Project 1 - Search

Purpose: One claim is that for sufficiently large lists, binary search is significantly faster than sequential search. For this project, you will demonstrate 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 of 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. The timing/speed values we expect assume that you implement good versions of the algorithms.

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, the list must be sorted.

- linear_search(lyst, target)
- recursive_binary_search(lyst, target)
  - recursive_binary_search_helper(low_index, high_index, lyst, target)
- jump_search(lyst, target)

Main Program: Benchmarking

Create a **non-interactive** main function that runs each each search, times each run, and reports the results to the console. * Use large array size, typically in the range 1,000,000 - 10,000,000 values

- Your timing results should be to at least 2 significant digits.
- For all 3 algorithms, report 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)
- **Do not do any printing inside the searches as this will give incorrect timing results.**

```
Creating a sorted array of 10000000
Finished creating a sorted array of 10000000

Searching for a number at the start of the array
        linear_search() returned True in 0.0000082 seconds
        recursive_binary_search() returned True in 0.0000156 seconds
        jump_search() returned True in 0.0000133 seconds
Searching for a number in the middle of the array
        linear_search() returned True in 0.6767016 seconds
        recursive_binary_search() returned True in 0.0000053 seconds
        jump_search() returned True in 0.0009621 seconds
Searching for a number at the end of the array
        linear_search() returned True in 1.2818304 seconds
        recursive_binary_search() returned True in 0.0000172 seconds
        jump_search() returned True in 0.0014824 seconds
Searching for a number NOT in the array
        linear_search() returned False in 1.2937177 seconds
        recursive_binary_search() returned False in 0.0000182 seconds
        jump_search() returned False in 0.0003891 seconds
```

Figure 1. Example Benchmark Report for 3 Search Algorithms. Actual search algorithms expected times may de different.

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)

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

Timing functions

Use the time.perf_counter()

to calculate runtimes.

Test Cases

Test Search Timing

While it is difficult to test EXACT search timing, certain searches will be faster than others. This test will verify that the binary and jump searches are faster than the linear search, when the target is at the end of the list. In addition to timing, this test will check to see that the proper value (True or False) is returned from a search. It will search for a target at the beginning, middle, and end of the list. In addition, it will check for a False returned value if the target is not in the list.

Test Parameter Types

All searches need to validate the "type" of data passed in. The target must be an integer and the elements of lyst must also be integers. If not, the search should raise a ValueError exception.

Test Binary Search Recursion

The binary search must use recursion. To test for this, your recursive "helper" function needs to have one extra line at the beginning:

```
from recursioncounter import RecursionCounter
      # place at the beginning of the module


def recursive_binary_search_helper(lyst, low_index, high_index, target):
    RecursionCounter()     # recursion helper. DO NOT CALL DIRECTLY
```

This line to create a RecursionCounter object will allow the UnitTest to verify that your code really is using recursion.

Test Coding Standards

PyLint will be used to verify that your python code is clean and done according to the PEP-8 coding standard. Your code will need to achieve a score of 8.5 or higher.

Grading (100 pts)

This data will come from the results of the Unit Test

- Search Timing (relative) 55
- Parameter Checking 15
- Recursion 15

- Coding Standards 15

Files to turn in (through Canvas)

- search.py (includes the main() driver)