알고리즘 분석 HW01

2019102136 최성준

*** 전체 소스 코드(**과제 (1), (2), (3)에 대한 소스코드**)**

```
import random
import time
import matplotlib.pyplot as plt

# (2)-(A) O(n^2) 인 exchange sort 알고리즘을 구현
def exchangeSort(lst):
    n = len(lst)
    temp = 0
    for i in range(0, n-1):
        for j in range(i+1, n):
            if lst[j] < lst[i]:
                temp = lst[i]
                lst[i] = lst[j]
                lst[j] = temp

# (2)-(B) O(n log n) 알고리즘인 merge sort 알고리즘을 구현
def mergeSort(lst):
    if len(lst) <= 1:
        return lst

    mid = len(lst) // 2
    left = lst[:mid]
    right = lst[mid:]
    left = mergeSort(left)
    right = mergeSort(right)
    return merge(left, right)

def merge(left, right):
    result = []
    i = 0; j = 0
    while( i < len(left) and j < len(right)):
        if left[i] <= right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    if i == len(left): result += right[j:len(right)]
    if j == len(right): result += left[i:len(left)]
    return result
```

 # (1) n개의 데이타를 random으로 생성(data = [random.randrange(1, 101) for i in range(n)]**)**
 # (3) n=2,000, 4,000, 12,000에 대해 알고리즘 A, B가 종료될 때까지의 시간을 측정한다.

```
num = [2000, 4000, 12000]
for n in num:
    dataA = [random.randrange(1, 101) for i in range(n)]
    dataB = [random.randrange(1, 101) for i in range(n)]

    Stime = time.time()
    exchangeSort(dataA)
    Ltime = time.time()
    Atime = Ltime - Stime

    Stime = time.time()
    mergeSort(dataB)
    Ltime = time.time()
    Btime = Ltime - Stime
    print(f'{Atime:10.5f} {Btime:10.5f}')
```

*** 결과**

```
  0.28133    0.01000
  1.07523    0.02104
  8.02086    0.06900
```

출력 결과는 각 행마다 랜덤으로 생성한 데이터의 개수가 2000, 4000, 12000일 때의 정렬알고리즘의 실행시간을 나타낸 것입니다. 각 랜덤 데이터에 대해서 첫번째 열은 exchange sort를 실행한 시간, 두번째 열은 merge sort를 실행한 시간을 출력하였습니다.

(4) A, B 를 1분간 수행할 때 해결할 수 있는 문제의 크기 n' 를 (3)의 결과를 이용하여 추정한다. A와 B의 수행시간과 n의 관계를 고려해서 추정할 것.

각 데이터에 대한 알고리즘 수행시간을 표로 정리하면 다음과 같다.

	실행시간	
n	Exchange Sort	Merge Sort
2000	0.28133	0.01000
4000	1.07523	0.02104
12000	8.02086	0.06900

Exchange Sort(A)의 경우 데이터의 개수가 2000개에서 4000개로 늘어남에 따라 실행시간의 변화를 살펴보면, 다음과 같다.
2000 : 4000 = 0.28133 : 1.07523
이는 n이 2배 증가할 때 실행시간은 4배가량 증가함을 보여준다.
같은 맥락으로 데이터의 개수가 4000개에서 12000개로 늘어남에 따라 실행시간의 변화를 살펴보면, 다음과 같다.
4000 : 12000 = 1.07523 : 8.02086
이는 n이 3배 증가할 때 실행시간은 8배가량 증가함을 보여준다.

이는 이론적 시간 복잡도인 O(N^2)와 유사하게 진행되는 것으로 확인할 수 있으며,
실행시간이 60초일 때는 n이 4000개일때의 실행시간의 약 60배 가량이 된다.
n^2가 60인 데이터의 개수 n을 계산해보면 7.74597이 나오므로
n' = 4000 * 7.74597 = 30983
따라서 1분간 수행할 때 해결할 수 있는 문제의 크기는 **대략 30,983개 가량**으로 추정할 수 있다.

Merge Sort(B)의 경우 데이터의 개수가 2000개에서 4000개로 늘어남에 따라 실행시간의 변화를 살펴보면, 다음과 같다.
2000 : 4000 = 0.01000 : 0.02104
이는 n이 2배 증가할 때 실행시간은 2배가량 증가함을 보여준다.
같은 맥락으로 데이터의 개수가 4000개에서 12000개로 늘어남에 따라 실행시간의 변화를 살펴보면, 다음과 같다.
4000 : 12000 = 0.02104 : 0.06900
이는 n이 3배 증가할 때 실행시간은 3.5배가량 증가함을 보여준다.

3*log2(3) = 3 * 1.58496250072이므로
이는 이론적 시간 복잡도인 O(Nlog(N))와 유사하게 진행되는 것으로 확인할 수 있으며,
실행시간이 60초일 때는 n이 2000개일때의 실행시간의 약 6000배 가량이 된다.
n*log2(n)가 6000인 데이터의 개수 n을 계산해보면 643.153이 나오므로
n' = 2000 * 643.153 = 1286306
따라서 1분간 수행할 때 해결할 수 있는 문제의 크기는 **대략 1,286,306개 가량**으로 추정할 수 있다.

(5) n=5,000만일 때의 A, B의 수행시간을 (3)의 결과를 이용하여 추정한다.

Exchange Sort(A)의 경우
n = 4000일 때 실행시간이 1초가량 걸렸으므로, 이론적 시간 복잡도인 O(N^2)에 의하여 실행시간 1초와 4,000^2 = 16,000,000가 비례한다는 것을 확인가능하다.
따라서 n=5,000만일 때의 실행시간은 다음의 비례식을 통해서 계산할 수 있다.
1 : 16,000,000 = t : 50,000,000^2
t = 50,000,000^2 * 1 / 16,000,000 = 156,250,000
따라서 **n=5,000만일 때의 A의 수행시간은 약 156,250,000초**가 걸린다.

Merge Sort(B)의 경우
n = 2000일 때 실행시간이 0.01초가량 걸렸으므로, 이론적 시간 복잡도인 O(Nlog(N))에 의하여 실행시간 0.01초와 2,000 * log(2,000) = 21,931가 비례한다는 것을 확인가능하다.
따라서 n=5,000만일 때의 실행시간은 다음의 비례식을 통해서 계산할 수 있다.
0.01 : 21,931 = t : 50,000,000 * log(50,000,000)
t = 50,000,000 * log(50,000,000) * 0.01 / 21,931
= 1,278,771,237 * 0.01 / 21,931
= 583.0884305321235
따라서 **n=5,000만일 때의 B의 수행시간은 약 583초**가 걸린다.