Recommended Homework Problems

Salaar Liaqat

Data Sciences Institute, UofT

1: Motivation and Big-O

- Cormen: Chapter 1 exercises
  - 1.2-1, 1.2-2, 1.2-3
- Bhargava: Chapter 1 exercises
  - 1.3 to 1.5
- Additional (for the mathematically inclined)
  - In CS, log is usually base 2, but a strong distinction is not made because *logs of different bases only differ by a constant factor* and constants are dropped in Big-O. Show this is true
  - Show that exponents of different bases **do not** differ by a constant factor

2: Data Structures, Searching, and Sorting

- Bhargava: Chapter 5
  - 5.1 to 5.4
- Give examples of 2 situations to use a queue and 2 situations to use a stack
- In Python, code a `stack` class with `is_empty`, `push`, and `pop` methods using the end of a Python list as the top of the stack. Bonus: Compare the run time of using the start of the list versus the end of the list as the top of the stack using the `timeit` library!
- In Python, code a `binary_search` function.
- In Python, code a `hash_table` that can hash 4 values.

3: Recusion

- Bhargava: Chapter 4 exercises
  - 4.1 to 4.8
- Write a recursive function that produces the `RecursionError: maximum recursion depth exceeded` error.
- Write a iterative function to calculate the *n*th Fibonacci number. What is its time and space complexity?
- Write a recursive function to determine if a string is a palindrome. What is its time and space complexity?
- Write a recursive function to check if a given positive integer is a prime number. What is its time and space complexity?

3: Recusion

- Suppose you have a plot of land and want to divide the land into even square plots, while keeping the plots as big as possible. How would you do this using D&C? See Bhargava pg. 52.
- Explain why the "merge" step in mergesort is $O(n)$
- Implement mergesort. You might find using helper functions useful.
- Write a recursive function to perform binary search on a sorted list

4: Recursive Data Structures

- Cormen: Chapter 10 exercises
  - 10.3-1, 10.3-2, 10.3-3
- Bhargava: Chapter 6 exercises
  - 6.1 to 6.5

4: Recursive Data Structures

- Implement preorder, postorder, and level order traversal. Determine the time and space complexity in each case
- Implement a function that find an element in a BST and deletes it. The descendants of the deleted node are given to the deleted node's parent. (Hard)
- Using the `graph` class from the slides, implement BST search such that it stops and tell you the distance the node is from the starting point.
  - For instance, if we searched for 7 in the graph given in the slides, it would return `"Found! Distance 2"`.
  - If we searched for 100 in the graph, it would return `"Not found!"`
- Implement postorder graph traversal using the `graph` class from the slides.
- Implement a function using recursion to find the sum heterogeneous nested lists such as [[1, [2]], [[[3]]], 4, [[5, 6], [[[7]]]]].

5: Optimization

- Bhargava: Chapter 9 exercises
  - 9.1, 9.2
  - Read the knapsack problem FAQs on page 171
  - Follow the example about longest common substring on page 178

5: Optimization

- Write the code to brute force the diet problem. Compare the run times using the `timeit` library.
- Modify the code from the slide such that there is an upper bound for calories and vitamins.
- Page 17 of Bhargava covered the travelling sales person problem. Is it possible to improve the proposed solution using any method we learned today?

6: Slow Code

- Bhargava: Chapter 8 exercises
  - 8.1 - 8.8
- Vectorize the second code chunk in the Parallelization section of the lesson 6 slides
- Find the longest palindrome from a string Hint: use a greedy alogrithm
- Computing Pascal's triangle Hint: use dynamic programming

References

- Bhargava, A. Y. (2016). *Grokking algorithms: An illustrated guide for programmers and other curious people.* Manning. Chapter 1.
- Cormen, T. H. (Ed.). (2009). *Introduction to algorithms* (3rd ed). MIT Press. Chapter 1 and 3.