설계 과제 #1

B989003 권덕재

1. 설계 과제 내용

Iterative Quicksort, Recursive Quicksort, Randomized Quicksort 총 3개 버전의 퀵정렬을 구현하고 입력 데이터에 대한 비교 연산 횟수를 출력 및 비교 연산 횟수의 평균을 그래프로 나타내어 비교하기

2. 입력 데이터

알고리즘 및 그래프 출력을 테스트하기 위해 10^2, 10^4 개의 난수를 각각 30개씩 만들어서 테스트하였음. 최종 과제 시연에서 원래 과제의 조건에 따르면 10의 8승을 실행해야 하지만 시간이 너무 오래 걸리기 때문에 10의 6승까지 테스트하였음.

```
# 입력 데이터
dataSet = {}
dataSize = [10 ** (2 * (i + 1)) for i in range(3)]

# 원래 과제의 조건에 따르면 10의 8승을 실행해야 하지만
dataSize[2] = 10 ** 6 # 시간이 너무 오래 걸리기 때문에 10의 6승까지 테스트했습니다.

for i in dataSize:
    dataSet[i] = []
    for j in range(30):
        arr = [random.randint(1, i) for _ in range(i)]
        dataSet[i].append(arr)
```

3. 코드 설명

QuickSort 클래스 : 3가지 버전의 퀵정렬이 구현되어있는 클래스.

__init__(self) : 객체를 생성할 때 비교 연산의 횟수를 의미하는 self.count 변수를 0으로 초기화함.

initCount(self) : self.count를 다시 0으로 초기화하는 메서드.

plusCount(self, count) : 비교 연산이 발생할 때 비교 횟수만큼 self.count에 더하는 메서드.

popCount(self) : 비교 연산의 횟수를 return 하고 다시 0으로 초기화하는 메서드.

swap(self, arr, I, j) : arr[i]와 arr[j]의 값을 바꾸는 메서드.

partition(self, arr, left, right) : 입력 배열의 맨 왼쪽 값인 pivot을 기준으로 pivot보다 작은 값들을 왼쪽, 큰 값들을 오른쪽으로 보내고 pivot의 index를 reuturn하는 메서드.

randomizedPartition(self, arr, left, right) : left와 right 사이의 난수값 randomIdx를 pivot의 인덱스로 정하고 swap(self, arr, left, randomIdx)을 진행한 후에 self.partition 메서드를 return하는 메서드.

iterativeQuickSort(self, arr) : 기존의 quicksort와 달리 함수를 재귀적으로 호출하지 않고 while문 안에서 partition 메서드만 호출하며 정렬을 완료하는 메서드. Stack 자료구조를 사용한다.

recursiveQuickSort(self, arr, left, right) : 기본적인 quicksort, partition 메서드를 호출하여 pivot의 인덱스를 기준으로 왼쪽, 오른쪽으로 나눠 재귀적으로 함수를 호출하여 정렬을 완료하는 메서드.

randomizedQuickSort(self, arr, left, right) : pivot을 arr[left,..,right] 사이의 값 중 랜덤으로 선택하여 swap 메서드를 실행하고 recursiveQuickSort와 동일한 연산을 하는 메서드.

4. 알고리즘 테스트

3가지 버전의 퀵정렬 모두 정렬이 정상적으로 되었는지 확인하기 위한 테스트 결과

```
print('퀵정렬이 정상적으로 잘 동작하는지 확인하기 위한 테스트')
quickSortForTest = QuickSort()
arr = [random.randint(1, 10) for _ in range(10)]
print('입력 배열:', arr)

# Iterative Quicksort Test
arrCopy = arr.copy()
quickSortForTest.iterativeQuickSort(arrCopy)
print('iterativeQuickSort:', arrCopy)

# Recursive Quicksort Test
arrCopy = arr.copy()
quickSortForTest.recursiveQuickSort(arrCopy, 0, len(arr) - 1)
print('recursiveQuickSort:', arrCopy)

# Randomized Quicksort Test
arrCopy = arr.copy()
quickSortForTest.randomizedQuickSort(arrCopy, 0, len(arr) - 1)
print('randomizedQuickSort:', arrCopy)
```

```
퀵정렬이 정상적으로 잘 동작하는지 확인하기 위한 테스트
입력 배열: [2, 7, 1, 6, 1, 3, 2, 3, 8, 2]
iterativeQuickSort: [1, 1, 2, 2, 2, 3, 3, 6, 7, 8]
recursiveQuickSort: [1, 1, 2, 2, 2, 3, 3, 6, 7, 8]
randomizedQuickSort: [1, 1, 2, 2, 2, 3, 3, 6, 7, 8]
```

모든 알고리즘이 정상적으로 작동하는 것을 확인할 수 있음.

5. 시연 결과

1) 비교 연산 횟수 출력

입력 데이터의 크기가 100인 경우

	iterative	recursive	randomized
0	2183	2183	1838
1	1947	1947	1721
2	1720	1720	1744
3	1777	1777	1818
4	1728	1728	1710
5	1840	1840	1963
6	1796	1796	1775
7	1659	1659	1750
8	1778	1778	1690
9	1675	1675	1929
10	1830	1830	1845
11	1802	1802	1641
12	1748	1748	1931
13	1690	1690	1859
14	1595	1595	1800
15	1675	1675	1727
16	1698	1698	2003
17	1947	1947	1682
18	1694	1694	1679
19	1739	1739	1761
20	1714	1714	1670
21	1699	1699	1791
22	1683	1683	1712
23	1734	1734	1843
24	1770	1770	1902
25	1747	1747	1936
26	2041	2041	1825
27	1569	1569	1742
28	1749	1749	1963
29	1971	1971	1746

입력 데이터의 크기가 10000인 경우

	iterative	recursive	randomized
0	378993	378993	373239
1	361760	361760	356756
2	369448	369448	379962
3	359474	359474	374492
4	374316	374316	367659
5	381096	381096	366917
6	394657	394657	369308
7	381205	381205	380250
8	384251	384251	377496
9	366489	366489	380028
10	382012	382012	361772
11	376022	376022	375642
12	370377	370377	390176
13	412555	412555	377661
14	372902	372902	371769
15	421068	421068	389194
16	370619	370619	362901
17	406785	406785	414601
18	370588	370588	376554
19	388649	388649	382276
20	383450	383450	368179
21	370231	370231	381580
22	382111	382111	382441
23	395637	395637	369015
24	365220	365220	376244
25	379219	379219	377532
26	378705	378705	375308
27	369573	369573	365057
28	405703	405703	389945
29	369567	369567	381541

입력 데이터의 크기가 1000000인 경우

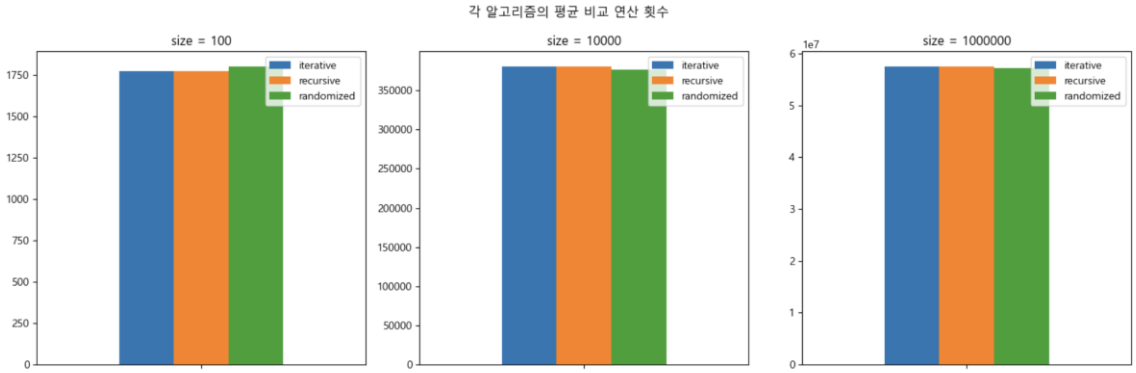
	iterative	recursive	randomized
0	58029897	58029897	56810095
1	56145233	56145233	57091424
2	59251353	59251353	58295225
3	57380387	57380387	57358950
4	56076297	56076297	58217102
5	57705872	57705872	57674187
6	58909218	58909218	57171946
7	57396265	57396265	56925422
8	59440024	59440024	57565737
9	58523461	58523461	58322055
10	59062413	59062413	56185887
11	57649405	57649405	56904434
12	56862722	56862722	55583072
13	58808808	58808808	57543901
14	56480105	56480105	58648144
15	57319598	57319598	58160584
16	56921061	56921061	57568091
17	57771883	57771883	56516214
18	58804590	58804590	57877872
19	57481708	57481708	56149952
20	56803619	56803619	56949886
21	58342508	58342508	56259591
22	55699829	55699829	57970277
23	57070836	57070836	56616563
24	57841592	57841592	57366901
25	59079295	59079295	55532788
26	56589642	56589642	57303544
27	56850352	56850352	58367231
28	55138530	55138530	57337218
29	57155254	57155254	58434663

(iterativeQuickSort와 recursiveQuickSort의 경우 비교 연산 횟수가 모두 동일한 것을 확인할 수 있음.)

2) 비교 연산 횟수의 평균 출력

size	iterative	recursive	randomized
100	1.773267e+03	1.773267e+03	1.799867e+03
10000	3.807561e+05	3.807561e+05	3.765165e+05
1000000	5.755306e+07	5.755306e+07	5.729030e+07

3) 비교 연산 횟수의 평균 그래프로 출력



6. 결과에 대한 생각

iterative quicksort는 알고리즘 동작 방식이 recursive quicksort와 동일하기 때문에 비교 연산 횟수가 같은 것을 알 수 있었습니다. 또한 iterative quicksort는 recursive quicksort와 달리 함수를 재귀적으로 호출함으로써 발생하는 오버헤드를 방지하고 메모리 관점에서도 더 효율적이라는 장점이 있지만 가독성이 떨어진다는 단점이 있습니다. 이와 달리 recursive quicksort는 가독성이 나머지 코드들보다 더 뛰어나서 해석이 쉽다는 장점이 있으며 randomized quicksort는 실행할 때마다 달라지지만 출력 결과와 그래프를 봤을 때 평균적으로 비교 연산 횟수가 나머지 2개의 quicksort 알고리즘보다 적어서 더 효율적이라는 장점이 있는 것을 확인할 수 있었습니다.

7. 느낌점

그동안 퀵정렬이 어떤 방식으로 작동하는지 추상적으로만 알고 있었는데 이번 설계 과제를 통해 퀵정렬 알고리즘의 작동 방식을 더 자세하게 알 수 있게 되었습니다. 특히 3가지 버전의 퀵정렬 중 iterative quicksort를 구현하면서 스택과 같은 자료구조를 활용하여 함수를 재귀적으로 호출하지 않고도 메모리 관점에서 더 효율적으로 정렬할 수 있다는 사실이 흥미로웠습니다. 여러모로 iterative 방식이 recursive 방식보다 장점이 많지만 가독성이 떨어지기 때문에 시기적절하게 사용해야겠다는 생각이 들었습니다.