The CSC148 Mega FSG

Ibrahim Chehab and Helia Seyedmazhari

UTM RGASC

April 9th, 2024

Table of Contents

Welcome to the CSC148 Mega FSG!

Welcome everyone, to the first (I think?) ever CSC148 *Mega FSG!*+

- Thank you for coming! We hope you enjoy the FSG!
- Today we will be going over some key concepts from CSC148, and we will be doing some practice problems in preparation for the exam.

Key Terms

We will now quickly go over all the key terms you need to know for the exam.
Note:

- This is **NOT** an exhaustive or comprehensive list by any means. When in doubt, always check with a TA/Professor.
- We will be going backwards, here's why:
    - All too often, I (Ibrahim) see people neglecting weeks 1-6 and focusing on the later weeks. This is a mistake.
    - Like all CS courses, everything builds on top of each other. If you don't understand the basics, you won't understand the more complex stuff.
    - We will be showing you how the content from previous weeks builds upon the more complex stuff, to hopefully emphasize the importance of understanding the basics.

Week 12 Key Terms

- **Big-O Notation ($\mathcal{O}(n)$)**
- **Big-Theta Notation ($\Theta(n)$)**

Week 11 Key Terms

- Divide and Conquer
  - Recursion

Week 11 Key Terms

- **Divide and Conquer**
    - **Recursion**
- **QuickSort**
    - *Partitioning*
    - *Pivot*
    - *How the Pivot can affect the efficiency of the algorithm*
    - *Why does Quicksort performance vary?*

Week 11 Key Terms

- **Divide and Conquer**
  - **Recursion**
- **QuickSort**
  - *Partitioning*
  - *Pivot*
  - *How the Pivot can affect the efficiency of the algorithm*
  - *Why does Quicksort performance vary?*
- **MergeSort**
  - *Merging*
  - *Why Mergesort has consistent performance*

Week 8, 9, 10 Key Terms

- **Root**
- **Subtree**
- **Branching Factor**
- **Height**
- **All Tree Traversals**
- **BSTs and why they're special**
  - *All time complexities of BST operations*
  - *How $\mathcal{O}(h)$ translates in terms of worst-case and best-case scenarios*
- **Polymorphism**
  - **Why?** Expression Trees abuse Polymorphism to exist

Week 6, 7 Key Terms

- List Comprehension
- Recursion

Week 5, 4, 3, 2, 1 Key Terms

- **Inhertiance**
  - *How it relates to Polymorphism*
- **Abstraction**
- **Stacks and Queues**
  - *How the implementation changes the time complexity*
  - *Why Stacks and Queues are important*
  - **Why** *they're callled* **Abstract** *data types*
- **Linked Lists**

Practice Problem 1: Did Somebody Say Palindrome?

Did Somebody Say Palindrome?
Implement a *recursive* function that checks whether a given string is a palindrome.

RESTRICTIONS:

i. This function **MUST** be implemented using recursion.
ii. This function must **NOT** mutate the original word/sentence.
iii. You may use slicing, but you may **NOT** use the built-in reversal `[::-1]` or the `reversed()` function.

Here are some examples:

(a) `ewe`
(b) `anna`
(c) `borrow or rob`
(d) `taco cat`
(e) `was it a car or a cat i saw`
(f) `racecar`

Hint: Recall Ibrahim's recursion analogy

Practice Problem 2: InsertMii

InsertMii

Consider the following implementation of a Doubly Linked List:

```
class DLLNode:
    """A node in a linked list."""
    item: Any
    next: Optional[DLLNode]
    prev: Optional[DLLNode]

class DoublyLinkedList:
    """A doubly linked list."""
    _first: Optional[DLLNode]
    _last: Optional[DLLNode]

    # Implementation omitted
```

Practice Problem 2: InsertMii (Cont'd)

Implement the following method in the `DoublyLinkedList` class:

```
def insert_last(self, value: Any, after: Any) -> bool:
    """Insert a new Node with the value <value> after the LAST
   occurrence of the value <after> in this list.
    If <after> does not exist in the list, then do not insert
   anything and return False.
    The list must be correctly linked after this operation.
    >>> sl = CustomDLL([7, 2, 7, 3])
    >>> str(sl)
    '7 2 7 3'
    >>> sl.insert_last(5, 7)
    True
    >>> str(sl)
    '7 2 7 5 3'
    >>> sl.insert_last(9, 8)
    False
    >>> str(sl)
    '7 2 7 5 3'
    """
```

Practice Problem 3: The Even-Worse-Stack

Nugget has entered their *evil era* and designed an evil ADT known as the `EvenWorseStack`. They've subjected Therapist to the `EvenWorseStack` and now Therapist is in a state of despair. Help Therapist by analyzing the time complexity of the pop method of the `EvenWorseStack` class.

```python
class EvenWorseStack:
    """
    A Stack implementation designed to be slow and
        inefficient.
    """
    _stack: Queue

    def __init__(self) -> None:
        self._stack = Queue()

    def push(self, value: int) -> None:
        self._stack.enqueue(value)

    def pop(self) -> int:
        temp = Queue()
        while self._stack.size() > 1:
            temp.enqueue(self._stack.dequeue())
        value = self._stack.dequeue()
        self._stack = temp
        return value
```

```python
class Queue:

    _queue: list[int]

    def __init__(self) -> None:
        self._queue = []

    def enqueue(self, value: int) -> None:
        self._queue.insert(0, value)

    def dequeue(self) -> int:
        index_to_remove = self.size() - 1
        value = self._queue[index_to_remove]
        self._queue = self._queue[:index_to_remove]
        return value

    def size(self) -> int:
        return len([i for i in self._queue])
```

What is the time complexity of the pop method?

Practice Problem 4: Efficiencii

Efficiencii

Select all the statements that are **TRUE**:

1. Choosing n_0 does not change the final result of the efficiency class
2. If a function has a time complexity of $\mathcal{O}(n^2)$, it might still be $\Theta(n)$
3. If a function has a time complexity of $\mathcal{O}(n)$, it might still be $\Theta(n^2)$
4. The iterative part of `QuickSort` is faster than the iterative part of `MergeSort`
5. The recursive part of `QuickSort` is faster than the recursive part of `MergeSort`
6. If a function is $\mathcal{O}(g(n))$, then it is also $\Theta(g(n))$
7. If a function is $\Theta(g(n))$, then it is also $\mathcal{O}(g(n))$

The Final Challenge. . .

Symbolab from Ohio

Peace and d.aki have been trying to get an internship at Symbolab, and they have been given an *at-home assignment* to complete. The assignment is to implement a primitive derivative calculator in Python from scratch. Help them out by:

- Designing classes that adhere to the *Class Design Recipe* that represent important elements of a derivative.
- Using *Polymorphism and Inheritance* to represent the different types of derivatives.
- Using *Trees and Recursion* to represent the structure of the derivative.
- Implementing a `derivative` method that takes a function and returns its derivative.

Conclusion

Thank you for attending the CSC148 Mega FSG!

- We hope you enjoyed the FSG!
- We hope you learned something new!
- We hope you're ready for the exam!

Thank you for your continued support and participation!