[I2A] - The In-Place-Partition Function

Background

In the partition problem (for the purpose of sorting),
we are given an array `A[1...n]` and a key (pivot) `k:=A[key]`, and the goal is to *rearrange* the elements of `A` such that, after rearranged,

there is an index `q` with `A[q] = k` and

- `A[i] <= A[q]` for all $1 \le i < q$,
- `A[q] < A[j]` for all $q < j \le n$.

In other words, the pivot is placed at position `q`, and

- all the elements before `q` are at most `A[q]`,
- all the elements after `q` are strictly greater than `A[q]`.

Problem Statement

Implement a function with the following prototype to do the above `Partition` operation *in place*.

```
int In_Place_Partition(int A[], int left, int right, int key);
```

Requirements:

- Let $x := A[key]$ and `q` be the value returned by the function.
  The elements in the array `A` after execution must satisfy the following criteria.
  - `A[i] <= A[q]` for all $left \le i < q$.
  - `A[q] = x`
  - `A[q] < A[j]` for all $q < j \le right$.
- The function must not declare any large temporary arrays.
  Note that, if you declare such an array, your program will exceed the memory limit and it causes a Run-Error.
- The time complexity of the function must be $O($right-left$)$.

For this problem, you can either implement the `Partition` function in Section 7.1 in the textbook, or, you can implement the following *two-pointer* method.

```
1. Swap A[key] with A[left]. // To preserve the pivot

2. // working interval [ left+1... right]
   // i and j point to the boundary of the two partitions
   i ← left+1.
   j ← right.

3. while i <= j do the following.

   a. // Extend the boundary for the left partition
      // When stopped, i points to an element not belonging to left partition
      while i <= j and A[i] <= A[left] do i++.

   b. // Extend the boundary for the right partition
      // When stopped, j points to an element not belonging to right partition
      while i <= j and A[j] > A[left] do j--.

   c. if i < j, then
      Swap A[i] with A[j] and set i++, j--.

4. Swap A[left] with A[j] and return j.
```

Submission Instructions

This is a function problem. You should submit the source code file which must contains the following problem identification:

```
/* probID: W1-A-InPlacePartition */
```

Your source file must contain the implementation of the `In_Place_Partition` function. Your source file must not include the `main` function, as this will cause a compilation error.

When submitting, select the language C - function only.

On the server side, the DomJudge system will compile and test your submitted program along with the following C program:

```
#include <stdio.h>

int In_Place_Partition(int[], int, int, int);

int A[10000000], n;
int left, right, key;

int main()
{
    scanf("%d", &n);
    for(int i=0; i<n; i++)
        scanf("%d", &A[i]);
    scanf("%d %d %d", &left, &right, &key);

    key = In_Place_Partition(A, left, right, key);
    for(int i=0; i<n; i++)
        printf("%d ", A[i]);
    printf("\n%d\n", key);

    return 0;
}
```

Example

Input:

```
5
4 3 2 1 5
0 4 2
```

Output:

```
1 2 4 3 5
1
```