Assignment 4

Searching in Arrays & Linked Lists

Release Date	Due Date
November 9, 2023	November 30, 2023

Objectives

- Experience various techniques to search arrays and linked lists
- Practice developing high-performance solutions
- Compare theoretical vs empirical complexities
- Compare linear and binary search of arrays
- Understand at times somewhat vague, requirements to develop a computing application
- Design and develop different solutions to a computing problem

Problem Specification

Linear and binary searches are common techniques used to search in linked lists and arrays. In this assignment, you will get experience in implementing these search algorithms and comparing their performances. You will also gain experience in extending binary search.

For this assignment you are given an input file named “games.csv” (this dataset is taken from website Kaggle.com and then edited). This input file consists of 6 columns: “id”, “name”, “average user rating”, “user rating count”, “developer” and “size”. This input file has ~2300 rows. In this assignment, you will read data from this input file and then store it in a LinkedList and perform linear search and binary search.

Write an application that performs following task:

1) Read in the row from the given input file and store them in a doubly-linked list, called games*LinkedList*.

   - Create a data structure to represent each game. This data structure should store the id, name, average user rating, user rating count, developer and size.

2) While storing data in the gamesLinkedList make sure that you don’t store duplicates. If there exist any duplicates based on name, store one with the higher “user rating count” only.

   Example: let’s say while reading file, you encounter game named ‘xyz’ then first check if you already have game stored in your LinkedList with same name (this time use linear search to traverse through LinkedList to search for duplicates). If not found in the LinkedList add this game to the LinkedList. But if found, then check for the value of “user rating count”. If the newly read row has a larger “user rating count” than one already stored in the LinkedList, remove the one stored in the LinkedList and add the newly read row to the LinkedList (don’t replace the old one with the new one, delete the old one and you have to store the data to LinkedList in the same order as you encounter it in

the input file). But if the newly read row have a smaller "user rating count" than one already stored in the LinkedList, just ignore this row (don't store the row in the LinkedList).

3) Randomly choose an item from the original linked list and then perform linear search based on the name of the item you chose. Report the item details you are searching for. Report the time spent searching for the item.

4) Further test the speed of your search algorithm. Repeat the process in step 3) multiple times, and report the average time searching. For this step, you don't have to print anything except the final average time that you calculated.

5) Sort your inventory by item name. Use two different sorting algorithms, namely insertion sort and quick sort, and record the time taken for these.

6) Repeat step 3) and 4), now using Binary search instead of Linear search (make sure you search for same items that you searched for in the linear search test in step 3).

   If your code runs very fast (especially for small input sizes) then the elapsed time (using system supplied time functions) might be too small to be recorded. What can you do to remedy the situation?

   Remember binary search works most efficiently on a sorted array! So, for m searches, there is no point in sorting repeatedly and then binary-searching or searching repeatedly using linear searches, but sort once and then perform multiple searches.

7) Find the range of *m* for your code where the solution of *m* linear searches is better than the solution of quick sort and m binary-searches.

Sample output

```
Number of elements in LinkedList: <number>

*** Linear Search Test ***

Before sorting:
# Print only first 5 elements from linked list, for example:
# 284921427, Sudoku, 4, 3553, Mighty Mighty Good Games, 15853568

Search number 1:
Searching for <name1>...
Single search time: xxxxx nanoseconds.
Average search time: yyyyy nanoseconds.

Search number 2:
Searching for <name2>...
Single search time: xxxxx nanoseconds.
Average search time: yyyyy nanoseconds.
```

```
Search number 3:
Searching for <name3>...
Single search time: xxxxx nanoseconds.
Average search time: yyyyy nanoseconds.

After sorting:
# Print only first 5 elements from sorted array

Time for insertion sort: xxxxx nanoseconds.
Time for quick sort: yyyyy nanoseconds.

*** Binary Search Test ***

Search number 1:
Searching for <name1>...
Single search time: xxxxx nanoseconds.
Average search time: yyyyy nanoseconds.

Search number 2:
Searching for <name2>...
Single search time: xxxxx nanoseconds.
Average search time: yyyyy nanoseconds.

Search number 3:
Searching for <name3>...
Single search time: xxxxx nanoseconds.
Average search time: yyyyy nanoseconds.

Break point m = xxxx between repeated linear searches and sort-once & multiple binary searches.
```

Output Requirements

- Report the number of elements in the gamesLinkedList.
- Step 3's output requirements (Average time spent searching, searched item's names).
- Steps 5 and 6’s output measured timings.
- Perform search at least 3 times for 3 different names and report result as mentioned above. For measuring better averages, you might actually perform many more searches than what you print.
- Step 7’s output is the value of m.

Design Requirements

Code Documentation

For this assignment, you must include documentation for your code. This includes how to compile and run your program.

Coding Conventions and Programming Standards

You must adhere to all coding conventions and standards. This includes the use of white spaces for readability and the use of comments to explain the meaning of various methods and attributes. Be sure to follow the conventions for naming files, classes, variables, method parameters and methods.

Testing

Make sure you test your application with several different values capturing different cases, to make sure it works.

Assignment Submission

- Generate a .zip file that contains all your files, including:
  - Source code files
  - Including any input or output files
  - Documentation of your code
  - A brief report (in a pdf file) on your observations of comparing theoretical vs empirically observed time complexities. Note this report will include (a) a brief description of problem statement(s), (b) algorithms descriptions (if these are standard, commonly known algorithms, then just mention their names along with customization to your specific solution(s), otherwise give the pseudo-code of your algorithms, (c) theoretically derived complexities of the algorithms used in your code, (d) table(s) of the observed time complexities, and (e) plots comparing theoretical vs. empirical along with your observations (e.g. do theoretical time complexities agree with your implementation, why? Why not?).