Queens College of CUNY
Department of Computer Science
Discrete Structures (CSCI 323)
June 2024

Programming Assignment #1:
"Array-Search Algorithms"
Due: June 5, 2024

Introduction:

One of the most basic programming problems is "search", that is finding an item ("key") in a structure. The structure may be linear (array) or hierarchical (tree), ordered (e.g. alphabetical or numerical order) or unordered (no particular order). There are several search algorithms available. In this assignment, we consider a sorted list of random data and compare several algorithmic approaches:

Submissions:

Please submit the following in the Google form that will be provided.

- Source code (Assignment1.py)
- Console output with chart (Assignment1.txt)
- Bar graph (Assignment1.png)

Preliminary Tasks:

[1] Install the Python language and interpreter and the PyCharm integrated development environment (IDE) and get the sample "Hello User" program running as outlined in Assignment #0.

[2] Add a comment at the top. (This will also be a template for commenting future assignments)

```
# Analysis of Algorithms (CSCI 323)
# Summer 2024
# Assignment 1 - Empirical Analysis of Search Algorithms
# Jane Doe (your name)

# Acknowledgements:
# I worked with ... (if applicable)
# I used the following sites ... (if applicable)
```

[3] Define a function main() which will be the springboard for the execution of all other code in this assignment.

```
def main():
    pass # placeholder until actual function content is provided
```

[4] Add these lines just below the main() function to ensure that importing of this module into another program will not automatically run it.

```
if __name__ == "__main__":
    main()
```

[4] Create a program configuration. Note: one will automatically be created if you run via the Run menu item.

Assignment-Specific Tasks:

[1] Define a function random_list(size) that returns a list of size random numbers.

Can use random.sample().
See https://stackoverflow.com/questions/22842289/generate-n-unique-random-numbers-within-a-range
Then sort the list in sorted order.

[2] Define a function native_search(list, key) that wraps the built-in index function

[3] Define the following functions utilizing the code from the references provided. Ensure that signatures are consistent.:

- linear_search(list, key) - see https://www.geeksforgeeks.org/linear-search/
- binary_search(list, key) - see https://www.geeksforgeeks.org/binary-search/
- better_binary_search(list, key) - see https://www.geeksforgeeks.org/the-ubiquitous-binary-search-set-1/
- randomized_binary_search(list, key) - see https://www.geeksforgeeks.org/randomized-binary-search-algorithm/
- exponential_search(list, key) - see https://www.geeksforgeeks.org/exponential-search/
- interpolation_search(list, key) - https://www.geeksforgeeks.org/interpolation-search/
- jump_search(list, key) - https://www.geeksforgeeks.org/jump-search/
- fibonacci_search(list, key) - https://www.geeksforgeeks.org/fibonacci-search/
- ternary_search(list, key) - https://www.geeksforgeeks.org/ternary-search/

See below regarding Interpolation Search, Jump Search

[4] Add code to verify that the current search determines the correct position

[5] Add code to time each of the search functions

[6] Add code to run each search multiple times for different random keys

[7] Add code to plot the timings for the algorithms. See https://www.geeksforgeeks.org/matplotlib-tutorial/

Use this code as a starting point for plotting the times. However, modify it to make it more generic (not just for searches).

```
def plot_times(dict_searches, sizes, trials, searches, file_name):
    search_num = 0
    plt.xticks([j for j in range(len(sizes))], [str(size) for size in sizes])
    for search in searches:
        search_num += 1
        d = dict_searches[search.__name__]
        x_axis = [j + 0.05 * search_num for j in range(len(sizes))]
        y_axis = [d[i] for i in sizes]
        plt.bar(x_axis, y_axis, width=0.05, alpha=0.75, label=search.__name__)
    plt.legend()
    plt.title("Runtime of search algorithms")
    plt.xlabel("Number of elements")
    plt.ylabel("Time for " + str(trials) + " 100 trials (ms)")
    plt.savefig(file_name)
    plt.show()
```

Use this code as a starting point for printing the results:

```
def print_times(dict_searches)
    pd.set_option("display.max_rows", 500)
    pd.set_option("display.max_columns", 500)
    pd.set_option("display.width", 1000)
    df = pd.DataFrame.from_dict(dict_searches).T
    print(df)
```

Use this code as a starting point for running the searches:

```
def run_searches(searches, sizes, trials)
    dict_searches = {}
    for search in searches:
        dict_searches[search.__name__] = {}
    for size in sizes:
        for search in searches:
            dict_searches[search.__name__][size] = 0
        for trial in range(1, trials + 1):
            arr = random_list(size)
            idx = random.randint(0, size-1)
            key = arr[idx]
            for search in searches:
                start_time = time.time()
                idx_found = search(arr, key)
                end_time = time.time()
                if idx_found != idx:
                    print(search.__name__, "wrong index found", arr, idx, idx_found)
                net_time = end_time - start_time
                dict_searches[search.__name__][size] += 1000 * net_time
      return dict_searches

def main():
    sizes = [10, 100, 1000, 10000]
    searches = [native_search, linear_search, binary_search, interpolation_search]
    trials = 1000
      print_times(dict_searches)
      plot_times(dict_searches, sizes, trials, searches)
```

Make this fix to Jump Search near the end:

```
if arr[int(prev)] == key:
        return int(prev)
```

The code for Interpolation Search on Geeks for Geeks has a bug. This one should work instead, though you may want to change the order of the parameters:

```
def interpolation_search_helper(arr, lo, hi, key):
    if arr[lo] == arr[hi]:
        if key == arr[lo]:
            return lo
        else:
            return -1
    if lo <= hi and arr[lo] <= key <= arr[hi]:
        pos = int(lo + ((hi - lo) / float(arr[hi] - arr[lo])) * float((key - arr[lo])))
        if arr[pos] == key:
            return pos
        elif arr[pos] < key:
            return interpolation_search_helper(arr, pos + 1, hi, key)
        else:
            return interpolation_search_helper(arr, lo, pos - 1, key)
    return -1
```