Project 1: Benchmarking Search Algorithms

Problem

Expected Duration: 4 hours

One claim is that for sufficiently large lists, binary search is significantly faster than sequential search. For this project, you will benchmark the speed difference between linear search, recursive binary search, and jump search. You will implement each search function yourself, and time the results. You will need to use lists large enough to show a difference in speed to 2 significant digits.

This project is about important theoretical and practical algorithms, rather than abstract data types. Sorting and searching are at the heart of many ideas and algorithms in Computing as a Science. This project will help train your intuition for algorithm analysis and Big-O notation. Expected timing values assume that you implement good versions of the algorithms. Recursion is not required though you may use it if you want.

Search Functions

Implement the following search algorithms as predicate (boolean) functions. By definition, this means they return True only if the target is in the list, False otherwise. Assume the target type is integer, and the list contains only integers. Use of "lyst" as an identifier is NOT a typo since "list" is a Python type. For all algorithms sort the list before calling the search.

- linear_search(lyst, target)
- binary_search(lyst, target)
- jump_search(lyst, target)

Main Program: Benchmarking

Create a **non-interactive** main function that runs each search, times each run, and reports the results to the console in the order above. Use an array size that is large enough to demonstrate differences in speed-- typically in the range 1,000,000 - 10,000,000 values, but you may have to go larger. Use the same list for all algorithms.

- Your timing results should be to at least 2 **significant** digits of difference.
- For all 3 algorithms, report times for the following results:
    1. the first element of the sorted array
    2. a number at the middle of the sorted array

3. a number at the end of the sorted array
4. a number NOT in the array (try -1)
5. print results in seconds

- Do not do any printing inside the searches as this will give incorrect timing results.

Generating Lists Using Random Numbers

Use the functions in the random module for generating lists of numbers.

- random.seed(seed_value): The seed can be any integer value, but should be the same each time so that you can duplicate results for testing and debugging.
- random.sample(*population*, *k*): generates *k*-length random sequence drawn from *population* without replacement. Example: `sample(range(10000000), k=60)` will generate a list of 60 unique values between 0 and 10 million

Use a built-in sort function to sort the list after it is generated.

Timing functions

There are several ways to benchmark function calls in Python. For this project, Use the `time.perf_counter()` to calculate elapsed time for runtimes.

Grading (100 points)

pylint will be run on your search.py. Expected minimum score is 8.5.

Score is the sum of:

- Percentage of automated test cases passed x 80 points
- min(Coding style score/8.5, 1) x 20 points
- Possible adjustment due to physical inspection of the code, to make sure you actually implemented things.

What to Submit

- search.py (includes conditional execution of main()) → your main code that does the timing. It will not be tested.
- A short 1-minute video showing your code running. If on Youtube, make sure the link is unlisted and not searchable, and paste the link in a comment with your submission.