할 후 각각의 입력 크기 $n/2^2$
  - ……
  - k 번 분할 후 각각의 입력 크기 $n/2^k$
  - $n/2^k = 1$일 때 더 이상 분할할 수 없음
  - $k = \log_2 n$



분할

- if $g(n) = n^2$, relation of k regarding time complexity?

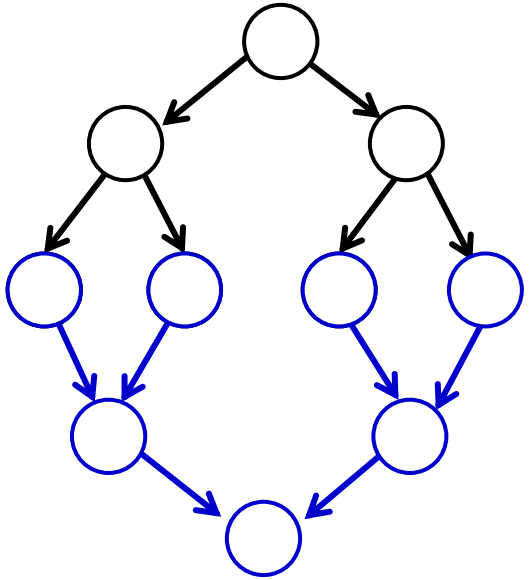
$$O\big(g(n)\big) = O(N^2) \Rightarrow \frac{n}{2^k} = N$$



정복 과정 + 합병(Merging)

- 대부분의 분할 정복 알고리즘
  - 분할만 해서는 해를 구할 수 없다.
- 따라서 분할된 부분 문제들을 정복해야 함
  - 부분해를 찾아야 한다.
  - 정복하는 방법은 문제에 따라 다르다.
  - 일반적으로 부분 문제들의 해를 취합하여 보다 큰 부분 문제의 해를 구한다.

분할 과정



정복 (취합) 과정

분할 정복 알고리즘의 분류

A problem => "**a**" subproblems with size "**1/b**"

- a=b=2 => merge sort
- a=2, b=varies by inputs => quick sort
- a=2, 1/b=0 or 2 => binary search
- a=2, 1/b=0 and varies by inputs => selection
- a=2, b=decreases by each division => insertion sort

1. 합병 정렬 (Merge Sort)

- 합병 정렬은 입력이 2개의 부분문제로 분할되고, 부분문제의 크기가 1/2로 감소하는 분할 정복 알고리즘
  - n개의 숫자들을 n/2개씩 2개의 부분문제로 분할 (Divide)
  - 각각의 부분문제를 해결(Conquer)
  - 재귀적으로 합병 정렬 (Merging)

합병 (merge)

- 2개의 각각 정렬된 숫자들을 1개의 정렬된 숫자로 합치는 것

배열 A: **6 14 18 20 29**

배열 B: **1 2 15 25 30 45**

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

합병 정렬과의 비교를 위한 선택 정렬

- Selection sort
  1. 최대 원소 찾기
  2. 최대 원소를 오른쪽 끝 원소와 교환
  3. 오른쪽 끝 원소를 제외
  4. 1~3 과정을 하나의 원소가 남을 때 가지 반복

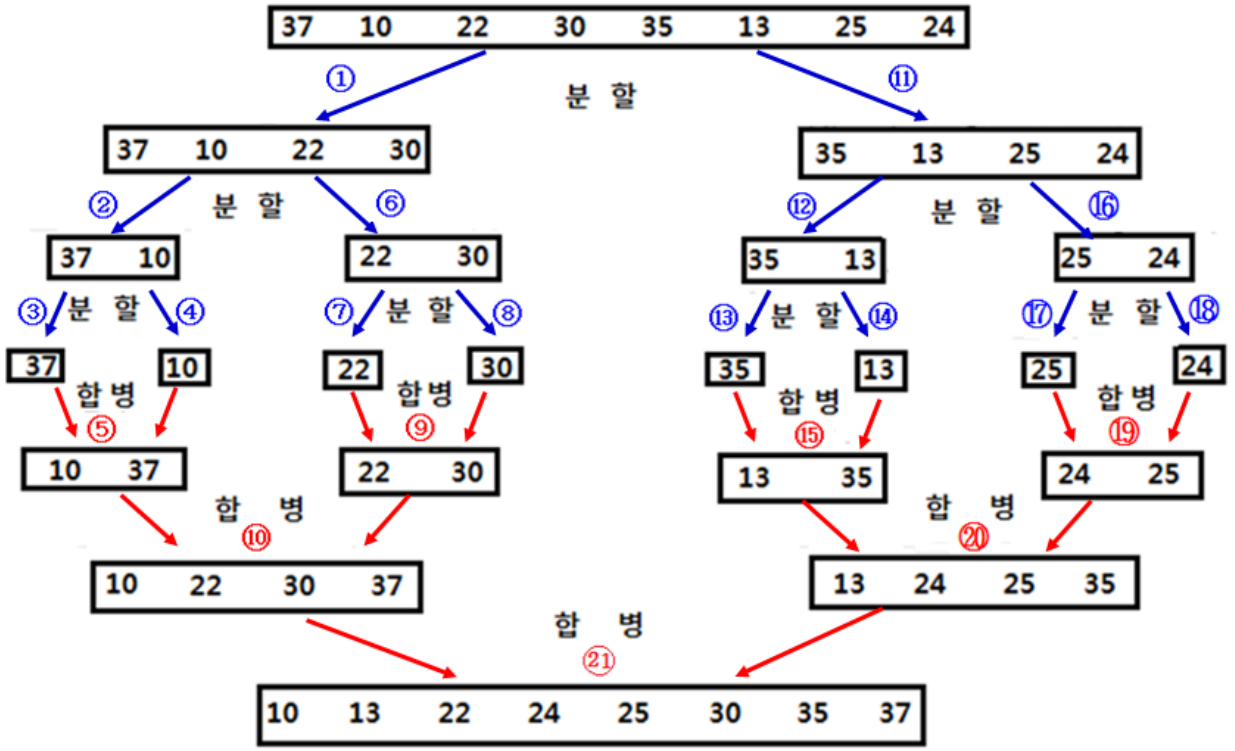
Time complexity of selection sort?

합병 정렬과의 비교를 위한 선택 정렬

- n=8인 배열 A=[37, 10, 22, 30, 35, 13, 25, 24]

합병 정렬 알고리즘의 수행 과정

- n=8인 배열 A=[37, 10, 22, 30, 35, 13, 25, 24]



합병 정렬 알고리즘(Pseudo Code)

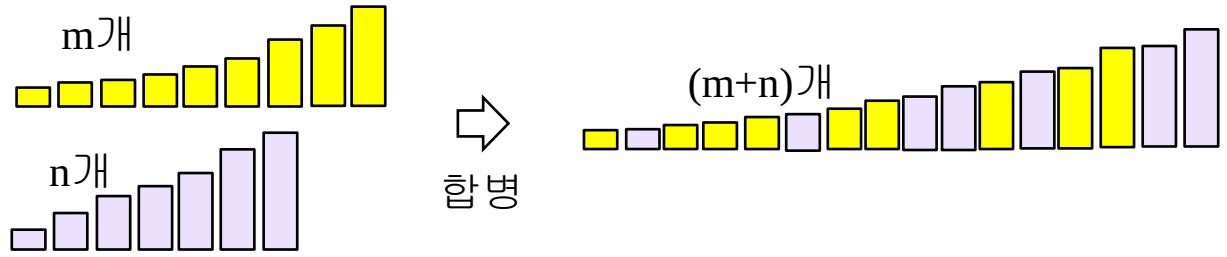
```
MergeSort(A,p,q)
입력: A[p]~A[q]
출력: 정렬된 A[p]~A[q]
1. if ( p < q ) {                  // 배열의 원소의 수가 2개 이상이면
2.    k = ⌊(p+q)/2⌋                // k: 반으로 나누기 위한 중간 원소의 인덱스
3.    MergeSort(A,p,k)             // 앞부분 재귀 호출
4.    MergeSort(A,k+1,q)           // 뒷부분 재귀 호출
5.    A[p]~A[k]와 A[k+1]~A[q]를 합병한다.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

합병 정렬 알고리즘

- Line 1
  - 정렬할 부분의 원소의 수가 2개 이상일 때에만 다음 단계 수행. 만일 n=1이면, 그 자체로 정렬된 것이므로 어떤 수행을 할 필요 없이 이전 호출했던 곳으로 리턴
- Line 2
  - 정렬할 부분의 원소들을 ½로 나누기 위해, k = ⌊(p+q)/2⌋를 계산. 즉, 원소의 수가 홀수인 경우에는 k는 소수점 이하를 버림
- Line 3~4
  - MergeSort(A,p,k)와 MergeSort(A,k+1,q)를 재귀 호출하여 각각 정렬
- Line 5
  - line 3~4에서 각각 정렬된 부분을 합병
  - 합병 과정의 마지막에는 임시 배열에 있는 합병된 원소들을 배열 A로 복사. 즉, 임시 배열 B[p]~B[q]를 A[p]~A[q]로 복사

시간복잡도(교재)

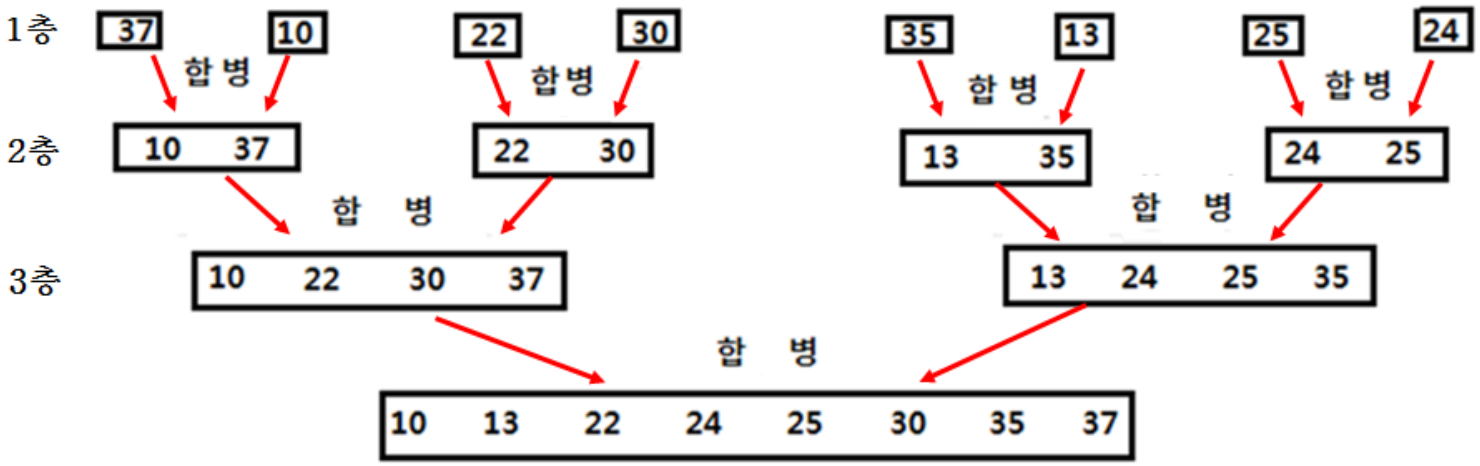
- 분할하는 부분은 배열의 중간 인덱스 계산과 2번의 재귀 호출이므로 O(1) 시간 소요
- 합병의 수행 시간은 입력의 크기에 비례.
  - 2개의 정렬된 배열 A와 B의 크기가 각각 m과 n이라면, 최대 비교 횟수 = (m+n-1)
  - 합병의 시간복잡도 = O(m+n)



- 합병 정렬에서 수행되는 총 비교 횟수
  - 합병 복잡도 * 각 층에서 발생하는 복잡도

시간복잡도(교재)

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



시간복잡도(교재)

- 층수의 계산
  - 층수를 세어보면, 8개의 숫자를 반으로, 반의 반으로 반의 반의 반으로 나눈다.
  - 이 과정을 통하여 3층이 만들어진다.

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

시간복잡도(교재)

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

- 합병 정렬의 시간복잡도:
  - (층수) x O(n) = $\log_2 n$ x O(n) = O(nlogn)

합병 정렬의 단점

- 대부분의 정렬 알고리즘들은 입력을 위한 메모리 공간과 O(1) 크기의 메모리 공간만을 사용하면서 정렬 수행
  - O(1) 크기의 메모리 공간이란 입력 크기 n과 상관없는 크기의 공간

- 합병 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, 합병된 결과를 저장할 곳이 필요하기 때문

응용

- 합병 정렬은 외부정렬의 기본이 되는 정렬 알고리즘
- 연결 리스트에 있는 데이터를 정렬할 때에도 퀵정렬이나 힙정렬 보다 훨씬 효율적
- 멀티코어 (Multi-Core) CPU와 다수의 프로세서로 구성된 그래픽 처리 장치 (Graphic Processing Unit)의 등장으로 정렬 알고리즘을 병렬화 하는 데에 합병 정렬 알고리즘이 활용