Quick Sort 알고리즘 (by Hoare and Lomuto)

Quick Sort 알고리즘은 1959 년에 영국의 Tony Hoare 에 의하여 개발되었으며, 1961 년에 발표되었다. 이 알고리즘은 현재에도 가장 많이 사용되는 정렬 알고리즘이다. Quick Sort 는 효율적으로 구현이 된다면, Merge Sort 보다는 상당히 빠르며, Heapsort 보다는 2~3 배 빠른 것으로 알려져 있다. Quick Sort 알고리즘이 Merge Sort 보다 빠른 이유 중의 하나는, 정렬과정에서 데이터를 이동시키는 회수가 Quick Sort 알고리즘이 훨씬 적기 때문이다. 본 문제에서는 Quick Sort 알고리즘 수행 과정에서 실행되는 데이터 이동의 횟수를 구해 보고자 한다. 또한 Quick Sort 알고리즘의 점근적 분석 (asymptotic analysis)에 사용되는 비교 연산자의 실행 횟수도 구해 보고자 한다.

Quick Sort 알고리즘은 Divide-and-Conquer 전략 알고리즘으로서, 다음과 같은 과정으로 실행된다.

(1) Divide

A. Pivot 선택 : 정렬하고자 하는 배열의 가장 왼쪽 (가장 낮은 인덱스를 가지는) 원소를 pivot 으로 선택한다. (참고로 pivot 을 선택하는 방법은 여러가지 있지만, 본 문제에서는 가장 왼쪽 원소를 선택하는 방법을 사용한다.)

B. Pivot 을 제외한 나머지 배열의 원소를 pivot 보다 작은 (혹은 작거나 같은) 원소들의 그룹과 pivot 보다 큰 (혹은 크거나 같은) 원소들의 그룹으로 나눈다.

(*) Quick Sort 알고리즘에서의 divide 과정을 특별히 partition 이라고 부른다.

(2) Conquer

A. Partition 과정에서 나누어진 두 개의 원소들의 그룹에 대하여 재귀적(recursion)으로 Quick Sort 알고리즘을 수행한다.

Quick Sort 의 partion 과정에는 여러가지 방법이 있으나, 대표적으로 Tony Hoare 와 Nico Lomuto 가 개발한 partition 을 많이 사용한다.

Tony Hoare 가 개발한 Hoare Partition 을 이용한 Quick Sort 알고리즘의 pseudo code 는 다음과 같다.

```
 1  algorithm quicksort(A, low, high)
 2      if low >= high then
 3          return;
 4      p := partition_Hoare(A, low, high)
 5      quicksort(A, low, p)
 6      quicksort(A, p+1, high)
 7
 8  function partition_Hoare(A, low, high)
 9      pivot := A[low]
10      i := low - 1
11      j := high + 1
12      while true do
```

```
13          do i := i + 1 while (A[i] < pivot) // 비교(comparison) 연산자 '<'
14          do j := j - 1 while (A[j] > pivot) // 비교(comparison) 연산자 '<'
15          if i < j then
16              swap A[i] with A[j]            // swap 연산
17          else
18              return j
```

다음 그림은 9 개의 정수를 Hoare Partition 을 이용한 Quick Sort 알고리즘의 실행과정을 나타낸 예이다.

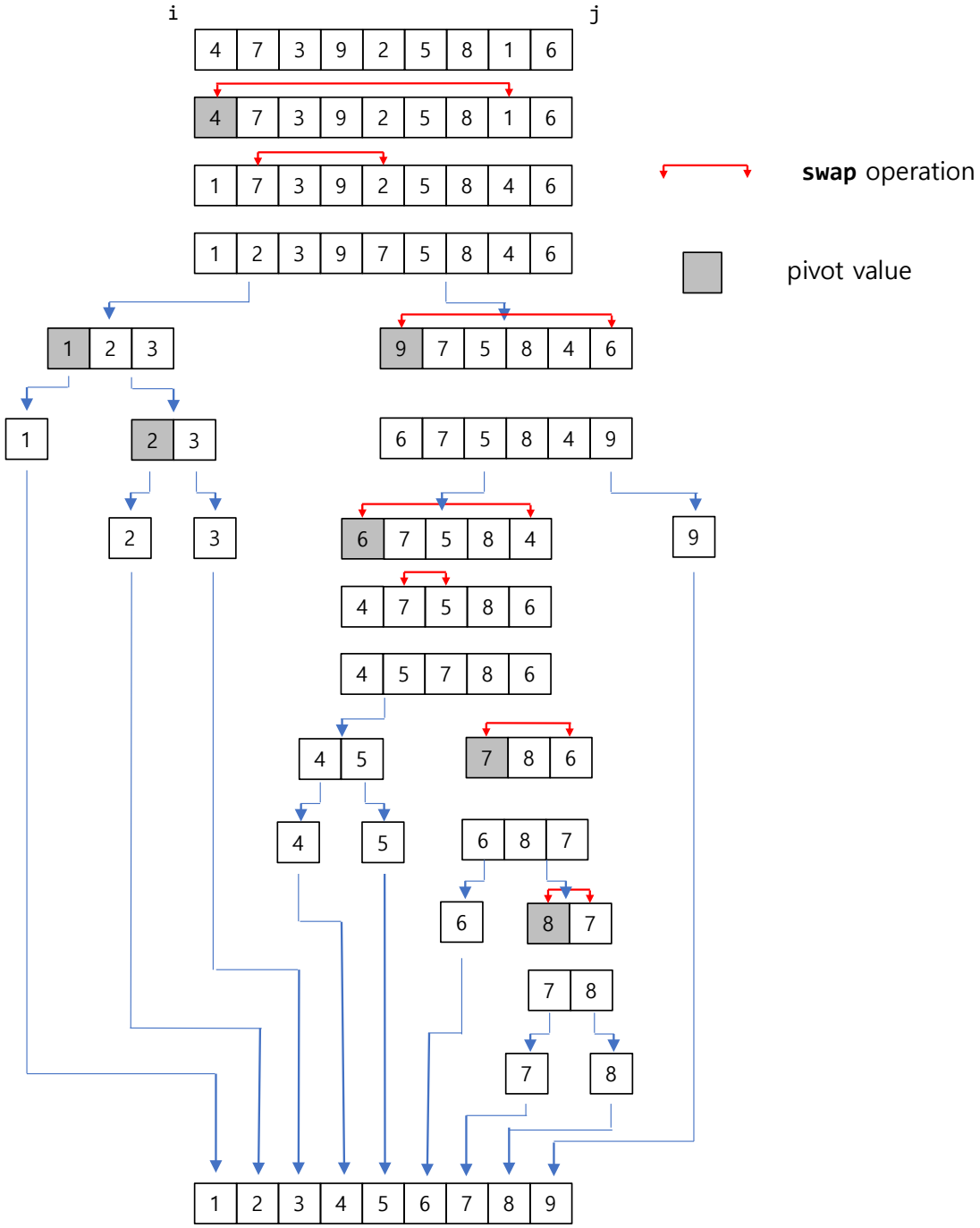

그림 1

Quick Sort 알고리즘에서 데이터들이 이동은 partition 과정에서 실행되는 swap 연산에서만 이루어진다. <그림 1>의 예는 주어진 9 개의 데이터에 대해서 Hoare partition 을 이용한 Quick Sort 를 수행하는 동안에는 총 7 번의 swap 연산 (pseudo code 의 line 16)이 수행됨을 보여준다. 같은 데이터의 경우를 Merge Sort 에서 수행하면 데이터의 이동의 회수는 훨씬 많을 것으로 예상할 수 있다.

Nico Lomuto 가 개발한 Lomuto Partition 을 이용한 Quick Sort 알고리즘의 pseudo code 는 다음과 같다.

```
 1  algorithm quicksort(A, low, high)
 2      if low >= high then
 3          return;
 4      p := partition_Hoare(A, low, high)
 5      quicksort(A, low, p-1)
 6      quicksort(A, p+1, high)
 7
 8  function partition_Lomuto(A, low, high)
 9      pivot := A[low]
10      j := low
11      for i := low+1 to high do
12          if A[i] < pivot then        // 비교(comparison) 연산자 '<'
13              j := j + 1
14              swap A[i] with A[j]     // swap 연산
15      pivot_pos := j
16      swap A[pivot_pos] with A[low]   // swap 연산
17      return pivot_pos
```

아래 <그림 2>는 9 개의 정수를 Lomuto Partition 을 이용한 Quick Sort 알고리즘의 실행과정을 나타낸 예이다. <그림 2>에서는 주어진 9 개의 데이터에 대해서 Lomuto Partition 을 이용한 Quick Sort 를 수행하는 동안에는 총 14 번의 swap 연산 (pseudo code 의 line 14, 16)이 수행됨을 보여준다.

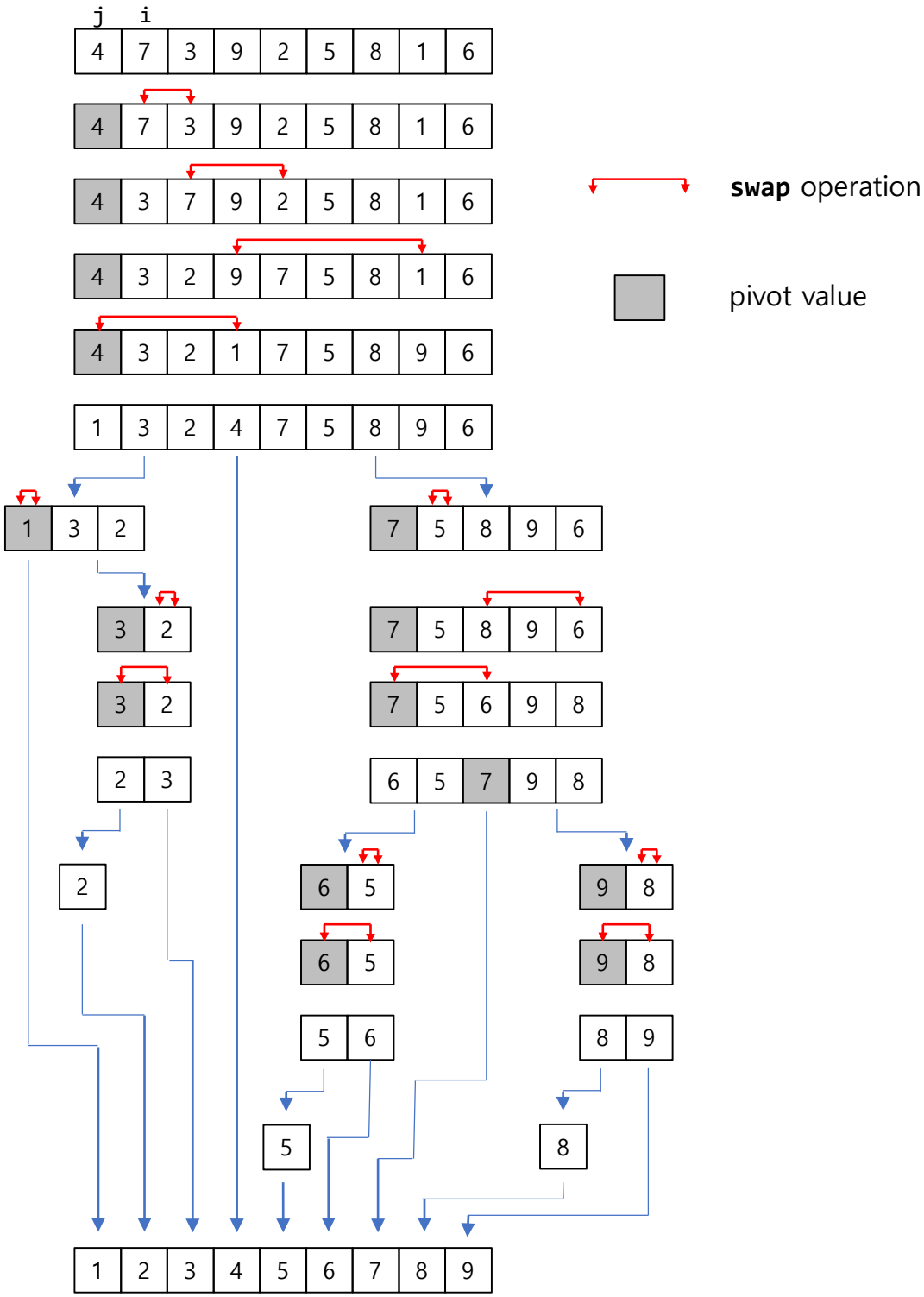

그림 2

Hoare Partition 은 Lomuto Partition 과 비교하여 다음과 같은 특징을 가지며, 따라서 상대적으로 더 효율적이라고 알려져 있다[Wiki: QuickSort, https://en.wikipedia.org/wiki/Quicksort].

(1) Hoare Partition 은 Lomuto Partition 과 비교하여 평균적으로 약 3 배 적게 swap 연산을 실행한다.

(2) 입력되는 모든 데이터가 동일한 경우에 Hoare Partition 은 나누어지는 두 그룹에 속하는 원소의 개수가 동일하도록 나누지만 (balanced partition), Lomuto Partition 은 그렇지 않다.

이번 문제를 통하여 위의 특성을 확인해 보도록 한다.

주어진 정수를 정렬하는 Quick Sort 알고리즘을 Hoare Partition 과 Lomuto Partition 을 이용하여 구현하고, 각 partition 방법을 사용할 때의 swap 연산과 비교 연산자의 실행 횟수를 구하는 프로그램을 구현하시오. 단, 위에서 제시된 pseudo-code 와 논리적으로 동일하게 구현하여야 한다. Hoare Partition 에서의 swap 연산은 pseudo-code 의 line 16 에 나타나며, 비교 연산자는 line 13, 14 에 나타난다. 또한 Lomuto Partition 에서의 swap 연산은 pseudo-code 의 line 14, 16 에 나타나며, 비교 연산자는 line 12 에 나타난다.

입력

입력은 표준입력(standard input)을 사용한다. 입력은 t 개의 테스트 케이스로 주어진다. 입력 파일의 첫 번째 줄에 테스트 케이스의 개수를 나타내는 정수 t 가 주어진다. 두 번째 줄부터 t 개의 줄에는 한 줄에 한 개의 테스트 케이스에 해당하는 정수들이 주어진다. 각 테스트 케이스에 해당되는 각 줄의 첫 번째 정수 n ($1 \leq n \leq 1{,}000$) 은 주어진 자연수의 개수를 나타낸다. 그 다음에는 n 개의 자연수가 주어지는데, 각 자연수의 최소값은 1 이며 최대값은 $2^{31}-1$ 이다. 같은 줄에 나열되는 각 정수들 사이에는 한 개의 공백이 있으며, 잘못된 데이터가 입력되는 경우는 없다.

출력

출력은 표준출력(standard output)을 사용한다. 입력되는 테스트 케이스의 순서대로 다음 줄에 이어서 각 테스트 케이스의 결과를 출력한다. 각 테스트 케이스에 해당하는 출력의 첫 줄에 입력되는 n 개의 자연수를 Hoare Partition 과 Lomuto Partition 을 이용한 Quick Sort 를 수행할 때 실행되는 swap 연산의 수를 순서대로 출력한다. 그 다음에는 Hoare Partition 과 Lomuto Partition 을 이용한 Quick Sort 를 수행할 때 실행되는 비교 연산자의 수를 순서대로 출력한다. 출력되는 4 개의 정수 사이에는 하나의 공백을 둔다.

입력과 출력의 예

입력
8 9 4 7 3 9 2 5 8 1 6 40 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 30 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 30 19 14 30 25 13 29 5 20 2 12 18 24 9 21 6 26 16 4 22 1 28 17 10 7 23 15 3 27 11 8 1000 9 9 9 9 ... <총 1000 개의 '9'> ... 9 9 9 1000 1 2 3 4 ... <1 부터 1000 까지의 정수> ... 998 999 1000 1000 1000 999 998 ... <1000 부터 1 까지의 정수> ... 4 3 2 1

출력
7 14 45 17 100 39 278 780 0 29 493 435 15 254 508 435 43 71 212 100 4932 999 11862 499500 0 999 501498 499500 500 250999 501998 499500

(참고 1) 위 테스트 데이터 파일은 별도 제공됩니다.