〈알고리즘 실습〉 - 합병정렬과 퀵정렬

※ 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) N개의 양의 정수를 입력(중복 허용)받아 정렬하는 프로그램을 작성하시오. 정렬은 **단일연결리스트**를 이용하여 합병정렬을 구현하여 사용한다.

- 구현해야할 합병 정렬 알고리즘:
  - 크기가 N인 단일연결리스트를 **동적 할당**하여, 입력된 양의 정수 저장 (입력 정수는 중복 허용)
  - mergeSort(L) 함수: 단일연결리스트 L의 원소들을 합병정렬하여 정렬된 결과를 오름차순으로 정렬
  - merge(L1, L2) 함수: mergeSort에 호출되어 두 개의 정렬된 단일연결리스트 L1과 L2를 합병한 하나의 단일연결리스트를 반환. 합병을 위해서 새로운 공간을 할당하면 안되고, L1과 L2 노드들의 링크만 변화시켜서 합병.
  - mg-partition(L, k) 함수: 단일연결리스트 L과 양의 정수 k를 입력받아서 L을 크기가 k이고 |L|-k인 두 개의 부분리스트 L1과 L2로 분할하여 (L1, L2)를 반환. 여기서 |L|은 L의 크기. 분할 시에도 추가로 공간을 할당해서 사용하지 않고, L의 공간을 그대로 사용해서 분할.

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 합병정렬을 구현하기 어려운 경우에는 아래의 배열 구현을 먼저하고, 연결리스트 구현을 해볼 것을 권장. 아래 배열 구현에서는 merge시에 B라는 보조적 배열공간을 추가적으로 만들어 사용하였으나, 문제1의 리스트구현에서는 추가로 공간을 사용하지 않는다.

```
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 from
      A[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 정수를 입력(중복 허용)받아 정렬하는 프로그램을 작성하시오. 정렬은 **아래에 명시된 퀵 정렬을** 구현하여 사용한다.

○ 구현해야할 퀵 정렬 알고리즘:

- 크기가 N인 배열을 **동적 할당**하여, 입력된 양의 정수 저장 (입력 정수는 중복 허용)
- 기준값(pivot)을 정할 때, 다음의 방법을 이용한다:
  (1) 입력된 수들 중에서 3개의 수를 랜덤하게 선택한다. (즉, 입력배열의 l번째 수부터 r번째 수 중에서 3개의 수를 랜덤하게 선택)
  (2) 랜덤하게 선택된 3개의 수 중에서 중간값(median)을 구하여 이를 pivot으로 한다.
  (3) pivot을 정하는 부분을 partition함수 내에서 처리해도 된다. 혹은, 힌트에 주어진 알고리즘에서 처럼 pivot을 정하고 그 인덱스를 반환하는 함수 find_pivot_index를 따로 작성해서 partiton에 pivot index를 인자로 넘겨줘도 된다.
- partition 함수의 반환 값은 두 인덱스인 (a,b)로 partition의 결과로, 배열의 l번째 수부터 a-1번째 수는 pivot보다 작은 값을 갖고, 배열의 a번째부터 b번째 수는 pivot과 같은 값을 갖고, b+1번째부터 r번째 수는 pivot보다 큰 값을 갖게 된다. (즉, 이후 호출되는 재귀함수는 l부터 a-1까지 배열에 대해서와 b+1부터 r까지의 배열에 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	출력 예시 2
8 ↦ N 73 65 48 31 29 20 8 3	□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 rank from l to r rearranged in increasing order

1. if (l≥ r)
      return
2. k ← find_pivot_index(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
```

Alg *inPlacePartition*(A, l, r, k)
input array $A[l..r]$ of (possibly) *duplicate* elements, index l, r, k
output a, b pair

1. $p \leftarrow A[k]$ {pivot}
2. $A[k] \leftrightarrow A[r]$ {hide pivot}
3. $i \leftarrow l$ {set i}
4. $j \leftarrow r - 1$ {set j}
5. **while** $(i \le j)$ {first run}
   while $(i \le j$ & $A[i] < p)$
   $i \leftarrow i + 1$
   while $(j \ge i$ & $A[j] \ge p)$
   $j \leftarrow j - 1$
   if $(i < j)$
   $A[i] \leftrightarrow A[j]$
6. $a \leftarrow i$ {a found}
7. $j \leftarrow r - 1$ {reset j}
8. **while** $(i \le j)$ {second run}
   while $(i \le j$ & $A[i] = p)$
   $i \leftarrow i + 1$
   while $(j \ge i$ & $A[j] > p)$
   $j \leftarrow j - 1$
   if $(i < j)$
   $A[i] \leftrightarrow A[j]$
9. $A[i] \leftrightarrow A[r]$ {replace pivot}
10. **return** a, i {a, b found}
    {Total $O(n)$}