CS 201, Fall 2022
Homework Assignment 2

Due: 23:59, November 23, 2022

In this homework, we will study the problem of finding the "median" of an array of `n` integers. The median of an array of integers is the middle element of the array if the numbers in the array are sorted. For example, consider the array `A = [ 3, 13, 39, 24, 14, 12, 56, 23, 29 ]`. If we put those numbers in order, we would have `A' = [ 3, 12, 13, 14, 23, 24, 29, 39, 56 ]`. There are nine numbers. Our middle number will be the fifth number. The median of `A` is 23. If the size of the array is even, there are two medians, the "lower-median" and the "upper-median". For example, assume `A = [ 3, 13, 39, 24, 14, 12, 56, 23, 29, 11 ]`. Then, `A' = [ 3, 11, 12, 13, 14, 23, 24, 29, 39, 56 ]`. There are now ten numbers so we do not have just one middle number, we have a pair of middle numbers: 14 and 23. 14 is the lower-median and 23 is the upper-median. For this homework you can assume that for inputs with even number of elements, the lower-median is the median.

Three alternative algorithms that find the median of an array are discussed below. Each algorithm has a different time complexity. The goal of this homework is to simulate the growth rates of all three algorithms using different inputs.

Algorithm 1 Finding the maximum number in an array can be done in $O(n)$ time. We can simply find the maximum value in the array, eliminate it (move it to the beginning of the array), and continue doing this $n/2$ times. Hence, this algorithm runs in $O(n^2)$-time. A pseudocode of this algorithm is given below:

```
1: procedure FIND_MEDIAN_1(A, size)
2:     count ← 0
3:     while count < (size+1)/2 do
4:         largest ← A[count]
5:         indexOfLargest ← count
6:         for i ← count to size-1 do
7:             if A[i] > largest then
8:                 largest ← A[i]
9:                 indexOfLargest ← i
10:            end if
11:        end for
12:        exchange A[count] ← A[indexOfLargest]
13:        count ← count + 1
14:    end while
15:    return A[count-1]
16: end procedure
```

Algorithm 2 You can sort the input array in descending order, and select the middle element (that is the median) in the sorted array. The overall complexity of the algorithm will be dominated by the complexity of the sorting algorithm because median can be selected in $O(1)$-time once the array is sorted. You can use an efficient sorting algorithm (such as Quicksort (see `http://en.wikipedia.org/wiki/Quicksort`) that runs in $O(n \log n)$ average time so that the average time complexity of the overall algorithm is also $O(n \log n)$. A pseudocode of median finding algorithm utilizing Quicksort is given below:

```
1: procedure FIND_MEDIAN_2(A, size)
2:     QUICKSORT(A)
3:     return A[(size-1)/2]
4: end procedure
```

Algorithm 3 There is a linear-time algorithm for finding the `k`'th largest number in an array. Read the section titled "Finding the `k`'th smallest value of an array" in Chapter 2 of the Carrano

book and the section titled "Partition-based selection" in the Wikipedia entry on selection algorithms (`http://en.wikipedia.org/wiki/Selection_algorithm`) for the description of an algorithm called *Quickselect* that has an average case time complexity of $O(n \log n)$. Then, read the "Median of Medians algorithm" in the Wikipedia entry `https://en.wikipedia.org/wiki/Median_of_medians` and the pseudocode and detailed analysis by Prof. David Eppstein at `http://www.ics.uci.edu/~eppstein/161/960130.html` for a linear ($O(n)$) worst case time selection algorithm. Utilizing this selection algorithm, we can find the median by just selecting the (n/2)'th element of the array. A pseudocode of this algorithm is given below:

```
1: procedure FIND_MEDIAN_3(A, size)
2:     return SELECT(A, (size+1)/2)
3: end procedure
```

Assignment:

1. Study all of the algorithms above using the given references and understand how the upper bounds are found for the running time of each solution. Then, describe how the complexity of each algorithm is calculated in your own words (as a separate paragraph for each of the algorithms).

2. You are given the implementation of each algorithm (in files named `findMedian.h` and `findMedian.cpp` on the course web site) with the following prototypes

   ```
   int FIND_MEDIAN_X( int input[], const int n );
   ```

   where `X` is the algorithm number (`X` $= 1, \ldots, 3$). Each solution requires that the input array is allocated with size `n`, and the input array is filled with `n` integers by the calling function. The solution function finds and returns the median. Study all of these solutions.

3. Write a driver (`main`) function that generates an array that contains random numbers and calls each of these solutions with that array as input. Then, run each solution and record the execution times when different input sizes are used. You can vary `n` within a large range. You are expected to try many different input sizes, both small inputs and very large inputs (as large as around half million to observe the effects of different growth rates.

4. The time required for constructing the input array is not included in the complexity analysis. Make sure that you give exactly the same array as input to each solution. Note that the contents of the `input` array may be modified inside each solution function.

5. Create a table containing all the values that you have recorded and include it in your report. In this table, each row should correspond to a particular value of `n` and each column should correspond to one of the three algorithms.

6. Use these results to generate a plot of running time (y-axis) versus the input size `n` (x-axis). In particular, you are expected to produce a plot as in Figure 2.3 of the handout chapter on algorithm analysis. Make sure that your plots are of high quality and all details are clearly labeled.

7. Similarly, plot the expected growth rates obtained from the theoretical analysis (as given for each algorithm above) by using the same `n` values that you used in your simulations.

8. Compare the expected growth rates and the obtained results, and discuss your observations in a paragraph in your report.

9. Provide the specifications of the computer you used to obtain these execution times. Make sure that you report the CPU, RAM, and operating system specifications. You can use any computer with any operating system for this assignment.

You can use the following code segment to compute the execution time of a code block (by including the `ctime` header file):

```
//Store the starting time
double duration;
clock_t startTime = clock();

//Code block
...

//Compute the number of seconds that passed since the starting time
duration = 1000 * double( clock() - startTime ) / CLOCKS_PER_SEC;
cout << "Execution took " << duration << " milliseconds." << endl;
```

An alternative code segment is as follows (by including the `chrono` header file):

```
//Declare necessary variables
std::chrono::time_point< std::chrono::system_clock > startTime;
std::chrono::duration< double, milli > elapsedTime;

//Store the starting time
startTime = std::chrono::system_clock::now();

//Code block
...

//Compute the number of milliseconds that passed since the starting time
elapsedTime = std::chrono::system_clock::now() - startTime;
cout << "Execution took " << elapsedTime.count() << " milliseconds." << endl;
```

Notes:

1. In this assignment, you MUST submit a report (as a `pdf` file) that contains all information requested above (description of the algorithms, table, plots, computer specification, discussion, etc.) and a `cpp` file that contains the `main` function that you used as the driver in your simulations in a single zip file. The name of this zip file should conform to the following name convention: secX-Firstname-Lastname-StudentID.zip where X is your section number. You MUST NOT include the solution functions that we provided in your submission. The submissions that do not obey these rules will not be graded.

2. You should prepare your report using a word processor (in other words, do not submit images of handwritten answers) and convert it to a PDF file. Handwritten answers and drawings will not be accepted. In addition, DO NOT submit your report in any other format than PDF.

3. Make sure that each file that you submit (each and every file in the archive) contains your name, section, and student number at the top as comments.

4. You are free to write your programs in any environment (you may use Linux, Windows, MacOS, etc.). On the other hand, we will test your programs on "dijkstra.ug.bcc.bilkent.edu.tr" and we will expect your programs to compile and run on the dijkstra machine. If we could not get your program properly work on the dijkstra machine, you would lose a considerable amount of points. Thus, we recommend you to make sure that your program compiles and properly works on dijkstra.ug.bcc.bilkent.edu.tr before submitting your assignment.

5. This assignment is due by 23:59 on Wednesday, November 23, 2022. You should upload your work to Moodle before the deadline. No hardcopy submission is needed. The standard rules about late homework submissions apply. Please see the course home page for further discussion of the late homework policy.

6. We use an automated tool as well as manual inspection to check your submissions against plagiarism. Please see the course home page for further discussion of academic integrity and the honor code for programming courses in our department.

7. This homework will be graded by your TA Klea Zambaku (klea.zambaku at bilkent.edu.tr). Thus, you may ask your homework related questions directly to her. There will also be a forum on Moodle for questions.