Programming Assignment 2.

Divide and Conquer, Partition, K-th largest element, Nuts and Bolts Problem

Purpose of the exercise

This exercise will help you do the following:

1. Practice developing Divide and Conquer based algorithm to solve problems.
2. Practice implementing Partition algorithm and apply it to solve K-th largest element problem, sorting problem etc.

Tasks

1. Partition algorithm

A partition is an arrangement of the array's elements so that all the elements to the left of some element `A[s]` are less than or equal to `A[s]`, and all the elements to the right of `A[s]` are greater than or equal
to it:

$$\underbrace{A[0] \dots A[s-1]}_{\text{all are} \le A[s]} \quad A[s] \quad \underbrace{A[s+1] \dots A[n-1]}_{\text{all are} \ge A[s]}$$

There are many algorithms to implement the partitioning, here is the pseudocode implementing Hoare Partitioning procedure:

```
ALGORITHM HoarePartition(A[l..r])
    //Partitions a subarray by Hoare's algorithm, using the first element
    // as a pivot
    //Input: Subarray of array A[0..n – 1], defined by its left and right
    // indices l and r (l<r)
    //Output: Partition of A[l..r], with the split position returned as
    // this function's value
    p ← A[l]
    i ← l; j ← r + 1

    repeat
        repeat i ←i + 1 until A[i]≥ p
        repeat j ←j – 1 until A[j]≤ p
        swap(A[i], A[j])
    until i ≥ j

    swap(A[i], A[j]) //undo last swap when i ≥ j
    swap(A[l], A[j])
    return j
```

Your task is to implement a partition algorithm in function:

```
int partition(std::vector<int> & arr, int l, int r)
```

Notes:

1. Hoare Algorithm shown above is only for your reference. You're free to choose any other partitioning algorithm to implement. However, **using the first element** as the pivot is required.
2. Test case 0 and 1 are used for testing your implementation.

2. K-th smallest element

Given an unsorted array containing `n` positive integer numbers, and a number `k` where k `is` smaller than the size of the array, we need to find the `k`-th smallest element in the given array. For example, the 5-th smallest in the array A = [10, 18, 16, 28, 25, 32, 11, 18] is 18, since there are greater than or equal to 5 values less than or equal to it.

Your task it to design and implement an **partition** based algorithm in function `find_k`. Its prototype is as follow:

```
int find_k(std::vector<int> & nums, int l, int r, int k)
```

where `l` and `r` are begin and end of array `nums`, `k` is as `k`-th element.

Notes:

1. The only **requirement** is that using your implemented **partition** function for the `find_k` implementation.
2. Test case 2, 3, 4 are used for testing the implementation.

3. Rearrange numbers

Design an algorithm to rearrange elements of a given array of `n` integer numbers so that all its negative elements precede all its positive elements. For the given input array A = [-10, 18, -16, 28, -25, 32, 11, -18], the output could be: [-10, -18, -16, -25, 28, 32, 11, 18], or [-10, -25, -18, -16, 11, 28, 18, 32, 11] etc.

Design and implement an algorithm with time efficiency in `O(n)`.

Notes:

1. Time complexity of a sorting algorithm is `O(nlogn)` , while the order of growth of **partition** is `O(n)`.
2. Assuming that number 0 is not in the array.
3. Test case 5 is used for testing your implementation.
4. Implement the algorithm in function:

   ```
   void neg_bef_pos(std::vector<int> & nums)
   ```

4. Nuts and Bolts Matching

You are given a collection of `n` bolts of different width and `n` corresponding nuts. You are allowed to try a nut and a bolt together, from which you can determine whether the nut is larger than the bolt, smaller than the bolt, or matches the bolt exactly. However, there is no way to compare two bolts together, or two nuts together. The problem is to match each bolt to its nut.

Sample input:

```
nuts  = { ), @, *, ^, (, %, !, $, &, # }
bolts = { !, (, #, %, ), ^, &, *, $, @ }
```

After the matching, its output:

```
Nuts:  ! # $ % & ( ) * @ ^
Bolts: ! # $ % & ( ) * @ ^
```

Design and implement an algorithm for this problem with efficiency in `O(nlogn)`.

Notes:

1. The algorithm to solve the problem is again based on **partition**. However, this time, we are not allowed to compare the size of two nuts or two bolts. But we could take an element from the bolt list as a pivot.

   > 1. Taking the last element of the bolt as pivot, rearrange the nuts list and get the final position of the nut whose bolt is the pivot element.
   > 2. After partitioning the nuts list, we can partition the bolts list using the selected nut (from the result of step 1).
   > 3. The same tasks are performed for left and right sub-lists to get all matches. (recursively call the algorithm)

2. Implement a new partition algorithm in function `partition_pivot`, it uses the parameter `pivot` to rearrange the list. The prototype is as follow:

   ```
   int partition_pivot(std::vector<int> & nums, int l, int r, int pivot)
   ```

3. Implement the nuts and bolts matching in function `nuts_bolts_match()`

   ```
   void nuts_bolts_match(std::vector<int> & nuts,
                         std::vector<int> & bolts, int l, int r)
   ```

4. Test case 6 is used for testing your implementation.

5. Compile, run and test code

Compile and link the completed source file *q.cpp* using the required g++ options:

```
g++ -std=c++17 -pedantic-errors -Wall -Wextra -Werror q.cpp qdriver.cpp -o q.out
```

Then run the executable with input 0, ..., 6 respectively ：

```
./q.out >> myout0.txt
0
...
./q.out >> myout6.txt
6
```

Use the *diff* command in the Linux bash shell to compare the output files:

```
diff --strip-trailing-cr -y --suppress-common-lines myout0.txt out0.txt
...
diff --strip-trailing-cr -y --suppress-common-lines myout6.txt out6.txt
```

If *diff* completes the comparison without generating any output, then the contents of the two files are an exact match.

Submission

Once your implementation of *q.cpp* is complete, again ensure that the program works and that it contains updated **file-level** documentation comments.

```
/*!*****************************************************************************
\file    q.cpp
\author  ABC
\par     DP email: ABC@digipen.edu
\par     Course: CS330
\par     Section: A
\par     Programming Assignment #2
\date    07-07-2021

\brief

*******************************************************************************/
```

Then upload the file `q.cpp` to the submission page in Moodle and the system will auto-grade your submission.