CS 201, Spring 2021

Homework Assignment 2

Due: 23:55, April 10, 2021

Note: Before implementing this assignment please make sure that you understand the concept of time complexity analysis. Please revisit course material on Algorithm Analysis.

In this homework, you will study search algorithms. That is, given an ordered collection of items, you are supposed to search for a specific element in a collection. The element is identified by a key. If the key exists, your solution needs to return the position of the key (i.e., the index). If not, it needs to return -1.

Consider the following 4 search algorithms. Each algorithm has different time complexity. The goal of this homework is to evaluate the growth rates of these algorithms.

Algorithm 1: Linear search (iterative implementation) which works in O(N) time, where N is the size of the collection.
Algorithm 2: Linear search (recursive implementation) which works in O(N) time, where N is the size of the collection.
Algorithm 3: Binary search which works in O(logN) time, where N is the size of the collection.
Algorithm 4: Jump search which works in O(N/m + m) time, where N is the size of the collection and m is the block size. Assume m = $\sqrt{N}$. So it works in O($\sqrt{N}$) time.

Algorithms 1 and 3 are discussed in the class. You can use the codes in the slides. Algorithm 2 is the recursive version of Algorithm 1, which you should implement yourself. You can find the description of Algorithm 4 here.

Note that the binary and jump search algorithms work on only sorted collections, so you have to provide a sorted collection of numbers for both algorithms (ascending).

In this assignment, you will need to record execution times within your code. Snippets for this task are provided below.

ASSIGNMENT:

1. Study algorithms and understand how the upper bounds are found for each.
2. Implement these algorithms and write a driver (main) function that calls these functions. Then, create sorted arrays of different sizes. You are expected to try many different input sizes, both small inputs and very large inputs. For each input size, consider the following four possibilities for the position of the key: (a) the key is close to the beginning, (b) the key is around the middle, (c) the key is close to the end, and (d) the key does not exist in

the collection. Run your program for each array size and for each possibility and record the execution times. Note that you need to pick a proper range of values for your array. That is if the size of your array has N items, pick items in the range [0: 10*N]. For instance if, N = 100, pick numbers from the range of [0: 1000]. **Do not include the time elapsed to allocate the array and initialize its entries.** See the pseudo procedure below

2.1.1. Pick an N. For instance N = 10.

2.1.2. Create a sorted random array of size N. For instance, [0,5,7,12,14,23,34,78,84,98]

2.1.3. Run Algorithm 1 on this array where the key is close to the beginning. Measure the time that it takes to find that key. Do this also for all other three scenarios. You will obtain 4 execution times.

2.1.4. Using the same exact array, repeat the process in (2.1.3) for Algorithms 2 - 4. Again, you will get four different values for each run.

2.1.5. Return to step (2.1.1) and repeat this process over and over again each time increasing the size of N. Until you think you have accumulated enough results to explain the behavior of each algorithm (Minimum 10).

Let's say you run this method for ten N values. Then you should get 10x4=40 running times for each algorithm.

Once again, you HAVE TO use the same array for all algorithms and scenarios to keep the experiment consistent.

3. Create a table containing all the values that you have recorded and include it in your report. The layout of the table is shown below. You cannot change the column format of the table but you can include as many rows as you like as long as it is more than 10.

N	Linear (Iterative)				Linear (Recursive)				Binary Search				Jump Search			
	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d
10																
100																
1000																

4. Create a plot for each search algorithm where N values are on the x-axis and elapsed times are on the y axis. You will get 4 plots (algorithms) each with 4 lines (scenarios). Include these plots in your report.

Note you are responsible for the quality of your plots. Sloppy and poorly scaled plots will lead to a reduction of a significant amount of points even if you conducted all the experiments perfectly. Consider using Python or MATLAB for plotting, as they contain built-in libraries for plotting similar functions. If the plot you generated makes it hard to draw conclusions about the behavior of the algorithm, then you should probably consider another way of plotting it.

5. Comment on your results. More specifically, you must answer the following questions
   5.1. What are the specifications of your computer? (Processor, RAM, operating system and etc)
   5.2. What are the theoretical worst, average, and best cases for each algorithm? What are the expected running times for each scenario for each algorithm?
   5.3. What are the worst, average, and best cases for each algorithm based on your table and your plots? Do theoretical and observed results agree? If not, give your best guess as to why that might have happened.
   5.4. Plot theoretical expected running times for each algorithm across different scenarios and compare them to the plots at step 4. Comment on consistencies and inconsistencies. (These plots should have the same organization as plots in step 4)

You can use the following code segment to compute the execution time of a code block. For these operations, you must include 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 to compute the execution time is as follows. For these operations, you must include 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 seconds that passed since the starting time
elapsedTime = std::chrono::system_clock::now() - startTime;
cout << "Execution took " << elapsedTime.count() << " milliseconds." << endl;
```

NOTES ABOUT SUBMISSION:

1. This assignment will be graded by your TA Aydamir Mirzayev (aydamir.mirzayev@bilkent.edu.tr). Thus, you should ask your homework-related questions directly to him.

2. The assignment is due by April 10, 2021, at 23:55. You should upload your homework to the upload link on Moodle before the deadline. No hardcopy submission is needed. The standard rules about late homework submissions apply. Please see the course web page for further discussion of the late homework policy as well as **academic integrity**.

3. You must submit a report (as a PDF file) that contains all information requested above (plots, tables, computer specification, discussion) and a cpp file that contains the main function that you used as the driver in your simulations as well as the solution functions in a single archive file.

4. You should prepare your report (plots, tables, computer specification, discussion) 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.

5. The code segments given above may display 0 milliseconds when the value of N is small. Of course, the running time cannot be 0 but it seems to be 0 because of the precision of the used clock. However, we will not accept 0 as an answer. Thus, to obtain a running time greater than 0, please consider running algorithm M times using a loop, and then divide the running time (which will be greater than 0) to M.