CMPS 6100 Lab 04

In this lab, you will practice solving recurrences, implement and analyze a recursive algorithm, and then empirically explore the runtime difference between two sorting algorithms. You will empirically benchmark Selection Sort and Merge Sort on lists of increasing sizes to build a sense of differences between their runtimes.

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

Coding Problems

Unimodal Max (3 pts)

9. 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. (3 pts)

   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.

10. Prove that your **Unimodal Max** algorithm runs in $O(\lg n)$ work by deriving the recurrence for your algorithm and solving it. (2 pts)

Empirical Sorting Analysis

Verifying Sorting Correctness

11. Both `selection_sort` and `merge_sort` have been provided for you along with unit tests for them. The unit tests depend on the function `is_sorted`. Implement this function by manually checking if the input list is sorted. Note: A list is sorted if, for every subsequent pair of elements, the latter is greater than or equal to the former. Once implemented, verify that all the tests pass.

    `test_is_sorted` is worth 2 pts.

Benchmarking the Sorting Algs

In order to benchmark `selection_sort` and `merge_sort` you will need to implement two functions: `get_sorting_time` and `benchmark_sorting_algs`.

12. Implement `get_sorting_time`. This function takes in `sort_fn`, the sorting function to use, and a `lst` to sort. It returns the number of milliseconds that it took to sort the list.

    Example usage:

    ```
    lst = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
    ss_time = get_sorting_time(selection_sort, lst)
    ms_time = get_sorting_time(merge_sort, lst)
    ```

13. Implement `benchmark_sorting_algs`. This function takes in a list of sizes, generates lists for each size in reverse order (see the doc comment for more details), and uses `get_sorting_time` to get selection sort and merge sorts run times on each list. It returns a list of tuples of the form `(n, selection_sort_time, merge_sort_time)`.

Analysis

Once you have the benchmarking code complete, let's see how Selection Sort and Merge Sort compare.

14. Use `print_results` to print a table of selection sort and merge sort's runtimes for each of the sizes:

```
`[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]`

**Include this table in `answers.md` (5 pts)**
```

15. Plot these values. Generate plots containing the runtimes of both Selection Sort and Merge Sort for different ranges of list sizes. A function to generate the plots is not given in this lab. Refer back to the Asymptotic Analysis notes for example code to produce plots. You will need to modofy it to work in this context, then plot the runtimes for the following lists of sizes.

    1. `[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]`
    2. `[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]`
    3. `[1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000]`

    Before running these, consider what you expect to see. Then, after generating them, answer question 6.

    Add your plots to `answers.md` (6 pts)

16. Do your results match your expectations? That is, are the trends that you see between the Selection Sort and Merge Sort's runtimes what you expected? Why or why not?

    Add your answer to `answers.md` (2 pts)

17. How long does it take your computer to sort lists of sizes `[1000, 10000, 100000]` using Selection Sort and Merge Sort? Based on this, how long would you expect Selection Sort to take to sort a list of size 1,000,000? Express this in minutes, hours, or days, which ever is the appropriate scale. Justify your answer. How long does it take Merge Sort to sort a list of 1,000,000?

    Add your answers to `answers.md` (2 pts)

Note:

Before pushing your final version to github, make sure that all instructions outside of your functions which will result in long runtimes are commented out. Otherwise, the tests on github will timeout and fail.