<알고리즘 실습> – 합병 정렬과 퀵 정렬

※ 입출력에 대한 안내

- 특별한 언급이 없으면 문제의 조건에 맞지 않는 입력은 입력되지 않는다고 가정하라.
- 특별한 언급이 없으면, 각 줄의 맨 앞과 맨 뒤에는 공백을 출력하지 않는다.
- 출력 예시에서 □는 각 줄의 맨 앞과 맨 뒤에 출력되는 공백을 의미한다.
- 입출력 예시에서 □ 이 후는 각 입력과 출력에 대한 설명이다.

[문제 1] **(합병 정렬)** **n**개의 양의 정수(중복 가능)를 입력받아 정렬하는 프로그램을 작성하시오. 정렬은 **단일연결리스트**를 이용하여 합병 정렬을 구현하여 사용한다.

◦ 구현해야 할 합병 정렬 알고리즘:

- 크기가 **n**인 단일연결리스트를 **동적 할당**하여, 입력된 양의 정수 저장(입력 정수는 중복 가능)
- **mergeSort**(**L**) 함수: 단일연결리스트 **L**의 원소들을 합병 정렬하여 정렬된 결과를 오름차순으로 정렬
- **merge**(**L1**, **L2**) 함수: **mergeSort**에 호출되어 두 개의 정렬된 단일연결리스트 **L1**과 **L2**를 합병한 하나의 단일연결리스트를 반환. **L1**과 **L2** 합병을 위해 **O**(**1**) 공간만 추가 사용 가능.
- **partition**(**L**, **k**) 함수: 단일연결리스트 **L**과 양의 정수 **k**를 전달받아 **L**을 크기가 **k**이고 |**L**| – **k**인 두 개의 부리스트 **L1**과 **L2**로 분할하여 (**L1**, **L2**)를 반환. 여기서 |**L**|은 **L**의 크기. 분할 시에도 주어진 **L** 공간 외에 **O**(**1**) 공간만 추가 사용 가능.

입력 예시 1

```
3     □ n
4 9 1
```

출력 예시 1

```
□1 4 9□ 정렬 결과
```

입력 예시 2

```
8□ n
73 65 48 31 29 20 8 3
```

출력 예시 2

```
□3 8 20 29 31 48 65 73□ 정렬 결과
```

힌트: 다음은 합병 정렬의 배열 구현 알고리즘이다. 곧바로 연결리스트로 합병 정렬을 구현하기 어려운 경우에는 아래의 배열 구현을 먼저 하고, 연결리스트 구현을 해볼 것을 권장. 아래 배열 구현에서는 **merge**에서 배열 **B**를 위해 **O**(**n**) 공간을 추가 사용하지만, 문제 **1**의 리스트 구현에서는 **O**(**1**) 공간만 추가 사용 가능하다.

```
Alg mergeSort(A)
input array A of n keys
output sorted array A

1. rMergeSort(A, 0, n – 1)
2. return

Alg rMergeSort(A, l, r)
input array A[l..r]
output sorted array A[l..r]

1. if (l < r)
m ← (l + r)/2
rMergeSort(A, l, m)
rMergeSort(A, m + 1, r)
merge(A, l, m, r)
2. return
```

```
Alg merge(A, l, m, r)
input sorted array A[l..m], A[m + 1..r]
output sorted array A[l..r] merged fromA[l..m
] and A[m + 1..r]

1. i, k ← l
2. j ← m + 1
3. while (i ≤ m & j ≤ r)
if (A[i] ≤ A[j])
B[k++] ← A[i++]
else
B[k++] ← A[j++]
4. while (i ≤ m)
B[k++] ← A[i++]
5. while (j ≤ r)
B[k++]←A[j++]
6. for k ← l to r
A[k] ← B[k]
7. return
```

[문제 2] (퀵 정렬) n개의 양의 정수(중복 가능)를 입력받아 정렬하는 프로그램을 작성하시오. 정렬은 **아래에 명시된 퀵 정렬을** 구현하여 사용한다.

◦ 구현해야 할 퀵 정렬 알고리즘:

- 크기가 n인 배열을 **동적 할당**하여, 입력된 양의 정수 저장(입력 정수는 중복 가능).
- 기준값(pivot)을 정할 때, 다음 방법을 이용한다:
  (1) 부리스트의 수들 중 1개의 위치를 무작위로 선택(즉, 입력배열의 l과 r 사이의 1개 위치를 랜덤하게 선택)하여 pivot의 위치로 정한다.
  (2) 아래 힌트처럼 pivot의 위치를 정하는 findPivot 함수를 별도 작성해서 이 함수의 반환값을 inPlacePartition 함수가 사용하는 방식을 추천한다.
- inPlacePartition의 반환값은 두 인덱스 (a, b)로, 그 의미는 분할 결과, 배열의 l번째 수부터 a – 1번째 수는 pivot보다 작은 값을, 배열의 a번째부터 b번째 수는 pivot과 같은 값을, b + 1번째부터 r번째 수는 pivot보다 큰 값을 가진다는 것이다(즉, 이후 호출되는 재귀함수는 l부터 a – 1까지 부배열에 대해서와 b + 1부터 r까지의 부배열에 대해서 다루고, pivot과 같은 값들인 a부터 b번째 값들은 재귀에서 제외된다).

입력 예시 1

```
3    □ n
4 9 1
```

출력 예시 1

```
□1 4 9□ 정렬 결과
```

입력 예시 2

```
8□ n
73 65 48 31 29 20 8 3
```

출력 예시 2

```
□3 8 20 29 31 48 65 73□ 정렬 결과
```

힌트: 다음은 제자리 퀵 정렬 알고리즘이다. 아래 내용을 참고하여 구현하되 위의 명시된 조건에 따라 pivot을 정한다.

```
Alg inPlaceQuickSort(L, l, r)
input list L, rank l, r
output list L with elements of rankfrom l to r rearranged in increasing order

1. if (l ≥ r)
return
2. k ← findPivot(L, l, r)
3. (a, b) ← inPlacePartition(L, l, r, k) {elements from a to b are same to pivot}
4. inPlaceQuickSort(L, l, a – 1) {elements from l to a - 1 are smaller than pivot}
5. inPlaceQuickSort(L, b + 1, r) {elements from b + 1 to r are bigger than pivot}
```