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 Github.
- Click on the assignment link posted on canvas and accept the assignment.
- Click on your personal github repository for the assignment (e.g., https://github.com/CMPS2200-Fall2021/recitation-01-your_username).
- Clone the repository to your local device
- Complete the lab task
- Add, commit, and push your completed lab back up to GitHub.
  - You will need to issue `git add` for all files that you have modified, e.g., `main.py`, `README.md`, and any others that you modify as well.
  - For example, on the command line, in the same directory as your cloned lab:

```
$ git add main.py
$ git commit -m "Implement Required Functions"
$ git push origin main
```

Running and testing your code

- You can run the tests using `pytest`. To install `pytest`, on your terminal:
  - `pip3 install pytest`
  - You may also have to install other python modules such as `tabulate` or other imported modules as you work through these recitations.
- It's usually best to run only one test at a time. To run tests, from the command-line, you can run
  - `pytest -s main.py` will run all tests
  - `pytest -s main.py::test_one` will just run `test_one`
- If you want to run your whole program, make sure to use `python3`. `python` still defaults to python version 2.

Turning in your work

- You may work with a partner to complete this recitation.
- Only one team member needs to push your completed lab to github.
- In the README.md file, include the names of the team members.

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 $f(n) = 1$, $f(n) = n$, and $f(n) = n^2$ 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 `test_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. In other words, assume that we always have enough processors for every generated subproblem. 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