CHAPTER.24

고급 정렬 알고리즘의 원리와 구현 및 응용

학습 내용

[1] 고급 정렬 알고리즘의 원리와 구현

[2] 고급 정렬 알고리즘의 응용

학습 목표

- 성능을 향상시킬 수 있는 정렬 방식을 설명할 수 있다.
- 고급 정렬을 활용한 응용 프로그램을 작성할 수 있다.

세종사이버대학교

01

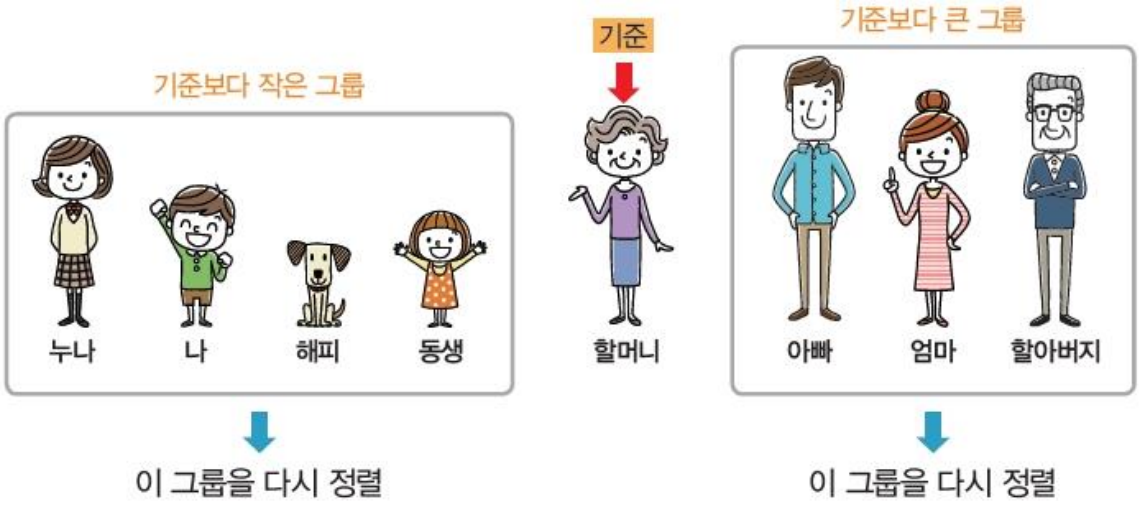
고급 정렬 알고리즘의 원리와 응용

1) 퀵 정렬(Quick Sort)

1] 퀵 정렬(Quick Sort)

퀵 정렬의 개념

기준을 하나 뽑은 후 기준보다 작은 그룹과 큰 그룹을 나누어 다시 각 그룹으로 정렬하는 방법



퀵 정렬의 구현

가족을 정렬하는 경우

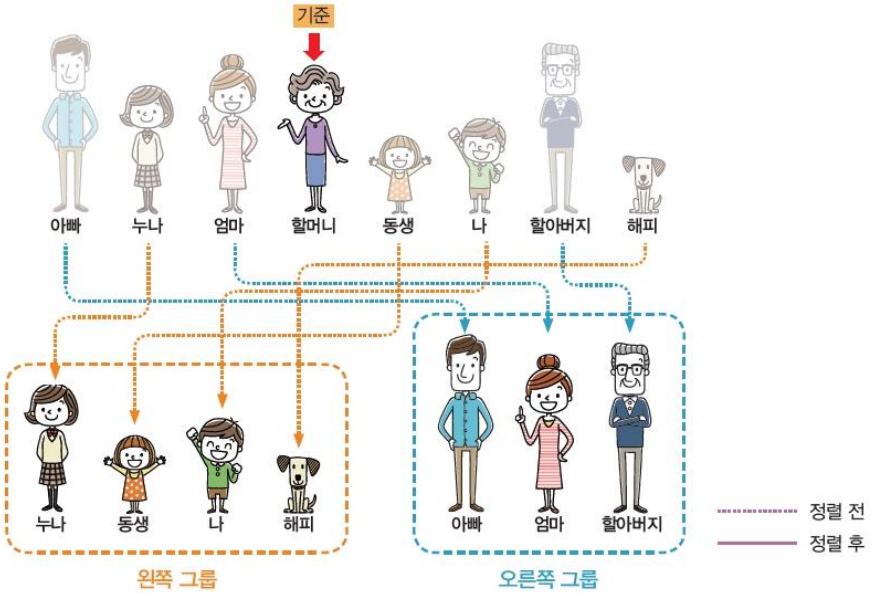
- 일반적으로 중간에 위치한 데이터를 기준으로 선정
- 퀵 정렬 전 초기 상태

1] 퀵 정렬(Quick Sort)

퀵 정렬의 구현

1 할머니를 기준으로 선정한 후 할머니보다 작은 가족은 왼쪽 그룹, 큰 가족은 오른쪽 그룹으로 보냄

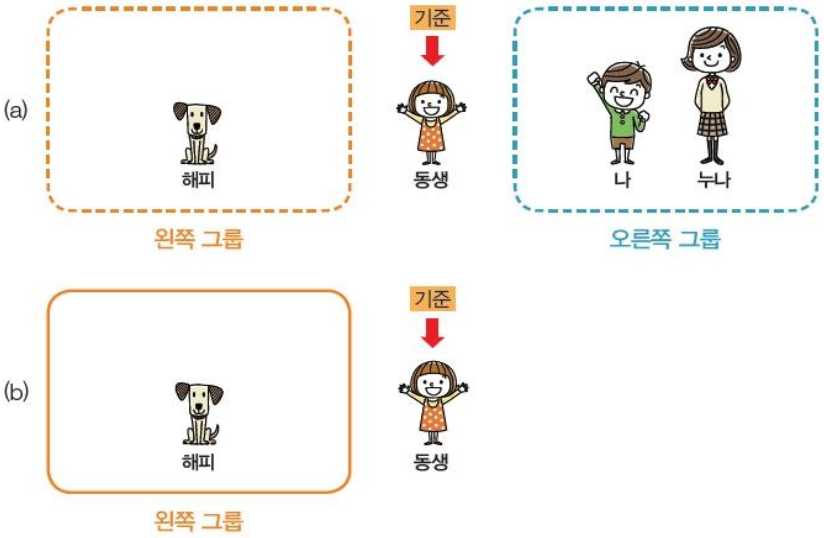
- 퀵 정렬 1단계



퀵 정렬의 구현

2 나뉜 두 그룹에서 왼쪽 그룹 정렬 예

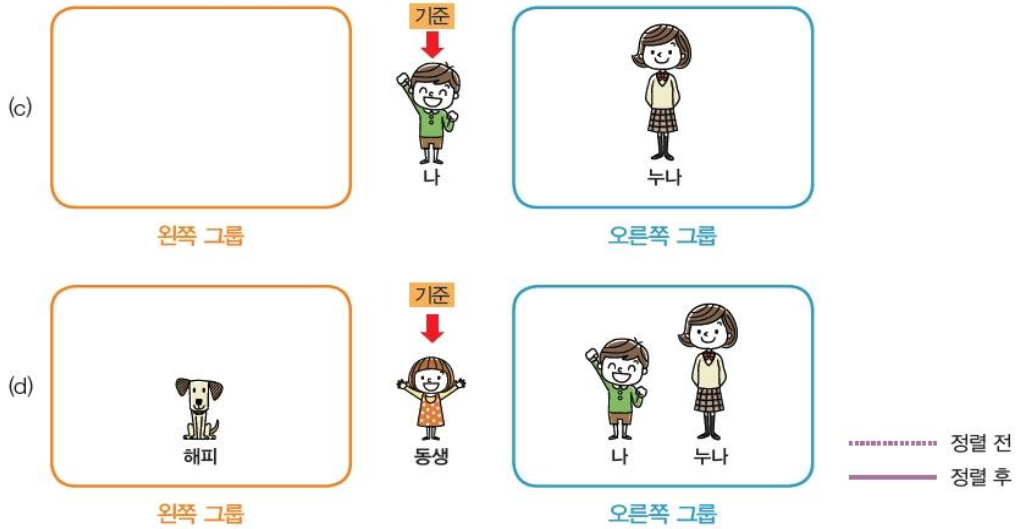
- 퀵 정렬 2단계 : 왼쪽 그룹 정렬



퀵 정렬의 구현

2 나뉜 두 그룹에서 왼쪽 그룹 정렬 예

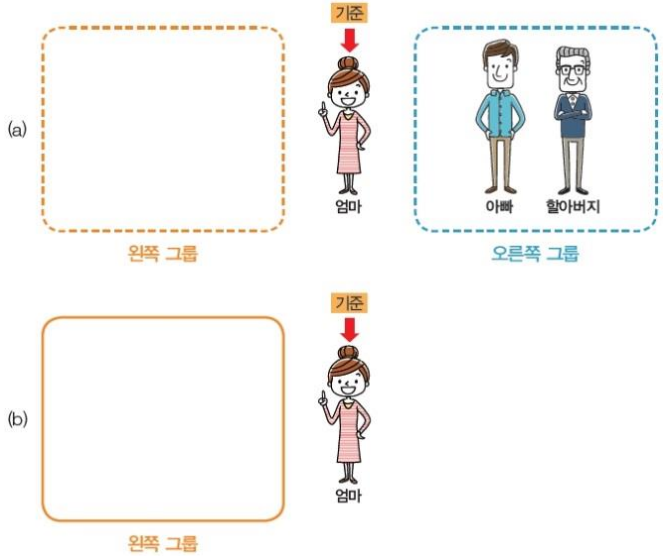
- 퀵 정렬 2단계 : 왼쪽 그룹 정렬



퀵 정렬의 구현

3 나뉜 두 그룹에서 오른쪽 그룹 정렬 예

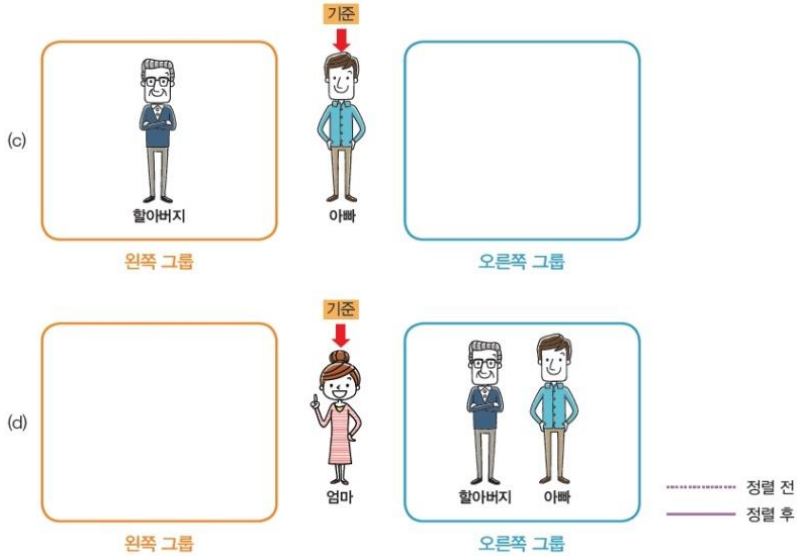
- 퀵 정렬 3단계 : 오른쪽 그룹 정렬



퀵 정렬의 구현

3 나뉜 두 그룹에서 오른쪽 그룹 정렬 예

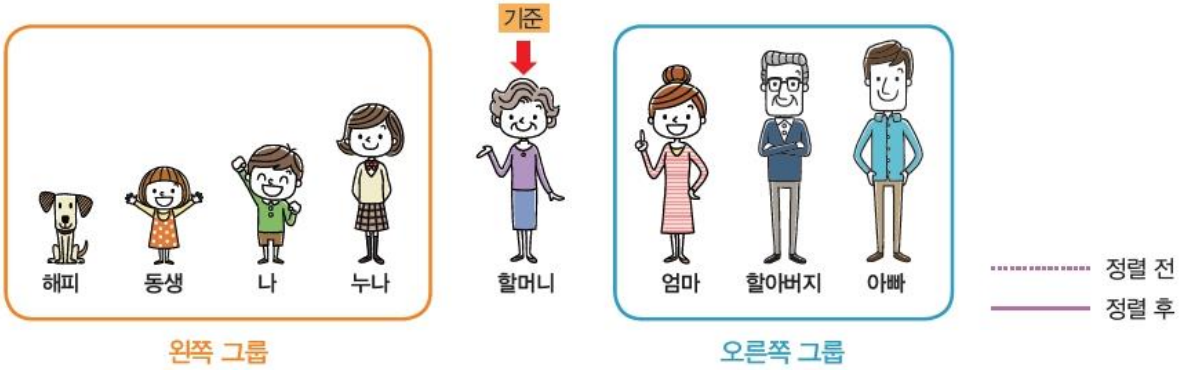
- 퀵 정렬 3단계 : 오른쪽 그룹 정렬



퀵 정렬의 구현

4 전체 데이터가 모두 정렬된 최종 상태

- 퀵 정렬 완료



퀵 정렬의 구현

퀵 정렬의 간단한 구현

```
## 클래스와 함수 선언 부분 ##
def quickSort(ary) :

    n = len(ary)
    if n <= 1 :                 # 정렬할 리스트의 개수가 1개 이하면
        return ary

    pivot = ary[n // 2]         # 기준 값을 중간값으로 지정
    leftAry, rightAry = [], []

    for num in ary :
        if num < pivot :
            leftAry.append(num)
        elif num > pivot :
            rightAry.append(num)
```

퀵 정렬의 구현

퀵 정렬의 간단한 구현

```
17      return quickSort(leftAry) + [pivot] + quickSort(rightAry)
18
19  ## 전역 변수 선언 부분 ##
20  dataAry = [188, 150, 168, 162, 105, 120, 177, 50]
21
22  ## 메인 코드 부분 ##
23  print('정렬 전 -->', dataAry)
24  dataAry = quickSort(dataAry)
25  print('정렬 후 -->', dataAry)
```

실행 결과

```
정렬 전 --> [188, 150, 168, 162, 105, 120, 177, 50]
정렬 후 --> [50, 105, 120, 150, 162, 168, 177, 188]
```

퀵 정렬 실습

> 앞쪽의 소스를 수정해서 랜덤하게 0과 200 사이의 숫자 20개를 생성한 후 내림차순으로 정렬하도록 코드를 작성하자.
> 그리고 몇 번 만에 정렬했는지 횟수를 출력하자.

실행 결과

```
정렬 전 --> [31, 49, 175, 33, 103, 76, 63, 151, 166, 25]
정렬 후 --> [175, 166, 151, 103, 76, 63, 49, 33, 31, 25]
## 25회로 정렬 완료
```

퀵 정렬 실습

중복 값을 고려한 퀵 정렬

퀵 정렬의 간단한 구현(중복된 값을 고려)

```
1  ## 클래스와 함수 선언 부분 ##
2  def quickSort(ary) :
3      n = len(ary)
4      if n <= 1 :                  # 정렬할 리스트 개수가 1개 이하면
5          return ary
6
7      pivot = ary[n // 2]          # 기준 값을 중간 값으로 지정
8      leftAry, midAry, rightAry = [], [], []
9
10     for num in ary :
11         if num < pivot :
12             leftAry.append(num)
13         elif num > pivot :
14             rightAry.append(num)
15         else :
16             midAry.append(num)
17
```

중복 값을 고려한 퀵 정렬

퀵 정렬의 간단한 구현(중복된 값을 고려)

```
18      return quickSort(leftAry) + midAry + quickSort(rightAry)
19
20  ## 전역 변수 선언 부분 ##
21  dataAry = [120, 120, 188, 150, 168, 50, 50, 162, 105, 120,  177,  50]
22
23  ## 메인 코드 부분 ##
24  print('정렬 전 -->', dataAry)
25  dataAry = quickSort(dataAry)
26  print('정렬 후 -->', dataAry)
```

실행 결과

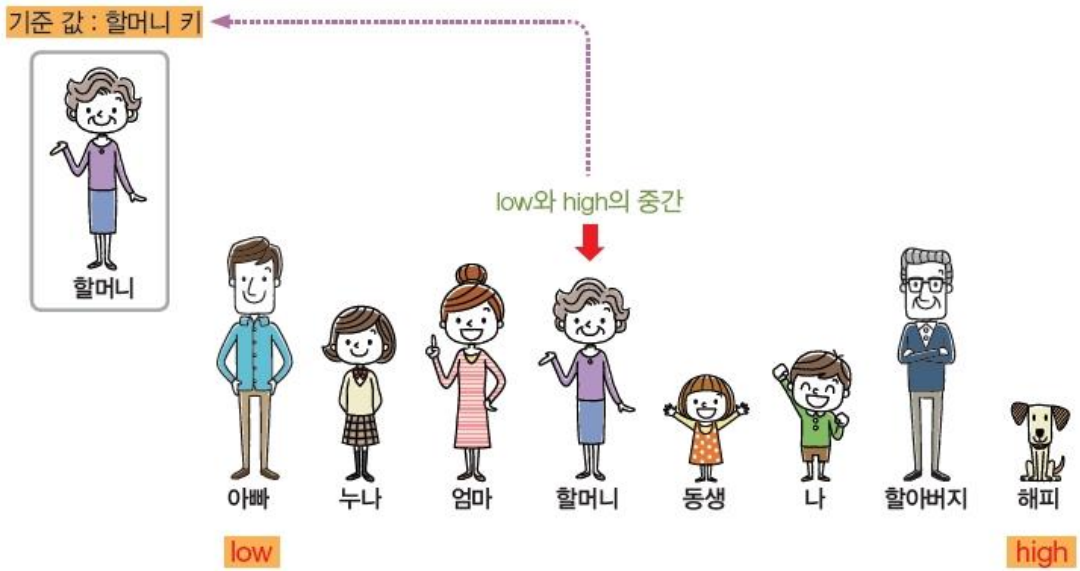
```
정렬 전 --> [120, 120, 188, 150, 168, 50, 50, 162, 105, 120, 177, 50]
정렬 후 --> [50, 50, 50, 105, 120, 120, 120, 150, 162, 168, 177, 188]
```

퀵 정렬의 일반 구현

1 start(시작점)에 있는 아빠 위치를 low로,
end(끝점)에 있는 해피 위치를 high로 두기

퀵 정렬의 일반 구현

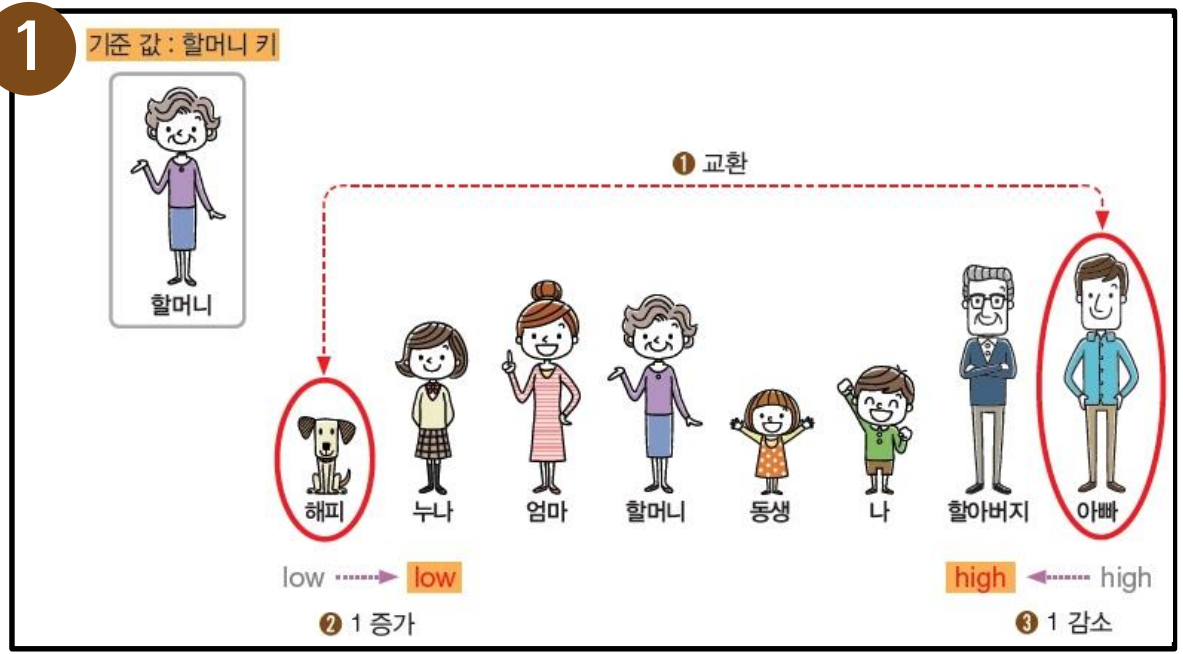
2 low와 high의 중간 위치에 있는 할머니 키를 기준 값으로 설정



1] 퀵 정렬(Quick Sort)

퀵 정렬의 일반 구현

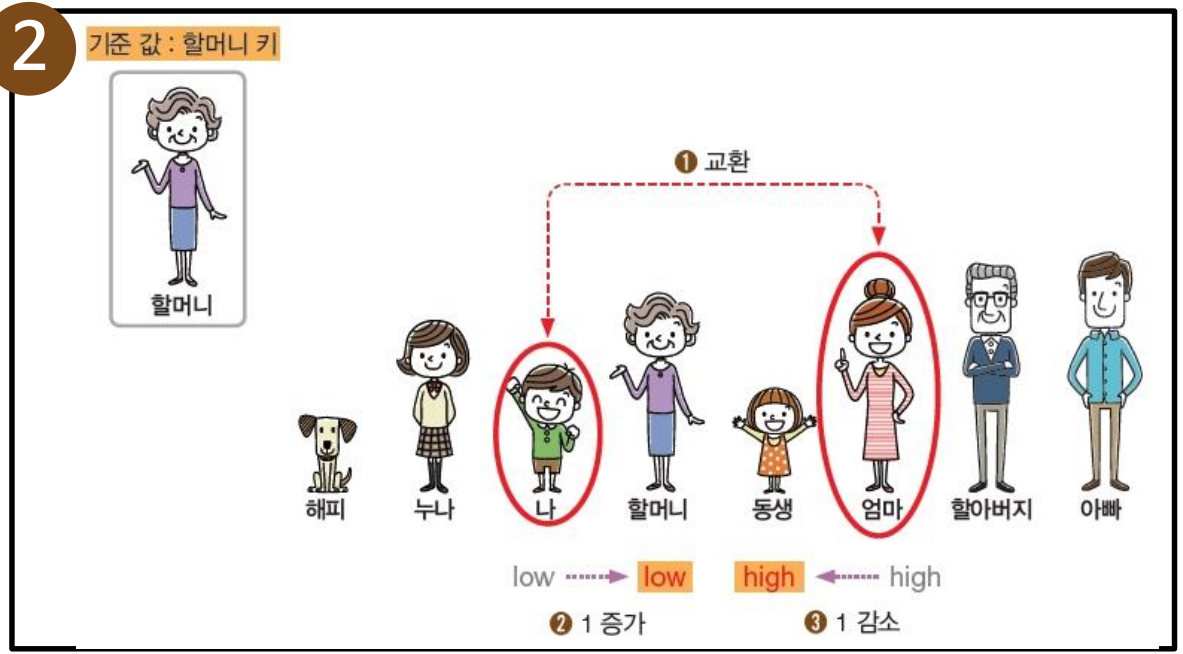
3 low가 high보다 크거나 같아질 때까지 ❶~❸과 같이 반복



퀵 정렬의 일반 구현

3 low가 high보다 크거나 같아질 때까지 ❶~❸과 같이 반복

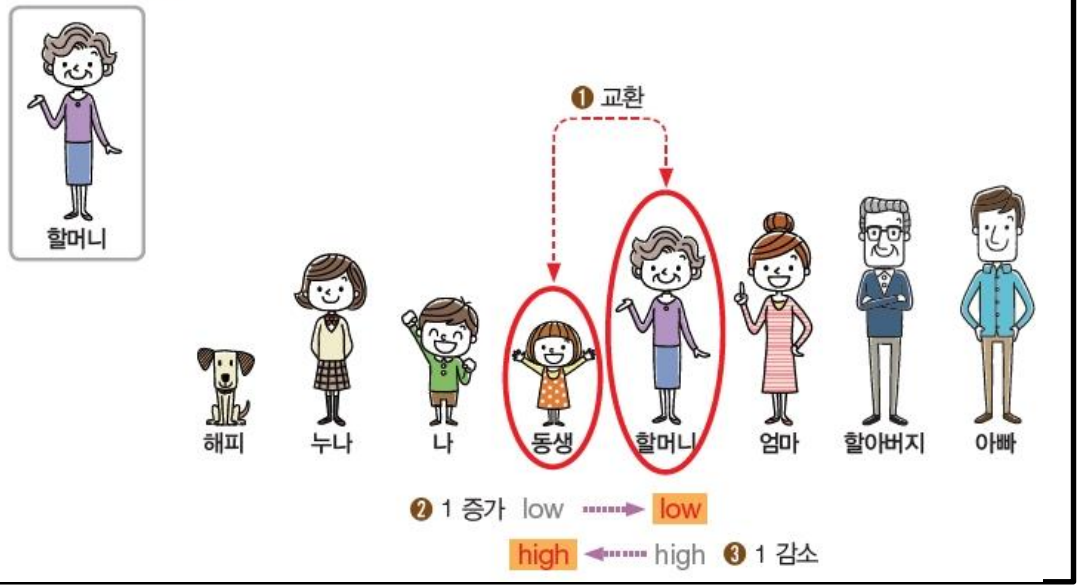
2 기준 값 : 할머니 키



퀵 정렬의 일반 구현

3 low가 high보다 크거나 같아질 때까지 ❶~❸과 같이 반복

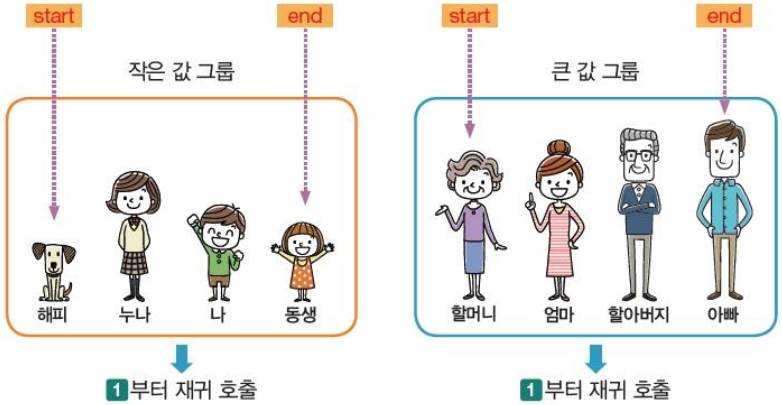
3 기준 값 : 할머니 키



퀵 정렬의 일반 구현

4 low 위치를 중심 위치(mid)로 하면 왼쪽에는 작은 값들이, 오른쪽에는 큰 값들이 들어감

- 즉, 두 그룹으로 분리됨
  → 왼쪽 작은 값 그룹과 오른쪽 큰 값 그룹에 대해
  ❶ ~ ❸을 재귀적으로 진행



퀵 정렬의 일반 구현

퀵 정렬의 일반적인 구현

```
1  ## 클래스와 함수 선언 부분 ##
2  def qSort(arr, start, end) :
3      if end <= start :
4          return
5
6  1   low = start
7      high = end
8
9    2 pivot = arr[(low + high) // 2]  # 작은 값은 왼쪽, 큰 값은 오른쪽으로 분리
10     while low <= high :
11         while arr[low] < pivot :
12             low += 1
13     3   while arr[high] > pivot :        (3의 1)
14             high -= 1
```

퀵 정렬의 일반 구현

퀵 정렬의 일반적인 구현

```
15          if low <= high :
16              arr[low], arr[high] = arr[high], arr[low]   3의 ❷ 및 ❸
17              low, high = low + 1, high - 1
18
19      mid = low
20  4
21      qSort(arr, start, mid-1)
22      qSort(arr, mid, end)
23
24  def quickSort(ary) :
25      qSort(ary, 0, len(ary)-1)
26
```

퀵 정렬의 일반 구현

퀵 정렬의 일반적인 구현

```
27 ## 전역 변수 선언 부분 ##
28 dataAry = [188, 150, 168, 162, 105, 120, 177, 50]
29
30 ## 메인 코드 부분 ##
31 print('정렬 전 -->', dataAry)
32 quickSort(dataAry)
33 print('정렬 후 -->', dataAry)
```

실행 결과

```
정렬 전 --> [188, 150, 168, 162, 105, 120, 177, 50]
정렬 후 --> [50, 105, 120, 150, 162, 168, 177, 188]
```

퀵 정렬 성능과 특이점

퀵 정렬도 가장 나쁜 경우에는 연산 수는 $O(n^2)$ 이 되지만
평균적으로 $O(n \log n)$의 연산 수를 가짐

다른 정렬에 비해서 상당히 빠른 속도이며, 특히 정렬할
데이터양이 많을수록 다른 정렬보다 매우 우수한 성능을 냄

예 n이 데이터 100만 개를 정렬한다면,
퀵 정렬은 $100만 * \log 100만 =$ 약 2000만 의
연산 횟수로 처리됨

02

고급 정렬 알고리즘의 응용

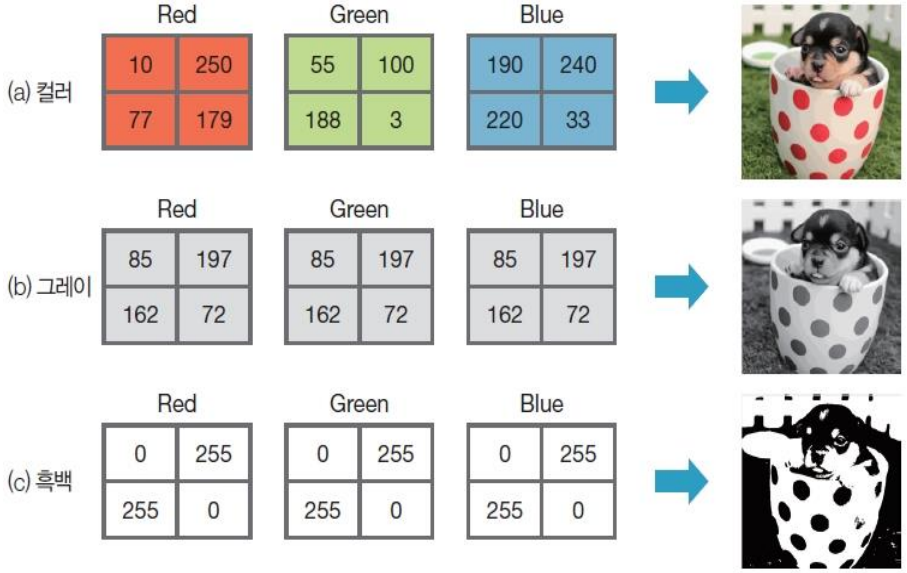
1) 컬러 이미지를 흑백 이미지로 만들기
2) 고급 정렬 알고리즘 실습

1] 컬러 이미지를 흑백 이미지로 만들기

컬러와 흑백 색상 표현 이해하기

Red, Green, Blue 색이 어떻게 조합되어 표현되는지 2×2픽셀의 이미지로 가정

- 컬러, 그레이, 흑백 이미지의 색상 표현



1] 컬러 이미지를 흑백 이미지로 만들기

파이썬으로 이미지 출력하기

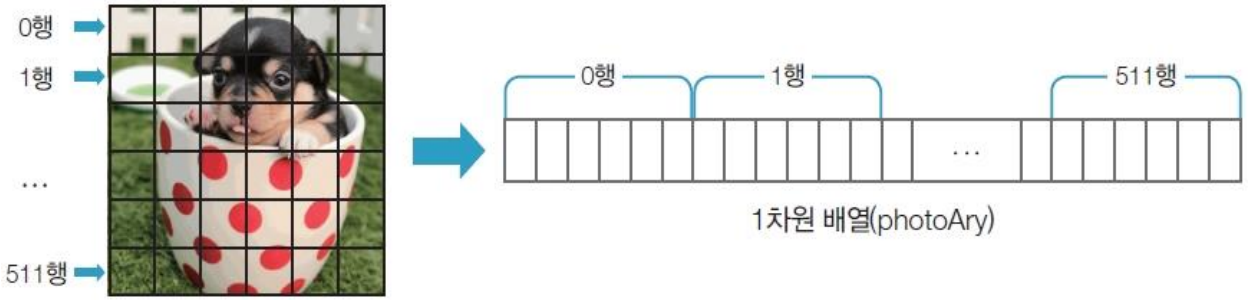
GIF 파일의 간단한 화면 출력

```
1  from tkinter import *
2
3  window = Tk()
4  window.geometry("600x600")
5
6  photo = PhotoImage(file = 'pet01.gif')
7
8  paper = Label(window, image=photo)
9  paper.pack(expand=1, anchor=CENTER)
10 window.mainloop()
```

1] 컬러 이미지를 흑백 이미지로 만들기

컬러 이미지를 1차원 배열로 만들기

2차원 이미지를 1차원 배열로 전환



1] 컬러 이미지를 흑백 이미지로 만들기

컬러 이미지를 1차원 배열로 만들기

> **1차원 배열 하나에 저장할 때는 세 점을 합한 평균으로 저장함**

```
❶ r, g, b = photo.get(i, k)
❷ value = (r + g + b) // 3
❸ photoAry.append(value)
```

1] 컬러 이미지를 흑백 이미지로 만들기

컬러 이미지를 1차원 배열로 만들기

1차원 배열 하나에 저장할 때는 세 점을 합한 평균으로 저장함

- GIF 파일을 1차원 배열로 저장

```
1  from tkinter import *
2
3  window = Tk()
4  window.geometry("600x600")
5  photo = PhotoImage(file = 'pet01.gif')
6
7  photoAry = []
8  h = photo.height()
9  w = photo.width()
10 for i in range(h) :
11     for k in range(w) :
12         r, g, b = photo.get(i, k)
13         value = (r + g + b) // 3
14         photoAry.append(value)
```

컬러 이미지를 1차원 배열로 만들기

> 1차원 배열 하나에 저장할 때는 세 점을 합한 평균으로 저장함

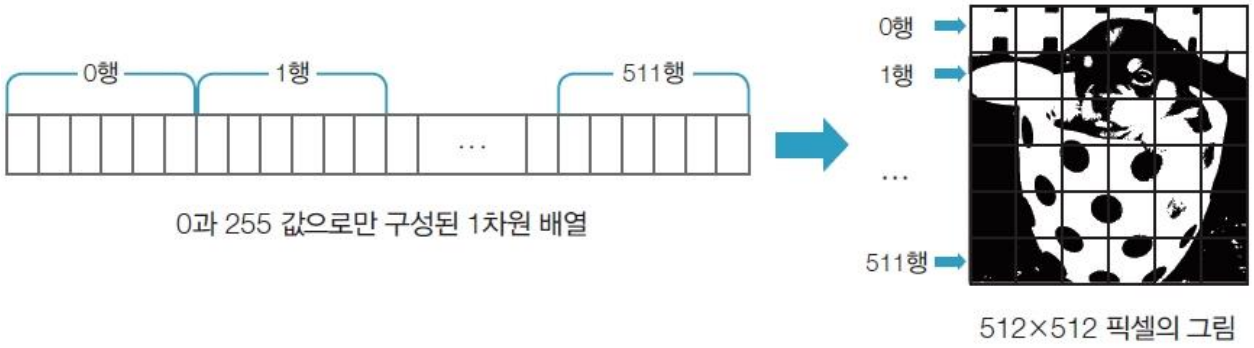
- GIF 파일을 1차원 배열로 저장

```
15
16 # 이 부분에 필요한 내용을 추가
17
18
19 paper = Label(window, image=photo)
20 paper.pack(expand=1, anchor=CENTER)
21 window.mainloop()
```

흑백 이미지로 만들기

```
if photoAry[i] <= 127 :
    photoAry[i] = 0
else :
    photoAry[i] = 255
```

1차원 배열을 2차원 이미지로 전환



흑백 이미지로 만들기

1차원 배열을 흑백 값으로 변환 후 화면 출력

```
… # 생략(      앞장의      1~14행과 동일)
15
16 for i in range(len(photoAry)) :
17     if photoAry[i] <= 127 :
18         photoAry[i] = 0
19     else :
20         photoAry[i] = 255
21
22 pos = 0
23 for i in range(h) :
24     for k in range(w) :
25         r = g = b = photoAry[pos]
```

흑백 이미지로 만들기

1차원 배열을 흑백 값으로 변환 후 화면 출력

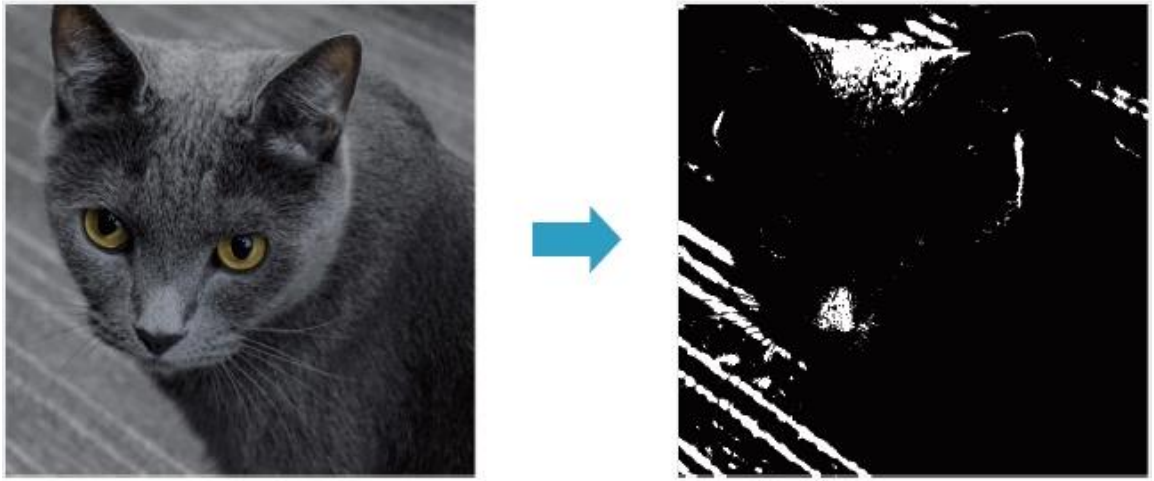
```
26            pos += 1
27            photo.put("#%02x%02x%02x" % (r, g, b), (i, k))
28
29 paper = Label(window, image=photo)
30 paper.pack(expand=1, anchor=CENTER)
31 window.mainloop()
```

1] 컬러 이미지를 흑백 이미지로 만들기

퀵 정렬로 중앙값 계산하기

어두운 이미지를 기준 값 127로 정해서 변환한 예

- 어두운 이미지를 127을 기준으로 흑백으로 변환한 예



퀵 정렬로 중앙값 계산하기

어두운 이미지를 기준 값 127로 정해서 변환한 예

- 퀵 정렬을 이용한 중앙값을 찾아서 처리

```
1  from tkinter import *
2
3  ## 클래스와 함수 선언 부분 ##
4  def qSort(arr, start, end) :
5      if end <= start :
6          return
7
8      low = start
9      high = end
10
```

퀵 정렬로 중앙값 계산하기

어두운 이미지를 기준 값 127로 정해서 변환한 예

- 퀵 정렬을 이용한 중앙값을 찾아서 처리

```
    pivot = arr[(low + high) // 2]
    while low <= high :
        while arr[low] < pivot :
            low += 1
        while arr[high] > pivot :
            high -= 1
        if low <= high :
            arr[low], arr[high] = arr[high], arr[low]
            low, high = low + 1, high - 1

    mid = low
```

작은 값은 왼쪽, 큰 값은 오른쪽으로 분리

퀵 정렬로 중앙값 계산하기

어두운 이미지를 기준 값 127로 정해서 변환한 예

- 퀵 정렬을 이용한 중앙값을 찾아서 처리

```
22
23      qSort(arr, start, mid-1)
24      qSort(arr, mid, end)
25
26  def quickSort(ary) :
27      qSort(ary, 0, len(ary)-1)
28
29  ## 메인 코드 부분 ##
30  window = Tk()
31  window.geometry("600x600")
32  photo = PhotoImage(file = 'pet02.gif')
33
```

퀵 정렬로 중앙값 계산하기

어두운 이미지를 기준 값 127로 정해서 변환한 예

- 퀵 정렬을 이용한 중앙값을 찾아서 처리

```
34 photoAry = []
35 h = photo.height()
36 w = photo.width()
37 for i in range(h) :
38     for k in range(w) :
39         r, g, b = photo.get(i, k)
40          value = (r + g + b) // 3
41          photoAry.append(value)
42
43 dataAry = photoAry[:]
44 quickSort(dataAry)
45 midValue = dataAry[h*w // 2]
```

퀵 정렬로 중앙값 계산하기

어두운 이미지를 기준 값 127로 정해서 변환한 예

- 퀵 정렬을 이용한 중앙값을 찾아서 처리

```
46
47 for i in range(len(photoAry)) :
48     if photoAry[i] <= midValue :
49         photoAry[i] = 0
50     else :
51         photoAry[i] = 255
52
53 pos = 0
54 for i in range(h) :
```

퀵 정렬로 중앙값 계산하기

어두운 이미지를 기준 값 127로 정해서 변환한 예

- 퀵 정렬을 이용한 중앙값을 찾아서 처리

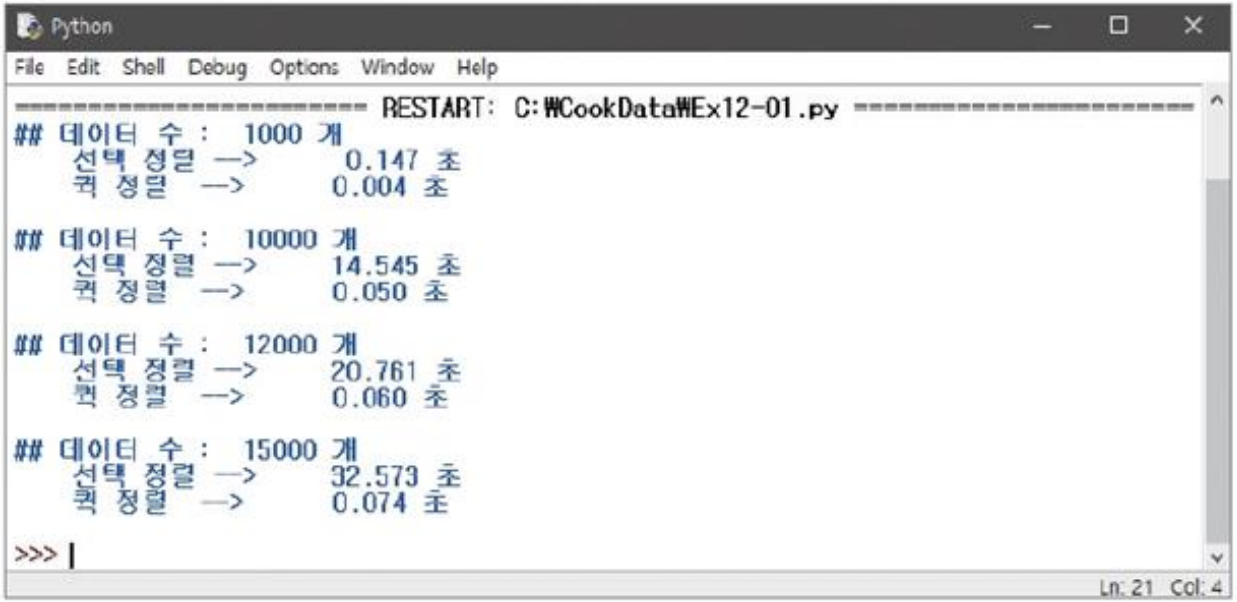
```
55      for k in range(w) :
56          r = g = b = photoAry[pos]
57          pos += 1
58          photo.put("#%02x%02x%02x" % (r, g, b), (i, k))
59
60  paper = Label(window, image=photo)
61  paper.pack(expand=1, anchor=CENTER)
62  window.mainloop()
```

고급 정렬 알고리즘 실습

선택 정렬은 $O(n^2)$의 연산 횟수를, 퀵 정렬은 평균 $O(n\ logn)$의 연산 횟수를 갖는다. 정렬할 데이터양에 두 정렬 방식의 시간 차이를 비교해 본다. 데이터는 1000, 10000, 12000, 15000개를 정렬한다. 실행 결과는 컴퓨터의 성능에 따라서 달리 나올 수 있지만, 선택 정렬은 개수가 많아질수록 시간이 급격히 증가하는 것은 동일하게 확인할 수 있다.
퀵 정렬은 개수가 많아져도 짧은 시간에 정렬 가능하다.

고급 정렬 알고리즘 실습

실행 결과



고급 정렬 알고리즘 실습

학습평가

Q1 Q2

Q1

퀵 정렬에 대한 설명으로 거리가 먼 것은?

1. 기준을 하나 뽑은 후 기준보다 작은 그룹과 큰 그룹을 나누어 다시 각 그룹으로 정렬한다.
2. 키 순서, 이름 순서, 몸무게 순서 등을 정렬할 때 사용할 수 있다.
3. 정렬은 오름차순과 내림차순으로 정렬할 수 있다.
4. 삽입 정렬이나 선택 정렬보다 성능이 더 나쁘다.

학습평가

Q1 Q2

Q1

퀵 정렬에 대한 설명으로 거리가 먼 것은?

1. 기준을 하나 뽑은 후 기준보다 작은 그룹과 큰 그룹을 나누어 다시 각 그룹으로 정렬한다.
2. 키 순서, 이름 순서, 몸무게 순서 등을 정렬할 때 사용할 수 있다.
3. 정렬은 오름차순과 내림차순으로 정렬할 수 있다.
4. ✓ 삽입 정렬이나 선택 정렬보다 성능이 더 나쁘다.

정 답 4 삽입 정렬이나 선택 정렬보다 성능이 더 나쁘다.

해 설 퀵 정렬도 가장 나쁜 경우에는 연산 수는 $O(n^2)$ 이 되지만 평균적으로 $O(n \log n)$의 연산 수를 가집니다.

학습평가

Q1 Q2

Q2

오름차순으로 정렬하는 퀵 정렬 함수의 (1)~(2)에 적합한 코드를 작성하시오.

```
def quickSort(ary):
  n = len(ary)
  if n <=1 :
    return ary
  pivot = (    1    )
  leftAry, rightAry = [], []
  for num in ary:
    if num < pivot:
      leftAry.append(num)
    elif num > pivot:
      rightAry.append(num)
  return (    2    )
```

(1)

(2)

학습평가

Q2

오름차순으로 정렬하는 퀵 정렬 함수의 (1)~(2)에 적합한 코드를 작성하시오.

```
def quickSort(ary):
  n = len(ary)
  if n <=1 :
    return ary
  pivot = (    1    )
  leftAry, rightAry = [], []
  for num in ary:
    if num < pivot:
      leftAry.append(num)
    elif num > pivot:
      rightAry.append(num)
  return (    2    )
```

(1) ary[n // 2]

(2) quickSort(leftAry) + [pivot] + quickSort(rightAry)

학습평가

정 답

(1) ary[n // 2]
(2) quickSort(leftAry) + [pivot] + quickSort(rightAry)

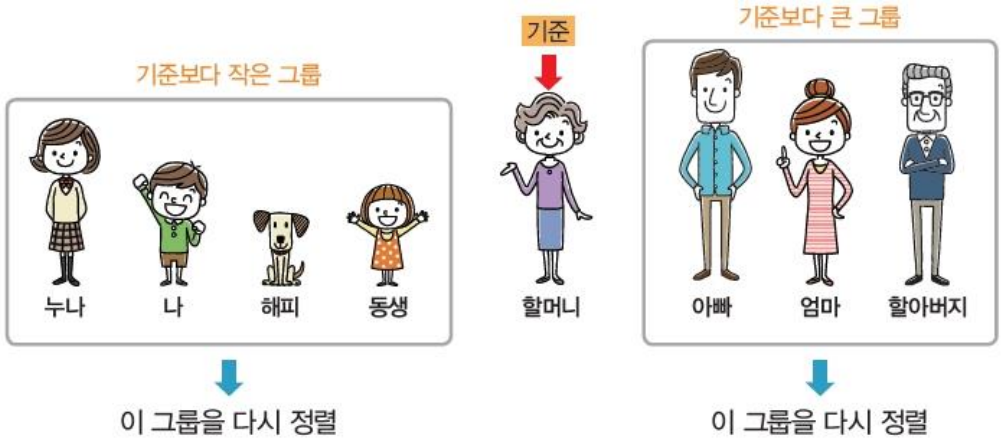
해 설

(1) 배열의 가운데 위치한 것을 기준(pivot)으로 선정합니다.
(2) 왼쪽 배열, 기준, 오른쪽 배열을 다시 재귀 호출해서 정렬합니다.

고급 정렬 알고리즘의 원리와 구현

퀵 정렬의 개념

- 기준을 하나 뽑은 후 기준보다 작은 그룹과 큰 그룹을 나누어 다시 각 그룹으로 정렬하는 방법



고급 정렬 알고리즘의 원리와 구현

퀵 정렬 성능과 특이점

- 퀵 정렬도 가장 나쁜 경우에는 연산 수는 $O(n^2)$이 되지만 평균적으로 $O(n \log n)$의 연산 수를 가짐
- 다른 정렬에 비해서 상당히 빠른 속도이며, 특히 정렬할 데이터양이 많을수록 다른 정렬보다 매우 우수한 성능을 냄

예) n이 데이터 100만 개를 정렬한다면,
퀵 정렬은 100만 × log100만 = 약 2,000만
연산 횟수로 처리됨