CMPS 6100 Lab 04

In this lab, you will practice solving recurrences, then implement and analyze a couple of recursive algorithms.

Some prompts will require you to edit `main.py` and others will require answers to go in `answers.md`.

Refer back to the README.md for instruction on git, how to test your code, and how to submit properly to get all the points you've earned.

Recurrences (1 pt ea.)

Solve the following recurrences. Some are solveable using the brick method. For others, you must use the tree method.

1. $T(n) = T(n/2) + \lg n$
2. $T(n) = 3T(n/2) + n$
3. $T(n) = 2T(n/3) + 1$
4. $T(n) = 4T(n/16) + n^{1/4}$
5. $T(n) = T(n-1) + n$
6. $T(n) = T(\sqrt{n}) + 1$
7. $T(n) = 3T(n/3) + n$
8. $T(n) = 2T(n/2) + n^2$
9. $T(n) = 4T(n/2) + n^2$
10. $T(n) = 8T(n/2) + n^2$

The Master Method (5 pts)

11. The Master Method gives an easy formula for solving recurrences of the form:

$$T(n) = aT(n/b) + n^c$$

    Its three cases correspond to the relationship between $\log_b a$ and c. Derive the asymptotic behavior of $T(n)$ by solving its general recursion tree for each of the three cases. This problem is graded on based on your work shown for your derivations, not on the final results. Show your recursion tree and derivations from it.

    1. $\log_b a < c$
    2. $\log_b a = c$
    3. $\log_b a > c$

Coding Problems

Unimodal Max (5 pts)

12. Suppose you are given a unimodal list of n distinct elements. A unimodal list is one in which the elements are in increasing order up to its maximum element, then in decreasing order after that.

    For example:

    ```
    unimodal_list = [4, 7, 11, 18, 15, 13, 12, 9, 3]
    ```

Give an algorithm which runs in $O(\lg n)$ work to compute the maximum element of a unimodal array. Implement `unimodal_max(lst)` in `main.py` and ensure that it passes `test_unimodal_max`.

Hint: Use the Binary Search Algorithm as inspiration for your Unimodal Max Algorithm.

13. Prove that your **Unimodal Max** algorithm runs in $O(\lg n)$ work by deriving the recurrence for your algorithm and solving it.

Recursive List Sum (5 pts)

14. We can easily sum up the values of a list by iterating over them, adding each to a running sum. However, this algorithm has linear work and span. That is $W(n), S(n) \in \Theta(n)$.

    Like we did for the Search Problem in our parallelism notes, come up with divide and conquer recursive implementation for **List Sum**. Your algorithm should take in a list and return the sum of all of the elements in that list. Design your algorithm so that we could parallelize the recursive calls. Implement `list_sum_DC(lst)` in `main.py` and ensure that it passes `test_list_sum_DC`.

15. What is the work and span of your **List Sum** algorithm? Derive the work and span recurrences for your algorithm and solve them.