CMPS 2200 Recitation 01

Name (Team Member 1):_____

Name (Team Member 2):_____

In this recitation, we will investigate asymptotic complexity. Additionally, we will get familiar with the various technologies we'll use for collaborative coding.

To complete this recitation, follow the instructions in this document. Some of your answers will go in this file, and others will require you to edit `main.py`.

Setup

- Make sure you have a Github account.
- Login to Github.
- Login to repl.it, using "sign in with github"
- Click on the assignment link sent through canvas and accept the assignment.
- Click on your personal github repository for the assignment (e.g., https://github.com/tulane-cmps2200/recitation-01-your_username).
- Click on the "Work in Repl.it" button. This will launch an instance of `repl.it` initialized with the code from your repository.
- You'll work with a partner to complete this recitation. To do so, we'll break you into Zoom rooms. You will be able to code together in the same `repl.it` instance. You can choose whose repl.it instance you will share. This person will click the "Share" button in their repl.it instance and email the lab partner.

Running and testing your code

- In the command-line window, run `./ipy` to launch an interactive IPython shell. This is an interactive shell to help run and debug your code. Any code you change in `main.py` will be reflected from this shell. So, you can modify a function in `main.py`, then test it here.
    - If it seems things don't refresh, try running `from main import *`
- You can exit the IPython prompt by either typing `exit` or pressing `ctrl-d`
- To run tests, from the command-line shell, you can run
    - `pytest main.py` will run all tests
    - `pytest main.py::test_one` will just run `test_one`
    - We recommend running one test at a time as you are debugging.

Turning in your work

- Once complete, click on the "Version Control" icon in the left pane on repl.it.
- Enter a commit message in the "what did you change?" text box
- Click "commit and push." This will push your code to your github repository.

- Although you are working as a team, please have each team member submit the same code to their repository. One person can copy the code to their repl.it and submit it from there.

Comparing search algorithms

We'll compare the running times of `linear_search` and `binary_search` empirically.

- □ 1. In `main.py`, the implementation of `linear_search` is already complete. Your task is to implement `binary_search`. Implement a recursive solution using the helper function `_binary_search`.

- □ 2. Test that your function is correct by calling from the command-line `pytest main.py::test_binary_search`

- □ 3. Write at least two additional test cases in `test_binary_search` and confirm they pass.

- □ 4. Describe the worst case input value of `key` for `linear_search`? for `binary_search`?

TODO: your answer goes here

- □ 5. Describe the best case input value of `key` for `linear_search`? for `binary_search`?

TODO: your answer goes here

- □ 6. Complete the `time_search` function to compute the running time of a search function. Note that this is an example of a "higher order" function, since one of its parameters is another function.

- □ 7. Complete the `compare_search` function to compare the running times of linear search and binary search. Confirm the implementation by running `pytest main.py::test_compare_search`, which contains some simple checks.

- □ 8. Call `print_results(compare_search())` and paste the results here:

TODO: add your timing results here

- □ 9. The theoretical worst-case running time of linear search is $O(n)$ and binary search is $O(log_2(n))$. Do these theoretical running times match your empirical results? Why or why not?

TODO: your answer goes here

- □ 10. Binary search assumes the input list is already sorted. Assume it takes $\Theta(n^2)$ time to sort a list of length n. Suppose you know ahead of time that you will search the same list k times.
  - − What is worst-case complexity of searching a list of n elements k times using linear search? **TODO: your answer goes here**

– For binary search? **TODO: your answer goes here**
– For what values of k is it more efficient to first sort and then use binary search versus just using linear search without sorting? **TODO: your answer goes here**