

제3장 분할 정복 알고리즘 (I)

과 목 명 정 보 처 리 알 고 리 즘

담당교수 김 성 훈

경 북 대 학 교 과 학 기 술 대 학 소 프 트 웨 어 학 과

이 장에서 배울 내용

1. 분할정복 개념
2. 합병 정렬 (Merge Sort)
3. 퀵 정렬 (Quick Sort)
4. 선택문제
5. 최근접 점의 쌍 찾기
6. 분할정복을 적용하는데 있어서 주의할 점

나폴레옹의 분할정복 전략 (Divide-and-Conquer Strategy)

- ✿ 1805년 12월 2일, 아우스터리츠 전투
- ✿ 적 오스트리아-러시아 연합군보다 1만5천명 열세
- ✿ 나폴레옹 군대는 적 연합군의 중앙으로 진격,
- ✿ 적을 두개 진영으로 갈라놓고서,
- ✿ 나뉘진 적을 각개격파해서 전체 전투를 승리하였다.

분할정복(Divide-and-Conquer) 알고리즘

✳ 주어진 문제의 입력을 분할하여 문제를 해결 또는 정복하는 방식의 알고리즘

- 분할한 입력에 대하여 또다시 동일한 알고리즘을 적용하여 해를 계산하며, 이들의 해를 취합하여 원래 문제의 해를 얻는다.
- 분할된 입력에 대한 문제를 부분문제 (subproblem)라고 하고, 부분 문제의 해를 부분해라고 한다.
- 부분문제는 더 이상 분할할 수 없을 때까지 계속 분할하게 된다.

부분 문제

정복

문 제

분할



취합

문제 해

- ❁ 크기가 n 인 입력을 3개로 분할하고, 각각 분할된 부분 문제의 크기가 $n/2$ 라고 하면, 아래의 그림처럼 문제가 분할된다.

입력 크기

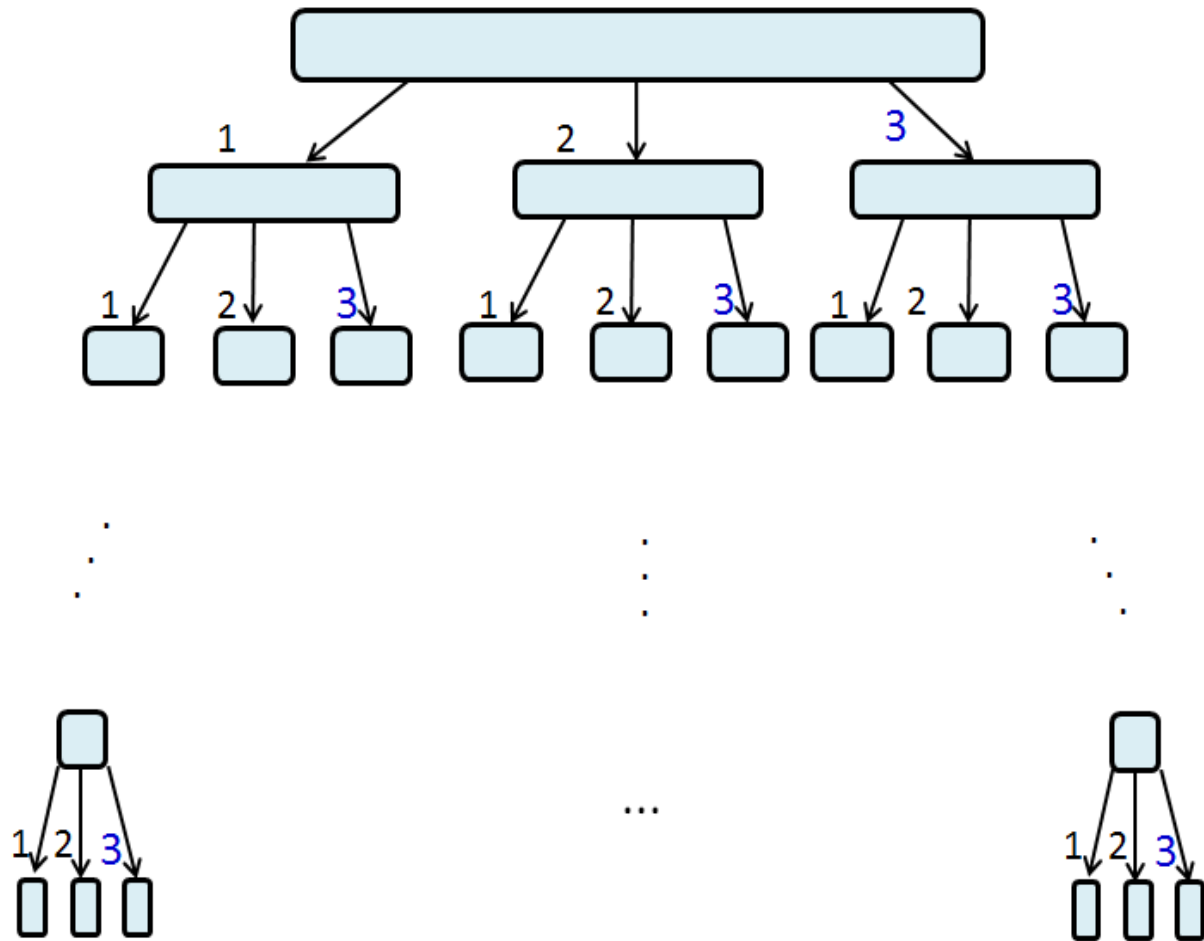
n

$\frac{n}{2}$

$\frac{n}{2^2}$

$\frac{n}{2^{k-1}}$

$\frac{n}{2^k}$



.....

- ✿ 입력 크기가 n 일 때 총 몇 번 분할하여야 더 이상 분할할 수 없는 크기인 1이 될까?

- ✿ 답을 계산하기 위해서 총 분할한 횟수 = k 라고 하자.

- ✿ 1번 분할 후 각각의 입력 크기가 $n/2$

- ✿ 2번 분할 후 각각의 입력 크기가 $n/2^2$

- ✿ ...

- ✿ k 번 분할 후 각각의 입력 크기가 $n/2^k$

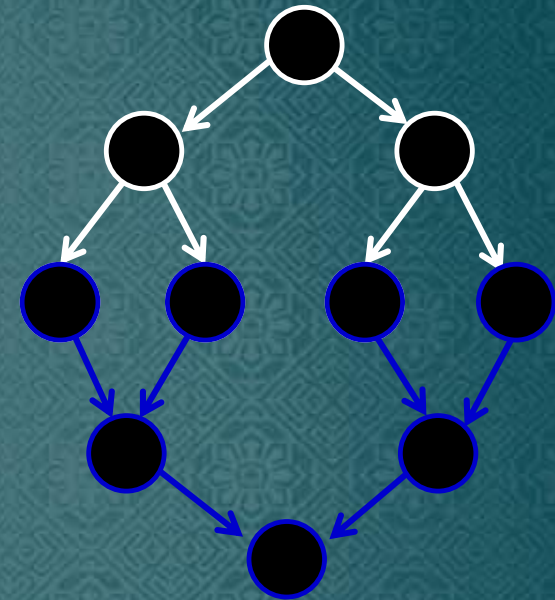
- ✿ 따라서 $n/2^k = 1$ 일 때 더 이상 분할할 수 없다.

- ✿ $k = \log_2 n$ 이다.

정복 과정

- ❁ 대부분의 분할 정복 알고리즘은 문제의 입력을 단순히 분할만 해서는 해를 구할 수 없다.
- ❁ 따라서 분할된 **부분 문제들을 정복해야 한다**. 즉, **부분해**를 찾아야 한다.
- ❁ 정복하는 방법은 문제에 따라 다르나 일반적으로 부분 문제들의 **해를 취합하여** 보다 큰 부분 문제의 해를 구한다.

분할 과정



정복 (취합) 과정

분할 정복 알고리즘의 분류

✿ 문제가 a 개로 분할되고, 부분 문제의 크기가 $1/b$ 로 감소하는 알고리즘:

- $a=b=2$ 인 경우 합병 정렬 (3.1절), 최근접점의 쌍 찾기 (3.4절), 공제선 문제 (연습문제 25)
- $a=3, b=2$ 인 경우. 큰 정수의 곱셈 (연습문제 21)
- $a=4, b=2$ 인 경우, 큰 정수의 곱셈 (연습문제 20)
- $a=7, b=2$ 인 경우, 스트라센(Strassen)의 행렬 곱셈 (연습문제 22)

분할 정복 알고리즘의 분류(계속)

- ❁ 문제가 2개로 분할, 부분문제의 크기가 일정하지 않은 크기로 감소하는 알고리즘: **퀵 정렬** (3.2절)
- ❁ 문제가 2개로 분할, 그 중에 1개의 부분 문제는 고려할 필요 없으며, 부분 문제의 크기가 $1/2$ 로 감소하는 알고리즘: **이진탐색** (1.2절)
- ❁ 문제가 2개로 분할, 그 중에 1개의 부분문제는 고려할 필요 없으며, 부분문제의 크기가 일정하지 않은 크기로 감소하는 알고리즘: **선택 문제 알고리즘** (3.3절)
- ❁ 부분문제의 크기가 1, 2개씩 감소하는 알고리즘: **삽입 정렬** (6.3절), **피보나치 수** (3.5절) 등

3.1 합병 정렬

- ✿ 합병 정렬 (Merge Sort)은 입력이 2개의 부분 문제로 분할, 부분문제의 크기가 $1/2$ 로 감소하는 분할 정복 알고리즘
- ✿ n 개의 숫자들을 $n/2$ 개씩 2개의 부분문제로 분할하고, 각각의 부분문제를 또다시(재귀적으로) 합병정렬을 하게 한 후,
- ✿ 이 결과, 2개의 정렬된 부분을 합병하면서 정렬(정복)한다.
- ✿ 합병 과정이 최종적으로 (문제를) 정복하는 것이다.

합병 (merge)

✿ 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

합병 정렬 알고리즘

MergeSort(A, p, q)

입력: $A[p] \sim A[q]$

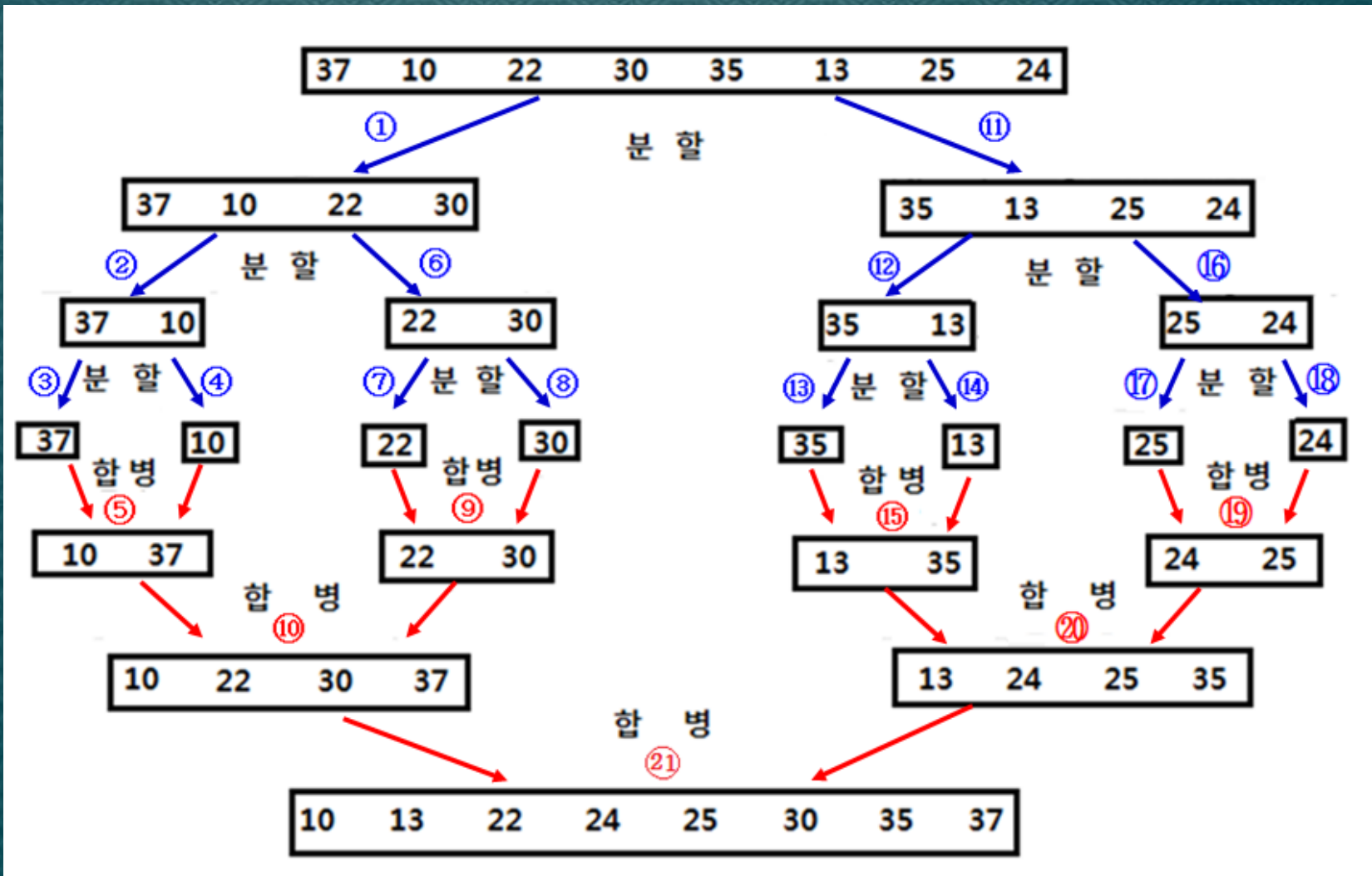
출력: 정렬된 $A[p] \sim A[q]$

1. if ($p < q$) { // 배열의 원소의 수가 2개 이상이면
2. $k = \lfloor (p+q)/2 \rfloor$ // k는 반으로 나누기 위한 중간 원소의 인덱스
3. MergeSort(A,p,k) // 앞부분 배열을 가지고 재귀 호출
4. MergeSort(A,k+1,q) // 뒷부분 배열을 가지고 재귀 호출
5. $A[p] \sim A[k]$ 와 $A[k+1] \sim A[q]$ 를 합병한다.
- }

합병 정렬 알고리즘(해설)

- ✿ Line 1에서는 정렬할 부분의 원소의 수가 2개 이상일 때에만 다음 단계 수행. 만일 $n=1$ 이면, 그 자체로 정렬된 것이므로 어떤 수행할 필요 없이 이전 호출했던 곳으로 리턴
- ✿ Line 2에서는 정렬할 부분의 원소들을 $\frac{1}{2}$ 로 나누기 위해, $k = \lfloor (p+q)/2 \rfloor$ 를 계산. 즉, 원소의 수가 홀수인 경우에는 k 는 소수점 이하는 버림
- ✿ Line 3~4에서는 MergeSort(A,p,k)와 MergeSort(A,k+1,q)를 재귀 호출하여 각각 정렬
- ✿ Line 5에서는 line 3~4에서 각각 정렬된 부분을 합병한다. 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.

✿ 입력 크기가 $n=8$ 인 배열
 $A=[37, 10, 22, 30, 35, 13, 25, 24]$ 에 대하여



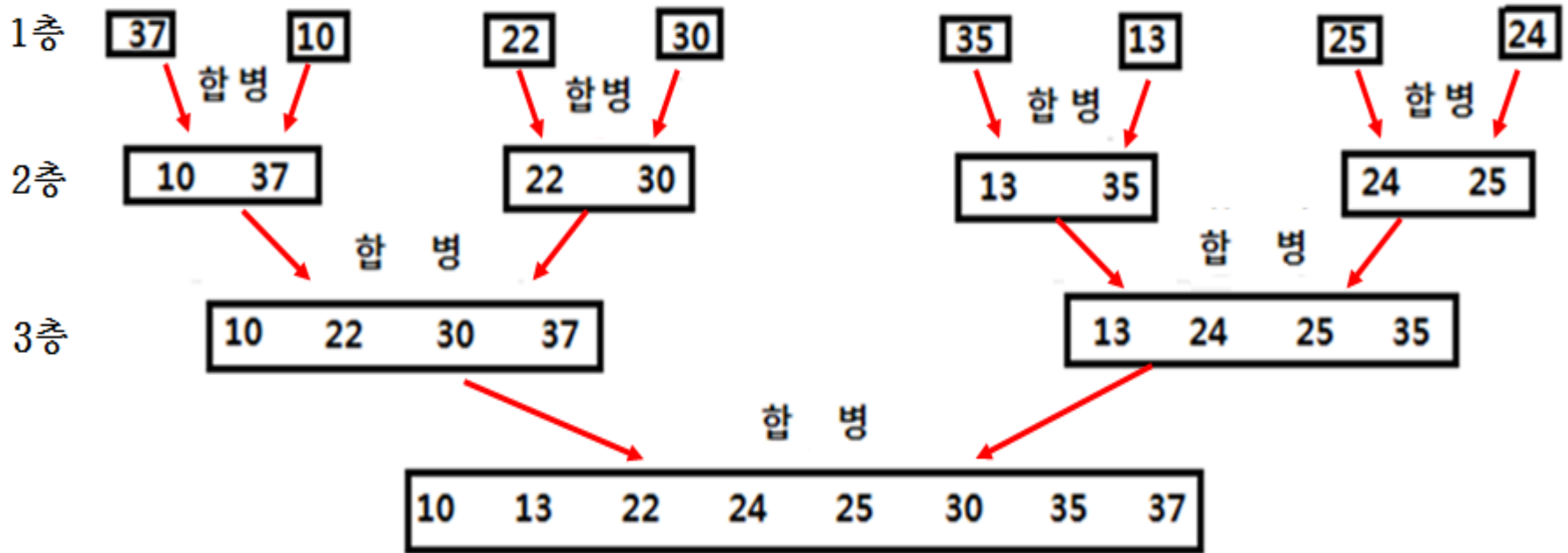
시간복잡도

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



시간복잡도(계속)

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



시간복잡도(계속)

- ✿ 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(계속)

- ✿ 입력의 크기가 n 일 때 몇 개의 층이 만들어질까?
- ✿ n 을 계속하여 $1/2$ 로 나누다가, 더 이상 나눌 수 없는 크기인 1 이 될 때 분할을 중단한다.
- ✿ 따라서 k 번 $1/2$ 로 분할했으면 k 개의 층이 생기는 것이고, k 는 $n=2^k$ 으로 부터 $\log_2 n$ 임을 알 수 있다.
- ✿ 합병 정렬의 시간복잡도:
 $(\text{층수}) \times O(n) = \log_2 n \times O(n) = O(n \log n)$

합병 정렬의 단점

- ✿ 합병 정렬의 공간 복잡도: $O(n)$, 하지만
- ✿ 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

합병정렬의 응용

- ✿ 합병정렬은 **외부정렬의 기본**이 되는 정렬 알고리즘이다.
 - 데이터가 대용량인 경우에, 메모리만으로는 불가능하기 때문.
- ✿ **연결 리스트에 있는 데이터를** 정렬할 때에는 퀵정렬이나 힙정렬 보다 훨씬 효율적이다.
 - 별도의 메모리 공간과 교체연산(자리바꿈)이 불필요하기 때문.
- ✿ 멀티코어 (Multi-Core) CPU와 다수의 프로세서로 구성된 그래픽 처리 장치 (Graphic Processing Unit)의 등장으로 정렬 알고리즘을 **병렬화하는 데에는 합병 정렬 알고리즘이 활용**

BREAK TIME



3.2 퀵 정렬

- ✿ 퀵 정렬 (Quick Sort)은 분할 정복 알고리즘으로 분류되나, 사실 알고리즘이 수행되는 과정을 살펴보면, 정복 후에 분할하는 알고리즘이다.
 - 이때의 정복은 완전한 정복은 아니고, 미완성(?) 정복임.
- ✿ 퀵 정렬 알고리즘은 문제를 2개의 부분문제로 분할하는데, 각 부분문제의 크기가 일정하지 않은 형태의 분할 정복 알고리즘

퀵 정렬의 핵심 아이디어

- ❁ 퀵 정렬은 **피벗 (pivot)**이라 일컫는 배열의 원소(숫자)를 기준으로 피벗보다 작은 숫자들은 왼편으로, 피벗보다 큰 숫자들은 오른편에 위치하도록 분할하고, 피벗을 그 사이에 놓는다.
- ❁ 퀵 정렬은 분할된 부분문제들에 대하여 다시 위와 동일한 과정을 재귀적으로 수행하여 정렬



예시

- ❁ 피벗은 분할된 왼편이나 오른편 부분에 포함되지 않는다. 피벗이 60이라면, 60은 [20 40 10 30 50]과 [70 90 80] 사이에 위치한다.



퀵 정렬 알고리즘

QuickSort(A, left, right)

입력: 배열 $A[\text{left}] \sim A[\text{right}]$

출력: 정렬된 배열 $A[\text{left}] \sim A[\text{right}]$

1. if ($\text{left} < \text{right}$) {
2. 피벗 인덱스 p 를 $A[\text{left}] \sim A[\text{right}]$ 중에서 선택하고,
피벗 $A[p]$ 를 $A[\text{left}]$ 와 자리를 바꾼 후,
피벗 $A[\text{left}]$ 와 배열 $A[\text{left}+1] \sim A[\text{right}]$ 의 각 원소 $A[i]$ 를 비교하여
 피벗보다 작은 숫자들은 $A[\text{left}] \sim A[p-1]$ 로 옮기고,
 피벗보다 큰 숫자들은 $A[p+1] \sim A[\text{right}]$ 로 옮기며,
 피벗 $A[\text{left}]$ 는 $A[p]$ 에 놓는다.
3. QuickSort(A, left, $p-1$) // 피벗보다 작은 그룹을 가지고 재귀호출
4. QuickSort(A, $p+1$, right) // 피벗보다 큰 그룹을 가지고 재귀호출
- }

퀵 정렬 알고리즘(해설)

- ❁ Line 1에서는 배열 A의 가장 왼쪽 원소의 인덱스 (left)가 가장 오른쪽 원소의 인덱스(right)보다 작으면, line 2~4에서 정렬을 수행한다.
만일 그렇지 않으면, 1개의 원소를 정렬하는 경우이다. 1개의 원소는 그 자체가 이미 정렬되어 있으므로, line 2~4의 정렬 과정을 수행할 필요 없이 그대로 호출을 마친다.
- ❁ Line 2에서는 $A[\text{left}] \sim A[\text{right}]$ 에서 피벗을 선택하고, 배열 $A[\text{left}+1] \sim A[\text{right}]$ 의 원소들을 피벗과 비교하여, 피벗보다 작은 그룹인 $A[\text{left}] \sim A[p-1]$ 과 피벗보다 큰 그룹인 $A[p+1] \sim A[\text{right}]$ 로 분할하고 $A[p]$ 에 피벗을 위치시킨다. 즉, **p는 피벗이 위치한 배열 A의 인덱스이다.**
- ❁ Line 3에서는 피벗보다 작은 그룹인 $A[\text{left}] \sim A[p-1]$ 을 재귀 호출한다.
- ❁ Line 4에서는 피벗보다 큰 숫자들은 $A[p+1] \sim A[\text{right}]$ 를 재귀호출한다.

QuickSort(A,0,11) 호출

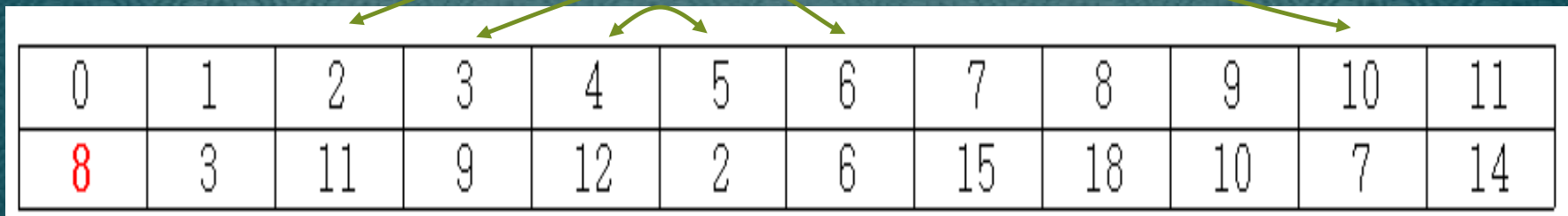
0	1	2	3	4	5	6	7	8	9	10	11
6	3	11	9	12	2	8	15	18	10	7	14

피벗 $A[6]=8$ 이라면, line 2에서 아래와 같이 차례로 원소들의 자리를 바꾼다. 먼저 피벗을 가장 왼쪽으로 이동시킨다.

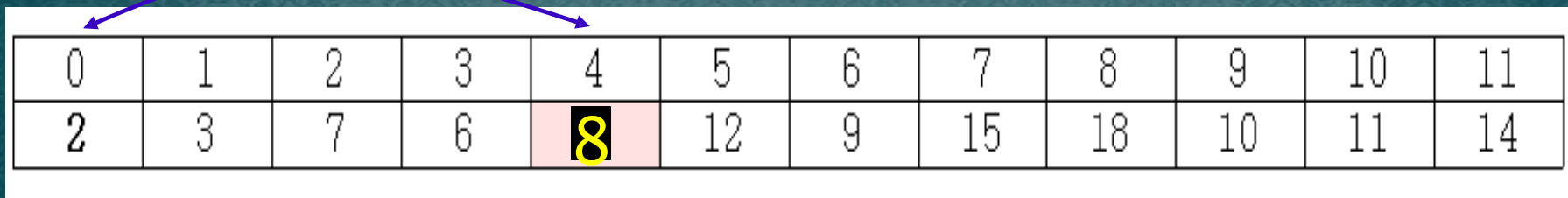
0	1	2	3	4	5	6	7	8	9	10	11
8	3	11	9	12	2	6	15	18	10	7	14

QuickSort(A,0,11) 호출(계속)

- 그 다음엔 피벗보다 큰 수와 피벗보다 작은 수를 다음과 같이 각각 교환



0	1	2	3	4	5	6	7	8	9	10	11
8	3	11	9	12	2	6	15	18	10	7	14



0	1	2	3	4	5	6	7	8	9	10	11
2	3	7	6	8	12	9	15	18	10	11	14

- ✿ line 3에서 QuickSort(A,0,4-1) = QuickSort(A,0,3)이 호출되고,
그 다음 line 4에서 QuickSort(A,4+1,11) = QuickSort(A,5,11)이 호출

QuickSort(A,0,3) 재귀호출

.....

0	1	2	3
2	3	7	6

❁ 피벗 A[3]=6이라면, line 2에서 아래와 같이 원소들의 자리를 바꾼다.

0	1	2	3
6	3	7	2

0	1	2	3
2	3	6	7

QuickSort(A,0,3) 재귀호출(계속)

- ✿ line 3에서 $\text{QuickSort}(A,0,2-1) = \text{QuickSort}(A,0,1)$ 이 호출되고, 그 다음 line 4에서 $\text{QuickSort}(A,2+1,3) = \text{QuickSort}(A,3,3)$ 이 호출


QuickSort(A,0,1) 호출

0	1
2	3

- 피봇 $A[1]=3$ 이라면, line 2에서 아래와 같이 원소들의 자리를 바꾼다.

0	1
3	2

0	1
2	3



QuickSort(A,0,3) 재귀호출(계속)

- ✿ line 3에서 $\text{QuickSort}(A,0,1-1) = \text{QuickSort}(A,0,0)$ 이 호출되고, line 4에서 $\text{QuickSort}(A,1+1,1) = \text{QuickSort}(A,2,1)$ 이 호출
- ✿ $\text{QuickSort}(A,0,0)$ 호출: Line 1의 if-조건이 '거짓'이 되어서 알고리즘을 더 이상 수행하지 않는다.
- ✿ $\text{QuickSort}(A,2,1)$ 호출: Line 1의 if-조건이 '거짓'이므로 알고리즘을 수행하지 않는다.
- ✿ 위의 과정과 유사하게 $A[2] \sim A[3]$ 도 정렬되며, $\text{QuickSort}(A,0,3)$ 이 아래와 같이 완성된다.

0	1	2	3
2	3	7	6

시간복잡도

- ❁ 퀵 정렬의 성능은 피벗 선택이 좌우한다. 피벗으로 가장 작은 숫자 또는 가장 큰 숫자가 선택되면, 한 부분으로 치우치는 분할을 야기한다.

피벗

1	17	42	9	18	23	31	11	26
---	----	----	---	----	----	----	----	----



1	9	42	17	18	23	31	11	26
---	---	----	----	----	----	----	----	----



...

1	9	11	17	18	23	26	31	42
---	---	----	----	----	----	----	----	----



시간복잡도(계속)

- ✿ 피벗=1일 때: 8회 - [17 42 9 18 23 31 11 26]과 각각 1회씩 비교
- ✿ 피벗=9일 때: 7회 - [42 17 18 23 31 11 26]과 각각 1회씩 비교
- ✿ 피벗=11일 때: 6회 - [17 18 23 31 42 26]과 각각 1회씩 비교
- ✿ ...
- ✿ 피벗=31일 때: 1회 - [42]와 1회 비교
- ✿ 총 비교 횟수는 $8+7+6+\dots+1 = 36$ 이다.
- ✿ 입력의 크기가 n 이라면, 퀵 정렬의 최악 경우 시간복잡도
 $= (n-1)+(n-2)+(n-3)+\dots+2+1 = n(n-1)/2 = O(n^2)$

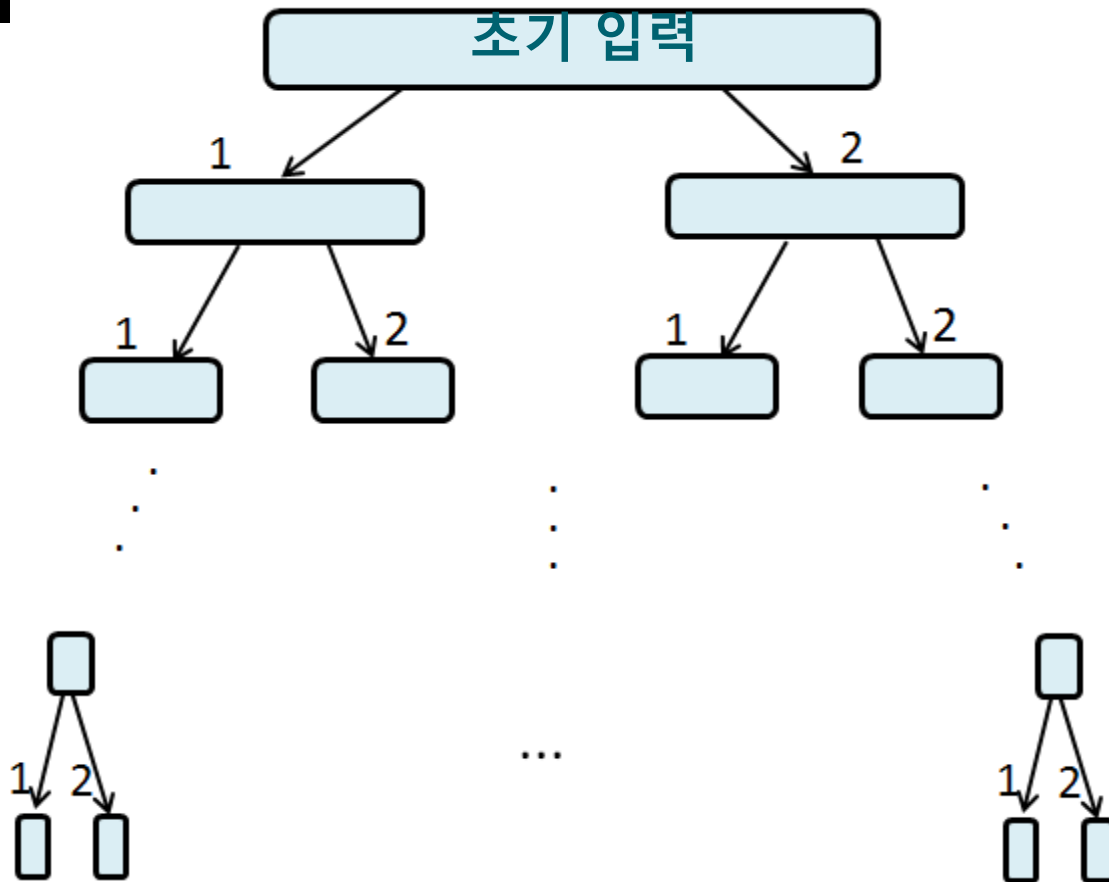
최선 경우의 분할

각 부분의 크기

n

$$\frac{n}{2}$$
$$\frac{n}{2^2}$$
$$\frac{n}{2^{k-1}}$$
$$\frac{n}{2^k}$$

1



1층

2층

k-1층

k층

최선 경우의 분할(계속)

- ✿ 각 층에서는 각각의 원소가 각 부분의 피봇과 1회씩 비교된다. 따라서 비교 횟수 = $O(n)$
- ✿ 총 비교 횟수 = $O(n) \times (\text{층수}) = O(n) \times (\log_2 n)$
- ✿ 층수가 $\log_2 n$ 인 이유는 $n/2^k=1$ 일 때 $k=\log_2 n$ 이기 때문
- ✿ 퀵 정렬의 최선 경우 시간복잡도: $O(n \log_2 n)$

평균 경우 시간복잡도

- ✿ 피벗을 항상 랜덤하게 선택한다고 가정하면, 퀵 정렬의 평균 경우 시간복잡도를 계산할 수 있다. 이때의 시간복잡도도 역시 최선 경우와 동일하게 $O(n\log_2 n)$ 이다.

피벗 선정 방법

✿ 랜덤하게 선정하는 방법

✿ 3 숫자의 중앙값으로 선정하는 방법: 가장 왼쪽 숫자, 중간 숫자, 가장 오른쪽 숫자 중에서 중앙값으로 피벗을 정한다. 아래의 예제를 보면, 31, 1, 26 중에서 중앙값인 26을 피벗으로 사용한다.

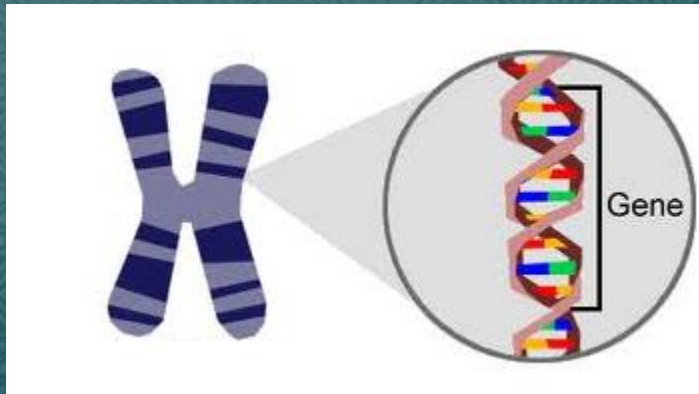
31	17	42	9	1	23	18	11	26
----	----	----	---	---	----	----	----	----

성능 향상 방법

- ✿ 입력의 크기가 매우 클 때, 퀵 정렬의 성능을 더 향상시키기 위해서, **삽입정렬**이 함께 사용된다.
 - 입력의 크기가 작을 때에는 퀵 정렬이 삽입 정렬보다 빠르지만은 않다. 왜냐하면 퀵 정렬은 재귀 호출로 수행되기 때문
 - 부분문제의 크기가 작아지면 (예를 들어, 25에서 50이 되면), 더 이상의 분할(재귀 호출)을 중단하고 삽입 정렬을 사용하는 것이다.

퀵정렬의 응용

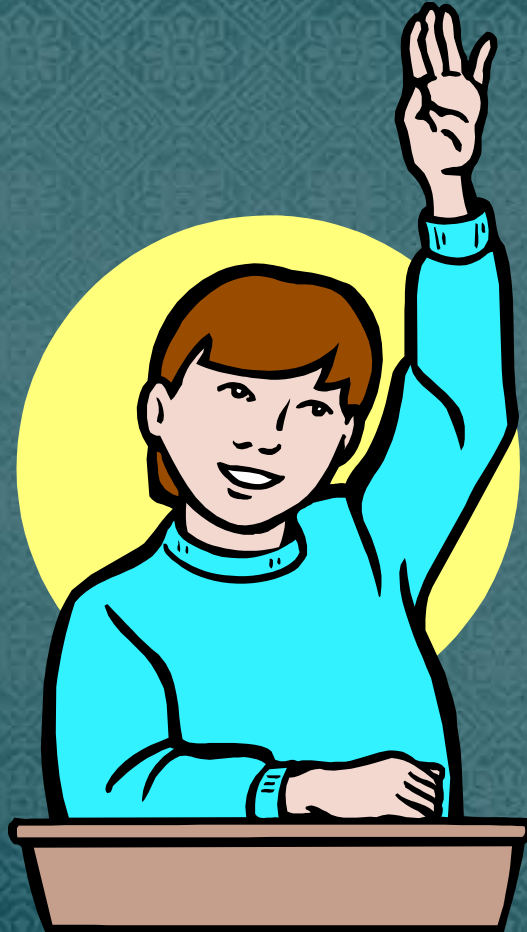
- ❁ 퀵정렬은 커다란 크기의 입력에 대해서 가장 좋은 성능을 보이는 정렬 알고리즘이다.
- ❁ 퀵정렬은 실질적으로 어느 정렬 알고리즘보다 좋은 성능을 보인다.
- ❁ 생물 정보 공학(Bioinformatics)에서 특정 유전자를 효율적으로 찾는데 접미 배열(suffix array)과 함께 퀵 정렬이 활용된다.



요약

- ※ 분할 정복 (Divide-and-Conquer) 알고리즘: 주어진 문제의 입력을 분할하여 문제를 해결 (정복)하는 방식의 알고리즘이다.
- ※ 합병 정렬 (Merge sort): n 개의 숫자들을 $n/2$ 개씩 2개의 부분 문제로 분할하고, 각각의 부분 문제를 재귀적으로 합병 정렬한 후, 2개의 정렬된 부분을 합병하여 정렬 (정복)한다.
 - * 합병 정렬의 시간복잡도는 $O(n \log n)$ 이다.
 - * 공간 복잡도는 $O(n)$ 이다.
- ※ 퀵 정렬 (Quick sort): 피벗 (pivot)이라 일컫는 배열의 원소를 기준으로 피벗보다 작은 숫자들은 왼편으로, 피벗보다 큰 숫자들은 오른편에 위치하도록 분할하고, 피벗을 그 사이에 놓는다. 퀵 정렬은 분할된 부분 문제들에 대하여서도 위와 동일한 과정을 재귀적으로 수행하여 정렬한다.
 - * 퀵 정렬의 평균 경우 시간복잡도는 $O(n \log n)$, 최악 경우 시간복잡도는 $O(n^2)$, 최선 경우 시간복잡도는 $O(n \log n)$ 이다.

Q&A



BREAK TIME



실 습 문 제

1. 합병정렬 알고리즘을 파이선 구현하기
 2. 퀵정렬 알고리즘을 파이선 구현하기
- 숙제: 3장 연습문제 6번과 8번의 알고리즘을 설계하고, 파이선으로 구현하여 실행결과를 보여라. 이때, 적절한 테스트 데이터를 사용하여 각 패스 단계별로 중간과정을 보이고, 알고리즘의 정확성을 검사하라.