Programming Assignment 1
Searching in 1-D Dynamic Data Structures

Assignment spec may change depending upon discussion in the class

Release Time	Due Date
September 10, 2024	**September 24, 2024**

Objectives

- Creating dynamic-arrays (aka growable-arrays, ArrayList of Java or simply List of Python)
- To practice add, delete, search, sort operations using linked lists and arrays
- Practice developing high-performance solutions
- Compare theoretical vs empirical complexities

Problem Specification

In reality, we always use various data structures to store data, and different data structures have different advantages and disadvantages. For instance, some data structures are good at adding/deleting new data, but they may have deficiency on searching. The two basic 1-d data structures, namely linked lists and arrays have been discussed in class. While the array provides random access, linked lists provide efficient insert and delete. Arrays needs allocation of contiguous memory locations. In an attempt to harness the capabilities of both, arrayList type of data structures have been designed which depict good amortization behavior. From this assignment, let's learn and compare. After the assignment is complete, you will know more about what are the real differences between these two data structures. Here we go.

Write a Python application to solve the following problem:

1) Use dataset Open English Word List (EOWL) - https://github.com/dwyl/english-words

2) Store the words in EOWL into a sorted dynamic-array which your program will manipulate. Start with size 2 dynamic array eowl[] and increase the size of eowl[] using a few different strategies when the current eowl[] array becomes full.

3) Increase strategyA: Incremental: increase the size of eowl[] by 10.

4) Increase strategyB: Doubling: double the size of eowl[].

5) Increase strategyC: Fib: use Fibonacci number sequence to increase the size of eowl[].

6) Use binary-search to insert the new word into eowl[], which will be done after allocating new array when an increase is needed.

7) Measure time at each search point.

8) Perform a theoretical complexity analysis of your design (i.e., count number of operations/instructions and space usage) and then express that using asymptotic notation as a function of the input size (Pause: what is the input size of your problem?)

9) Repeat steps 2 and 6 by simply using lists in Python which implement a version of ArrayLists.

10) Empirically measure the time and space complexity of your code (step 7 above should help towards that).

11) Compare the complexities from steps 8) and 10). (Long pause: in order to compare, what all you will need to do? We will discuss some alternatives in class, so stay tuned and this writeup may be modified accordingly at a later date.)

12) No need to print the whole eowl[]. At each increase point print the current size of eowl[], time elapsed and the following elements: 1^{st}, $[n/4]^{th}$, [n/2]th, $[3n/4]^{th}$ and n^{th}, separated by →.

Design Requirements

Code Documentation

For this assignment, you must properly document your code and use good software development practices.

Github

Use github to store your repository. Incorporate branching and forking as reasonable to practice these features of an RCS.

Testing

Make sure you test your application with several different values capturing different cases, to make sure it works. Allow your application to optionally read filename ending in .txt which can be used instead of the EOWL dataset.

Assignment Submission

- Generate a .zip file that contains all your files, including:
  - Source code files
  - any input or output files
  - a 4-5 screenshots of your testing of the implementation.
- Don't forget to follow the naming convention specified for submitting assignments
- You will also show execution of your application to grader / instructor. They may give you a test case or two on the spot.