CSCI 230 PA 5 Submission

Due Date: 03/25/2022 Late (date and time):________________

Name(s): Nero Li

Exercise 1 -- need to submit source code and I/O

-- check if completely done ✓; otherwise, discuss issues below

Source code below:
exercise_1.cpp:

```
/*  Program: PA_5_exercise_1
    Author: Nero Li
    Class: CSCI 230
    Date: 04/05/2022
    Description:
        Modify selection sort to find the kth smallest element (do
        not sort the whole list).  Output both kth smallest element
        and number of compares.  Test it with a small file with a few
        integers first like n = 10 (try k = 1, k = n/2, and k = n) and
        then try it to find the medians (k = n/2) of the two data
        files from previous PA, small1k.txt and large100k.txt.

    I certify that the code below is my own work.

      Exception(s): N/A

*/

#include <iostream>
#include <fstream>
#include <vector>

using namespace std;

int selectionSort(vector<int> vec, int k)
{
    for (int i = 0; i < k; ++i)
    {
        int min = i;
        for (int j = i; j < vec.size(); ++j)
            if (vec[min] > vec[j])
                min = j;

        if (min != i)
            swap(vec[i], vec[min]);
    }
    return vec[k - 1];
}

void smallTest()
{
    vector<int> vec = {5, 1, 9, 6, 4, 7, 8, 3, 0, 2};
    int k;

    cout << "For n = 10 test array:\n";

    k = 1;
    cout << "vec[" << k - 1 << "] = " << selectionSort(vec, k) << endl;

    k = vec.size() / 2;
    cout << "vec[" << k - 1 << "] = " << selectionSort(vec, k) << endl;

    k = vec.size();
    cout << "vec[" << k - 1 << "] = " << selectionSort(vec, k) << endl <<
endl;
}

void test(string str)
{
    vector<int> vec;
    ifstream fin;
    int k;

    fin.open(str, ios::binary);

    if(!fin)
        return;
    else
        cout << "For \"" << str << "\":\n";

    while (!fin.eof())
    {
        int n;

        fin >> n;
        vec.push_back(n);
    }

    k = vec.size() / 2;
    cout << "vec[" << k - 1 << "] = " << selectionSort(vec, k) << endl <<
endl;
}

int main()
{
    smallTest();
    test("small1k.txt");
    test("large100k.txt");

    cout << "Author: Nero Li\n";

    return 0;
}
```

Input/output below:

```
For n = 10 test array:
vec[0] = 0
vec[4] = 4
vec[9] = 9
For "small1k.txt":
vec[499] = 4235

For "large100k.txt":
vec[49999] = 50000

Author: Nero Li
```

Exercise 2 (with extra credit) -- need to submit source code and I/O

 -- check if completely done _✓_; otherwise, discuss issues below

Source code below:
exercise_2.cpp:

```
/*  Program: PA_5_exercise_2
    Author: Nero Li
    Class: CSCI 230
    Date: 04/05/2022
    Description:
        Implement the randomized quick select algorithm to find kth
        smallest element as discussed in lecture/book.  Output both kth
        smallest element and number of compares.  Test it with a small
        file with a few integers first like n = 10 (try k = 1, k = n/2,
        and k = n) and then try it to find the medians (k = n/2) of the
        two data files from previous PA, small1k.txt and large100k.txt.

    I certify that the code below is my own work.

      Exception(s): N/A

*/

#include <iostream>
#include <fstream>
#include <vector>
#include <chrono>
#include <ctime>
#include <string>
```

```
using namespace std;

int depth;

int quickSelect(vector<int> S, int k, int deep = 0)
{
    if (deep > depth)
        depth = deep;

    if (S.size() == 1)
        return S[0];

    int pivot = rand() % S.size();

    vector<int> L;
    vector<int> E;
    vector<int> G;

    for (int i = 0; i < S.size(); ++i)
    {
        if (S[i] < S[pivot])
            L.push_back(S[i]);
        else if (S[i] == S[pivot])
            E.push_back(S[i]);
        else
            G.push_back(S[i]);
    }

    if (k <= L.size())
        return quickSelect(L, k, deep + 1);
    else if (k <= (L.size() + E.size()))
        return S[pivot];
    else
        return quickSelect(G, k - L.size() - E.size(), deep + 1);
}

void smallTest()
{
    vector<int> vec = {5, 1, 9, 6, 4, 7, 8, 3, 0, 2};
    int k;

    cout << "For n = 10 test array:\n";

    k = 1;
    depth = 0;
    cout << "vec[" << k - 1 << "] = " << quickSelect(vec, k) << endl;
    cout << "Depth: " << depth << endl << endl;

    k = vec.size() / 2;
    depth = 0;
    cout << "vec[" << k - 1 << "] = " << quickSelect(vec, k) << endl;
    cout << "Depth: " << depth << endl << endl;

    k = vec.size();
    depth = 0;
    cout << "vec[" << k - 1 << "] = " << quickSelect(vec, k) << endl;
    cout << "Depth: " << depth << endl << endl;
}

void test(string str)
{
    vector<int> vec;
    ifstream fin;
    int k;

    fin.open(str, ios::binary);

    if(!fin)
        return;
    else
        cout << "For \"" << str << "\":\n";

    while (!fin.eof())
    {
        int n;

        fin >> n;
        vec.push_back(n);
    }

    depth = 0;
    k = vec.size() / 2;
    auto start = chrono::high_resolution_clock::now();
    cout << "vec[" << k - 1 << "] = " << quickSelect(vec, k) << endl;
    auto end = chrono::high_resolution_clock::now();
    cout << "Depth: " << depth << endl;
    cout << "Time: " << chrono::duration_cast<chrono::nanoseconds>(end -
start).count() << " ns" << endl << endl;
}

int main()
{
    srand(time(NULL));

    smallTest();
    test("small1k.txt");
    test("large100k.txt");

    cout << "Author: Nero Li\n";

    return 0;
}
```

Input/output below:

```
For n = 10 test array:
vec[0] = 0
Depth: 1

vec[4] = 4
Depth: 2

vec[9] = 9
Depth: 2

For "small1k.txt":
vec[499] = 4235
Depth: 8
Time: 1000900 ns

For "large100k.txt":
vec[49999] = 50000
Depth: 22
Time: 14012500 ns

Author: Nero Li
```

Answer for Question 1:

The first reason is if we just want to output a single element and do not need to care about all the other elements in an array, then we can use a selection algorithm to reduce the work we have to do to find the median.

The second reason is if we can quickly find the median for our whole array, we can easily divide it into two separate groups, and that situation is what our quick sort is like. In other words, it will help us easier to find the greatest pivot and then make our quick sort more stable.

Answer for Question 2:

The most important idea is that we need to "prune" all the data we have into several groups, usually three groups, and then "search" all the sub-groups to modify our next steps. So, for all patterns followed by prune and search, we should have two parts to do each of the work.

Although most of the work we have done with recursion, it is possible to do the work with several loops instead of recursion, and sometimes it might save more time and space to finish the work. It is just harder to think about the actual code.