CPSC 331 (Spring 2023)
Assignment 4
Hash Tables and Searching
(8% of final mark)
Due: Wednesday June 7th at 11:59PM

Lead TA: The lead TA for this assignment is: Joweria Ekram (joweria.ekram1@ucalgary.ca)

Objective: Compare linear search, binary search, and searching a hash table.

Part I
You will experimentally compare the running time of three different search algorithms in Java.

Write a method to randomly generate an array of integers called the *search* array. This is the array to be searched. The array size *n* is between 1000 and 1,000,000 and the values in the array are from 0 to 5000. You will generate 99 distinct random numbers between 0 and 5000 and store them in an array of size 100. Add 5001 as the 100th cell in the array. We will refer this as the *elements* array. The search array is searched for each element in the elements array. The elements array is generated once and is fixed for all the search methods. For each *n*, the array is also fixed for all search methods.

Sequential Search
Run the sequential search algorithm for each element in the elements array looking for elements in the search array. Measure the total time in nanoseconds required to search for all the elements in the elements array.

Binary Search
Sort the array using any sorting method provided in the Soring class provided on D2L or code any sorting method of your own. Then, run binary search on the array measuring the total time in nanoseconds required to search for all the elements from the elements array. Do not count the sorting time in the time you are measuring to search the array.

Plot the running time of each search method for a search array size of 1000 to an array of size 1,000,000. The time in nanoseconds is on the y-axis and the size of the search array size (*n*) on the x-axis. Each point in the chart for a given *n*, is the search time for all the 100 elements in the elements array for an array of size *n*. Make sure you use the same array contents for each *n*, whether you are using sequential or linear search.

For proper memory utilization, do not create all the search arrays at once. Create them one at a time: you can create one search array for size *i*, search it and record the running time, dispose of it (it is sufficient to do `a = null`, where `a` is the array name; this will make the memory allocated to the array a candidate for garbage collection and will be freed), then create the array of size *i*+1, use it, dispose of it and repeat.

Part II
Repeat Part I, comparing binary search and searching a hashing table. You must implement your own hash table using chaining. Use the following hash function:

```
int hash(int x) {
    x = ((x >>> 16) ^ x) * 0x45d9f3b; // >>> is unsigned right shift
    x = ((x >>> 16) ^ x) * 0x45d9f3b;
    x = (x >>> 16) ^ x;
    return Math.abs(x)%hashTableSize; }
```

Do not include the time needed to build the hash table in your running time analysis (just like you did not include the sorting time for binary search). Choose 9973 for the hash table size. This is a prime number.

Deliverables

- Java source code
- A PDF file containing a report (two pages maximum) that includes:
    - Your comments on the chart comparing linear search and binary search. Is there a big difference? If so, which value of *n* is the turning point? Which method would you use and under which circumstances taking into consideration the time needed to sort the array?
    - Your comments in the chart comparing binary search and hash search. Is there a big difference? If so, which value of *n* is the turning point? Which method would you use and under which circumstances taking into consideration the time to search the array and the time to build the hash table?

Submission: Submit the deliverables to the appropriate dropbox. The TA may provide extra submission requirements or instructions.

Collaboration: You are advised to work on this assignment in pairs. Groups of more than two members are not allowed. You may change your partner from the previous assignment. Any discussion with your colleagues outside your team regarding the assignment must be kept at a very high level.

Late Submission Policy: Late submissions will be penalized as follows:
-12.5% for each late day or portion of a day.

Academic Misconduct: Any similarities between submissions will be further investigated for academic misconduct. Your final submission must be your own team's original work. While you are encouraged to discuss the assignment with colleagues outside your team, this must be limited to conceptual and design decisions. Code sharing by any means with colleagues that are not in your team is prohibited. This includes and is not limited to looking at someone else's paper or screen. Any re-used code of excess of 5 lines must be cited and have its source acknowledged. Failure to credit the source will also result in a misconduct investigation.

D2L Marks: Any marks posted on D2L are tentative and are subject to change (UP or DOWN).

Marking Rubric (70 points total)

Code	**Total 50 points**
Code compiles	5
Code runs	5
Implementation of linear search is correct	5
Implementation of binary search is correct	5
Hash table implementation is correct	10
The stats are calculated correctly	10
Code is modular	5
Code contains enough documentation	5
Report	**Total 20 Points**
The charts are clear, readable, and convey the required message	10
The discussion is thoughtful and utilizes the charts	10

Submissions that do not compile will receive no points. Code that compiles but does not run will not receive more than 5/100.