

CSC211—Problem Solving, Algorithms and Programming

Compare timings for linear search and binary search.

Term 1 Practical 3

23/25 February 2026

Submit today *before* leaving the practical at 16h35, 23/25 February 2026

Please follow the instructions carefully. You will receive marks for everything done correctly that is submitted before you leave the practical *today*.

1. You must use **git** that should already have been initialized during the first week using **git init** inside your *name file* that was created to hold all your pracs. The practical for this week must be done inside the directory **13practical** which must lie inside your *name file* where you initialised **git** and parallel to the first week's **11practical** and last week's **12practical** directory. Your final submission from a Windows system is always a zipped file of the *entire name file*, or if you use Linux a tarball created of the directory containing your name file using **tar -zcvf surname,firstname.tgz ***.
2. You are required to run regularly, i.e., at least every thirty minutes, or after adding about ten to twenty lines of code, a commit command such as **git commit -m "Explanation of update"**
You will be penalized for not doing “enough” commits.
3. The next practical will be done under the directory **14practical**, and the one in the fifth week under **15practical**, etc.
4. Your work in the form of a **.zip** or **.tgz** file must be submitted to iKhamva, before leaving the practical.
5. We will mark your work as soon as possible when the demis start working, or when we get the automatic marking software going, and give you immediate feedback. We are still attempting to get the automatic marking software in place. When this works you will get feedback in real time, despite always uploading completed work to iKamva *today* before leaving the laboratory.
6. **Overview** The idea of today's practical is to collect useful data for the running times of two simple algorithms for *looking up* keys in order to access data in an array of records. Each of these records is a **Node**, that consists of an integer **key** and its associated **String** of **data**. These records are given to you in the file **ulysses.numbered**. Your code will run two sets of lookups on this data stored in an array in your program. When doing linear search one assumes that the keys are *unsorted* and when doing binary search the algorithm needs a sorted set of keys. You need to determine the speed of the two algorithms.
7. The keys lie in the range **00001–32654**. Your code must compare only the lookup—insertion and deletion need not be done—for each of the two kinds of lists. Note that you need to have only one copy of the array in your code. The assumption that the code is sorted for doing binary search must be made. The same data is used for doing both the linear searching and binary searching. The idea is that the experiment must convince you that binary searching does in fact work in logarithmic time and that linear searching is a lot slower.

8. The experiment will be done as follows. Generate 30 keys in the right range and **look up** each key, getting the average timing and standard deviation for each of these runs to produce meaningful statistics. Tabulate the average and standard deviation of these two times.
9. The final result that your code generates is four numbers, i.e., two averages and two standard deviations of each of the timings.
10. The code below may be used as a starting template for your programming.

```

1  // Code is stored as 13template.java
2  import java.lang.Math.*; import java.io.*; import java.text.*;
3
4  public class timeMethods{
5      public static int N = ....;
6      public static void main(String args[]){
7
8          DecimalFormat twoD = new DecimalFormat("0.00");
9          DecimalFormat fourD = new DecimalFormat("0.0000");
10         DecimalFormat fiveD = new DecimalFormat("0.00000");
11
12         long start, finish;
13         double runTime = 0, runTime2 = 0, time;
14         double totalTime = 0.0;
15         int n = N;
16         int repetition, repetitions = 30;
17
18         runTime = 0;
19         for(repetition = 0; repetition < repetitions; repetition++) {
20             start = System.currentTimeMillis();
21
22             // call the procedures to time here:
23             linearsrch (...);
24             binarysrch (...);
25             // Figure out how to alter this guideline here,
26
27             finish = System.currentTimeMillis();
28
29             time = (double)(finish - start);
30             runTime += time;
31             runTime2 += (time*time); }
32
33         double aveRuntime = runTime/repetitions;
34         double stdDeviation =
35             Math.sqrt(runTime2 - repetitions*aveRuntime*aveRuntime)/(repetitions-1);
36
37         System.out.printf("\n\nStatistics\n");
38         System.out.println("_____");
39         System.out.println("Total_time_uuu=uuuuuuuuuu" + runTime/1000 + "s.");
40         System.out.println("Total_time\u00b2_uu=uuuuuuuuuu" + runTime2);
41         System.out.println("Average_time_u=uuuuuuuuuu" + fiveD.format(aveRuntime/1000)
42             + "s." + "\u00B1" + "u" + fourD.format(stdDeviation) + "ms.");
43         System.out.println("Standard_deviation_u=uuuuu" + fourD.format(stdDeviation));
44         System.out.println("n_uuuuuuuuuuu=uuuuuuuuuu" + n);
45         System.out.println("Average_time_u/run_u=uuuuu" + fiveD.format(aveRuntime/n*1000)
46             + "\u00B5" + "s.");
47
48         System.out.println("Repetitions_uu=uuuuuuuuuuuu" + repetitions);
49         System.out.println("_____");
50         System.out.println();
51         System.out.println(); } }
52
53 static void oneofyourMethods(int n,
54                             yourMethodParameter1,
55                             yourMethodParameter2, . . . ) {
56     // The declarations and body of your method / s
57     // The final statement of this code.} }

```