CMPS 2200 Recitation 02

Name (Team Member 1):__________________________________
Name (Team Member 2):__________________________________

In this recitation, we will investigate recurrences. 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

- 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.
  - If you don't see a "Work in Repl.it" button, instead do:
    - Go to repl.it
    - Click the "+" to make a new repl.it
    - Click "Import from github"
    - Enter your github assignment url: e.g., https://github.com/tulane-cmps2200/recitation-01-your-username
- 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 the share link.

Running and testing your code

- Clicking the "play" button will run all tests in your code.
- It's usually best to run only one test at a time. To run tests, from the command-line shell, you can run
  - `pytest -s main.py` will run all tests
  - `pytest -s main.py::test_one` will just run `test_one`

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.

Recurrences

In class, we've started looking at recurrences and how to we can establish asymptotic bounds on their values as a function of n. In this lab, we'll write some code to generate recursion trees (via a recursive function) for certain kinds of recurrences. By summing up nodes in the recurrence tree (that represent contributions to the recurrence) we can compare their total cost against the corresponding asymptotic bounds. We'll focus on recurrences of the form:

$$W(n) = aW(n/b) + f(n)$$

where $W(1) = 1$.

- ☐ 1. (2 point) In `main.py`, you have stub code which includes a function `simple_work_calc`. Implement this function to return the value of $W(n)$ for arbitrary values of a and b with $f(n) = n$.
- ☐ 2. (2 point) Test that your function is correct by calling from the command-line `pytest main.py::test_simple_work` by completing the test cases and adding 3 additional ones.
- ☐ 3. (2 point) Now implement `work_calc`, which generalizes the above so that we can now input a, b and a *function* $f(n)$ as arguments. Test this code by completing the test cases in `test_work` and adding 3 more cases.
- ☐ 4. (2 point) Now, derive the asymptotic behavior of $W(n)$ using $f(n) = 1$, $f(n) = \log n$ and $f(n) = n$ with $a = 2$ and $b = 2$. Then, generate actual values for $W(n)$ for your code and confirm that the trends match your derivations.

TODO: your answer goes here

- ☐ 5. (4 points) Now that you have a nice way to empirically generate values of $W(n)$, we can look at the relationship between a, b, and $f(n)$. Suppose that $f(n) = n^c$. What is the asymptotic behavior of $W(n)$ if $c < \log_b a$? What about $c > \log_b a$? And if they are equal? Modify `compare_work` to compare empirical values for different work functions (at several different values of n) to justify your answer.

TODO: your answer goes here

- ☐ 6. (3 points) $W(n)$ is meant to represent the running time of some recursive algorithm. Suppose we always had a processors available to us and we wanted to compute the span of the same algorithm. Implement the function `span_calc` to compute the empirical span, where the work of the algorithm is given by $W(n)$. Implement `test_compare_span` to create a new comparison function for comparing span functions. Derive the asymptotic expressions for the span of the recurrences you used in problem 4 above. Confirm that everything matches up as it should.

TODO: your answer goes here