**자료구조 03**

****

**중앙대학교**

**자료구조 05 분반**

**20165079 김영빈**

**20160125 김소령**

**목차**

과제

[**1.** **대량의 데이터 생성하기** 3](#_Toc70263293)

[**2.** **내장된 정렬 방법으로 정렬하기** 3](#_Toc70263294)

[**3.** **힙 정렬** 6](#_Toc70263295)

[**4.** **비 재귀적 퀵 정렬** 8](#_Toc70263296)

[**5.** **재귀적 퀵 정렬** 11](#_Toc70263297)

[**6.** **선택 정렬** 13](#_Toc70263298)

[**7.** **정렬이 잘 되었는가에 대한 보증 방법** 15](#_Toc70263299)

[**8.** **종합** 17](#_Toc70263300)

# **대량의 데이터 생성하기**

*#Define Data Size*

DATA\_SIZE **=** 50000

*#Making Original Data*

*#학번 8자리 출력*

*#O(n)*

random\_id **=** random.sample(range(20130000, 20220000), DATA\_SIZE)

original\_data **=** list()

**for** r **in** random\_id:

    name **=** ''.join(random.choice(string.ascii\_uppercase) **for** \_ **in** range(10))

    phone\_number **=** "010" **+** str(format(random.randint(0, 100000000), "08"))

    original\_data.append([r, name, phone\_number])

*#Copy Original Data for Comparing Results*

selection\_sorting\_data **=** original\_data.copy()

quick\_sorting\_without\_recursive\_data **=** original\_data.copy()

quick\_sorting\_with\_recursive\_data **=** original\_data.copy()

heap\_sorting\_data **=** original\_data.copy()

과제 3에서 생성해야 하는 데이터의 조건은 다음과 같다.

* 학번: 8자리의 숫자로 랜덤하게 생성, 중복 불가
* 이름: 10자의 영문자로 랜덤하게 생성, 중복 가능
* 전화번호: 11자리의 숫자, 010으로 시작, 중복 가능

학번의 경우 random module의 sample 메서드를 사용했다. API 문서에 따르면 Random Sample은 sequence나 set, 뽑아야 하는 개수를 파라미터로 받고 그 개수만큼 sequence에서 뽑아주는 역할을 한다. Random module의 sample 메서드는 Unique 하게 원소를 뽑아준다는 점이 중요하다. 그래서 해당 작업을 하는데 시간 복잡도는 뽑은 학번이 중복되어 있는지를 생각할 필요가 없기에 O(n)이 된다. Sample 메서드에 range(20130000, 20220000)과 DATA\_SIZE를 파라미터로 전달하여 20130000부터 20219999까지의 숫자 중 겹치지 않게 50000개의 학번을 뽑도록 했다.

이름과 전화번호의 경우에는 중복이 가능하다. 따라서 alphabet을 랜덤하게 추출해서 더함으로 10 글자의 이름을 구성했다. 전화번호는 010 뒤에 8자리 숫자를 랜덤하게 생성한 뒤에 concatenate 함으로 생성했다. 최종적으로 중복되지 않는 학번, 이름, 전화번호에 해당하는 레코드 셋들이 만들어졌고 이 데이터들을 4가지의 정렬 알고리즘으로 정렬하기 위해서 .copy()를 사용하여 복사해주었다.

|  |  |  |
| --- | --- | --- |
| **Index 0** | **Index 1** | **Index 2** |
| 학번 | 이름 | 전화번호 |

|  |  |
| --- | --- |
| Selection\_sorting\_data | 선택 정렬에서 사용될 데이터 |
| Quick\_sorting\_without\_recursive\_data | 비 재귀적 퀵소트에서 사용될 데이터 |
| Quick\_sorting\_with\_recursive\_data | 재귀적 퀵소트에서 사용될 데이터 |
| Heap\_sorting\_data | 힙 소트에서 사용될 데이터 |

# **내장된 정렬 방법으로 정렬하기**

*#Sorting by Python Inherited Method*

*#Sorted는 정렬 대상이 되는 Data를 받아 정렬한 리스트를 리턴하기 때문에 기존의 데이터는 변화가 없다.*

python\_inherited\_sorting\_time **=** time.time()

python\_inherited\_sorting **=** sorted(original\_data, **key=**lambda **name**: name[1])

python\_inherited\_sorting\_time **=** time.time() **-** python\_inherited\_sorting\_time

*#Python Inherited Sorting Result*

print('\nPython inherited sorting')

**for** idx **in** range(DATA\_SIZE):

*#1000의 경우 50개의 데이터가 출력되어 2000으로 바꿔서 출력함*

**if** idx **%** 2000 **==** 0:

        print("%5d번 학생: 학번 -> %s, 이름 -> %s, 전화번호 -> %s" **%** (idx **+** 1, python\_inherited\_sorting[idx][0], python\_inherited\_sorting[idx][1], python\_inherited\_sorting[idx][2]))

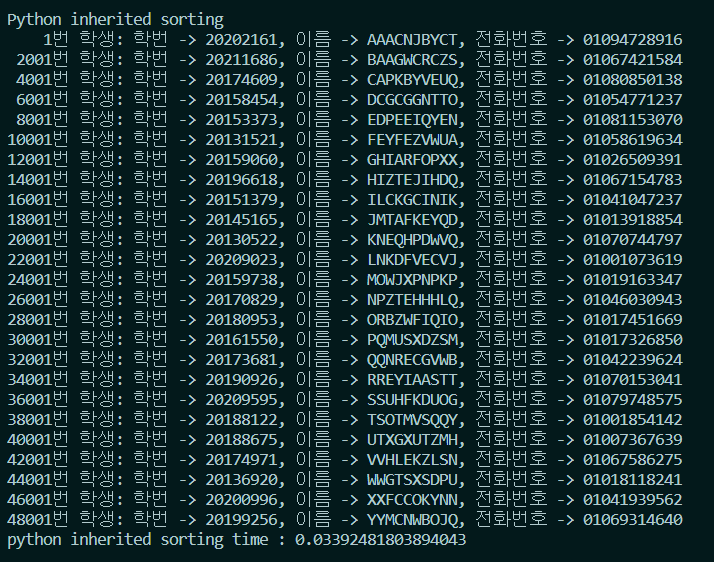
print("python inherited sorting time :", python\_inherited\_sorting\_time, **end=**"\n\n")

Python 에서는 내장된 sorting 함수가 존재한다.

* sort() 메서드는 파라미터로 전달되는 리스트를 정렬하여 준다. 기존의 리스트가 정렬된다.
* Sorted() 메서드는 파라미터로 전달되는 리스트를 정렬한 값을 return 하지만 기존의 리스트에는 변경 사항이 없다.

Assignment3에서는 sorted()를 사용했고 original data를 전달했다. 함수의 return 값을 python\_inherited\_sorting에 전달하면서 sorting된 결과가 할당되게 했고 리스트의 1번째 인덱스에 저장되어 있는 이름 값을 기준으로 정렬되도록 key의 옵션을 넣었다. 마지막으로 데이터는 2000과 modulo 연산을 이용해서 적절한 수만 출력하도록 했다. Python 내장함수의 sorting 시간은 0.03였다.

결과



# **힙 정렬**

*#Codes for HeapSort*

**def** adjust(**a**, **i**, **size**):

**while** 2 **\*** i **+** 1 **<=** size:

*#node의 왼쪽 index를 저장, root에는 root node의 index 저장*

        child **=** i **\*** 2 **+** 1

        root **=** i

*# 해당 depth에서 왼쪽과 오른쪽 child node 중 더 큰 수를 indexing 하는 역할*

**if** child **<** size **and** a[child][1] **<** a[child **+** 1][1]:

            child **+=** 1

*# 이미 root node에 가장 큰 Data가 있는 경우엔 break*

**if** a[root][1] **>=** a[child][1]:

**break**

*# 그렇지 않은 경우에는 위에서 비교했던 child node 중 큰 것과 변경*

        a[root], a[child] **=** a[child], a[root]

        i **=** child

**def** heap\_sort(**a**):

    hsize **=** len(a) **-** 1

*#initializing*

**for** i **in** reversed(range((hsize **+** 1)**//**2)):

        adjust(a, i, hsize)

*#sorting*

**for** i **in** range(hsize):

        a[0], a[hsize] **=** a[hsize], a[0]

        adjust(a, 0, hsize**-**1)

        hsize **-=** 1

**return** a

*#HeapSorting*

heap\_sorting\_time **=** time.time()

heap\_sort(heap\_sorting\_data)

heap\_sorting\_time **=** time.time() **-** heap\_sorting\_time

*#HeapSorting result*

print('\nHeap-Sorting')

**for** idx **in** range(DATA\_SIZE):

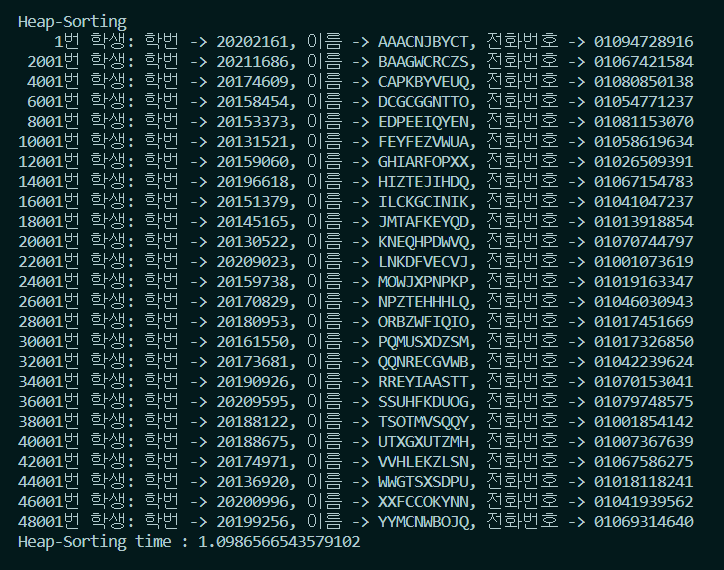
*#1000의 경우 50개의 데이터가 출력되어 2000으로 바꿔서 출력함*

**if** idx **%** 2000 **==** 0:

        print("%5d번 학생: 학번 -> %s, 이름 -> %s, 전화번호 -> %s" **%** (idx **+** 1, heap\_sorting\_data[idx][0], heap\_sorting\_data[idx][1], heap\_sorting\_data[idx][2]))

print("Heap-Sorting time :", heap\_sorting\_time, **end=**"\n\n")

결과



# **비 재귀적 퀵 정렬**

*#Stack*

class Stack:

**def** \_\_init\_\_(*self*):

*self*.items **=** []

*self*.top **=** **-**1

**def** push(*self*, **val**):

*self*.items.append(val)

**def** pop(*self*):

**try**:

**return** *self*.items.pop()

**except** IndexError:

            print("Stack is empty")

**def** \_\_len\_\_(*self*):

**return** len(*self*.items)

**def** isEmpty(*self*):

**return** *self*.\_\_len\_\_() **==** 0

**def** peak(*self*):

**try**:

**return** *self*.items[**-**1]

**except** IndexError:

            print("Stack is empty")

*#Codes for QuickSort with no Recursive*

**def** partition(**a**, **start**, **end**):

    pivot **=** a[start][1]

    left **=** start **+** 1

    right **=** end

**while** True:

**while** left **<=** right **and** a[left][1] **<=** pivot:

            left **+=** 1

**while** left **<=** right **and** a[right][1] **>** pivot:

            right **-=** 1

**if** right **<** left:

**break**

**else**:

            a[left], a[right] **=** a[right], a[left]

    a[start], a[right] **=** a[right], a[start]

**return** right

**def** quicksort\_without\_recursive(**a**):

    num\_stack **=** Stack()

    num\_stack.push(0)

    num\_stack.push(len(a) **-** 1)

**while** **not** num\_stack.isEmpty():

        end **=** num\_stack.pop()

        start **=** num\_stack.pop()

        pivot **=** partition(a, start, end)

**if** pivot **-** 1 **>** start:

            num\_stack.push(start)

            num\_stack.push(pivot **-** 1)

**if** pivot **+** 1 **<** end:

            num\_stack.push(pivot **+** 1)

            num\_stack.push(end)

**return** a

*#QuickSorting without Recursive*

quick\_sorting\_without\_recursive\_time **=** time.time()

quicksort\_without\_recursive(quick\_sorting\_without\_recursive\_data)

quick\_sorting\_without\_recursive\_time **=** time.time() **-** quick\_sorting\_without\_recursive\_time

*#QuickSorting without Recursive*

print('\nQuick-Sorting without recursive')

**for** idx **in** range(DATA\_SIZE):

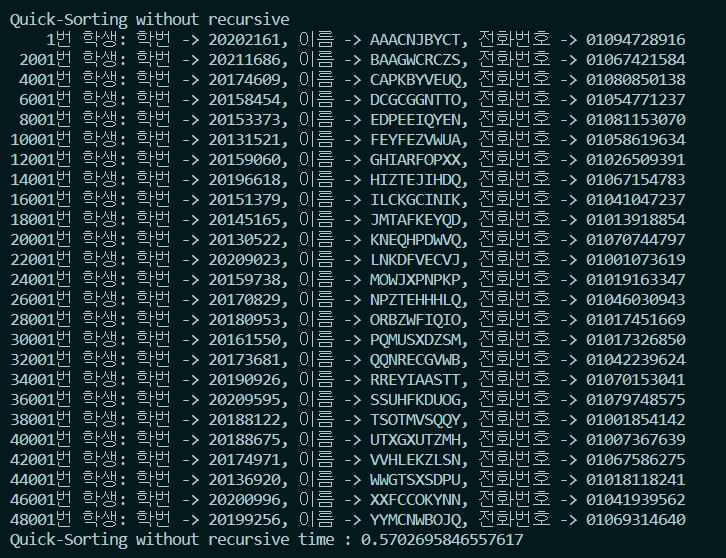
*#1000의 경우 50개의 데이터가 출력되어 2000으로 바꿔서 출력함*

**if** idx **%** 2000 **==** 0:

        print("%5d번 학생: 학번 -> %s, 이름 -> %s, 전화번호 -> %s" **%** (idx **+** 1, quick\_sorting\_without\_recursive\_data[idx][0], quick\_sorting\_without\_recursive\_data[idx][1], quick\_sorting\_without\_recursive\_data[idx][2]))

print("Quick-Sorting without recursive time :", quick\_sorting\_without\_recursive\_time, **end=**"\n\n")

결과



# **재귀적 퀵 정렬**

*#Codes for QuickSort with Recursive*

**def** quicksort\_recursive(**a**):

**if** len(a) **<=** 1:

**return** a

    pivot **=** a[0]

    tail **=** a[1:]

    left **=** [x **for** x **in** tail **if** x[1] **<=** pivot[1]]

    right **=** [x **for** x **in** tail **if** x[1] **>** pivot[1]]

**return** quicksort\_recursive(left) **+** [pivot] **+** quicksort\_recursive(right)

*#QuickSorting with Recursive*

quick\_sorting\_with\_recursive\_time **=** time.time()

quick\_sorting\_with\_recursive\_data **=** quicksort\_recursive(quick\_sorting\_with\_recursive\_data)

quick\_sorting\_with\_recursive\_time **=** time.time() **-** quick\_sorting\_with\_recursive\_time

*#QuickSorting with Recursive Result*

print('\nQuick-Sorting with recursive')

**for** idx **in** range(DATA\_SIZE):

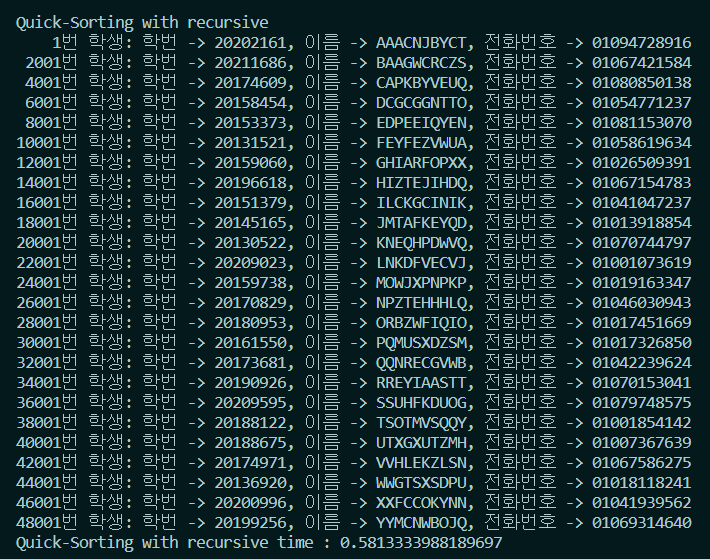
*#1000의 경우 50개의 데이터가 출력되어 2000으로 바꿔서 출력함*

**if** idx **%** 2000 **==** 0:

        print("%5d번 학생: 학번 -> %s, 이름 -> %s, 전화번호 -> %s" **%** (idx **+** 1, quick\_sorting\_with\_recursive\_data[idx][0], quick\_sorting\_with\_recursive\_data[idx][1], quick\_sorting\_with\_recursive\_data[idx][2]))

print("Quick-Sorting with recursive time :", quick\_sorting\_with\_recursive\_time, **end=**"\n\n")

결과



# **선택 정렬**

*#Codes for SelectionSort*

**def** selection\_sort(**a**):

    l **=** len(a)

**for** i **in** range(l **-** 1):

        least **=** i

**for** j **in** range(i **+** 1, l):

**if** a[j][1] **<** a[least][1]:

                least **=** j

        a[i], a[least] **=** a[least], a[i]

*#SelectionSorting*

selection\_sorting\_time **=** time.time()

selection\_sort(selection\_sorting\_data)

selection\_sorting\_time **=** time.time() **-** selection\_sorting\_time

*#SelectionSorting Result*

print('\nSelection-Sorting')

**for** idx **in** range(DATA\_SIZE):

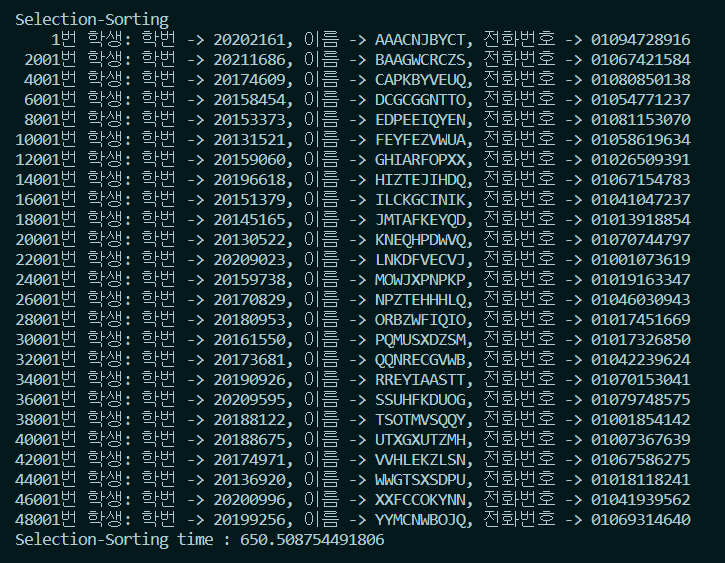
*#1000의 경우 50개의 데이터가 출력되어 2000으로 바꿔서 출력함*

**if** idx **%** 2000 **==** 0:

        print("%5d번 학생: 학번 -> %s, 이름 -> %s, 전화번호 -> %s" **%** (idx **+** 1, selection\_sorting\_data[idx][0], selection\_sorting\_data[idx][1], selection\_sorting\_data[idx][2]))

print("Selection-Sorting time :", selection\_sorting\_time, **end=**"\n\n")

결과



# **정렬이 잘 되었는가에 대한 보증 방법**

*# 정렬이 잘 되었는지 보증하는 알고리즘*

**def** is\_sorted(**target**):

**for** i **in** range(len(target) **-** 1):

**if** target[i][1] **>** target[i **+** 1][1]:

            print("not sorted well")

**return**

    print("well sorted")

*#original data의 sorting 여부*

print()

print("Original Data well sorted?")

is\_sorted(original\_data)

print()

*#python inhereted sorting 의 sorting 여부*

print()

print("Python Inherited Sorting Data well sorted?")

is\_sorted(python\_inherited\_sorting)

print()

*#heap sorting의 sorting 여부*

print()

print("Heap-Sorting Data well sorted?")

is\_sorted(heap\_sorting\_data)

print()

*#quick sorting without recursive 의 sorting 여부*

print()

print("Quick-Sorting without recursive Data well sorted?")

is\_sorted(quick\_sorting\_without\_recursive\_data)

print()

*#quick sorting with recursive 의 sorting 여부*

print()

print("Quick-Sorting with recursive Data well sorted?")

is\_sorted(quick\_sorting\_with\_recursive\_data)

print()

*#selection sorting with recursive 의 sorting 여부*

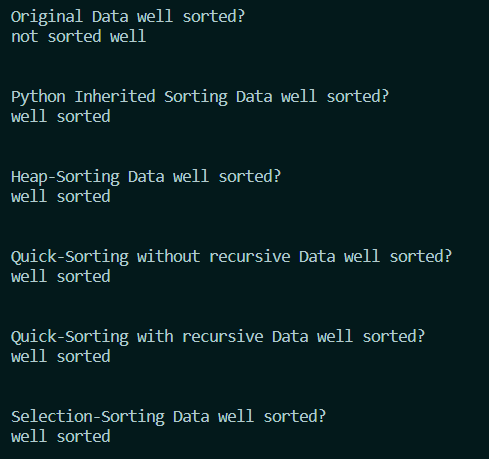
print()

print("Selection-Sorting Data well sorted?")

is\_sorted(selection\_sorting\_data)

print()

결과



# **종합**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 내장 함수 | Heap Sorting | Quick Sorting  Non-Recursive | Quick Sorting  Recursive | Selection Sort |
| 0.04 | 1.33 | 0.59 | 0.62 |  |
| Well sorted | Well sorted | Well sorted | Well sorted | Well sorted |