CS-UY 1134 Lab 2 Summer 2023

Lab 2 Overview

- This lab reviews asymptotic analysis, searching, and recursion.
- Assumes review of chapters 3 and 4 of the textbook. Refer to the text and lecture notes as needed.
- Think before you code. Consider test cases that might cause failure.
- Stay for the duration of the lab. Help others if you finish early; complete unfinished work outside of lab time.
- TAs are available for questions during lab and office hours.

Part A: Asymptotic Analysis and Searching (2:00 PM - 3:50 PM)

Vitamins (30 minutes)

Big-O proof: If there exist constants c and n_0 such that $f(n) \leq c \cdot g(n)$ for every $n \geq n_0$, then $f(n) = O(g(n))$.

Big-Θ proof: If there exist constants c_1, c_2, and n_0 such that $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for every $n \geq n_0$, then $f(n) = \Theta(g(n))$.

Big-Ω proof: If there exist constants c and n_0 such that $f(n) \geq c \cdot g(n)$ for every $n \geq n_0$, then $f(n) = \Omega(g(n))$.

1. Use the formal proof of big-O and big-Θ to show the following (10 minutes):
   - $3n^3 + 2n - 1$ is $O(n^3)$
   - $\frac{2n^3+n}{n^2+1}$ is $O(n)$
   - $2n^2 + 7n + 1$ is $\Theta(n^2)$
2. State True or False and explain why for the following (5 minutes):
   - $5n^2 \log(n)$ is $O(n^3)$
   - $5n^2 \log(n)$ is $\Theta(n^2 \log n)$

3. For each $f(n)$, write the summation results and provide a tight bound $\Theta(f(n))$ (5 minutes). Given n numbers:

   - $2 + 2 + 2 + \cdots + 2 =$ _____ $= \Theta($_____$)$
   - $n^2 + n^2 + n^2 + \cdots + n^2 =$ _____ $= \Theta($_____$)$
   - $n + 2n + 3n + \cdots + n^2 =$ _____ $= \Theta($_____$)$

   Given $\log_2(n)$ numbers, where n is a power of 2:

   - $1 + 4 + 16 + \cdots + n^2 =$ _____ $= \Theta($_____$)$
   - $n^2 + n^2/4 + n^2/16 + \cdots + 1 =$ _____ $= \Theta($_____$)$
   - $1 + 2 + 3 + \cdots + \log_2(n) =$ _____ $= \Theta($_____$)$

4. For each code snippet, find $f(n)$ for which the algorithm's time complexity is $\Theta(f(n))$ in its worst-case run and explain why (10 minutes).

```
a) def func(lst):
    for i in range(len(lst)//2):
        if (lst[i] == 0):
            return
```

```
b) def func(lst):
    for i in range(len(lst)):
        if (lst[i] > 10):
            print("i =", i)
        else:
            return
```

```
c) def func(lst):
    for i in range(len(lst)):
        for j in range(i,len(lst)):
            if (i*j) in lst:
                print("i*j = ", i*j)
```

```
d) def func(n):
    for i in range(int(n**(0.5))):
        for j in range(i):
            if (i+j) > n:
                print("i+j = ", i+j)
```

```
e) def func(n):
    for i in range(n):
        for j in range(n//2):
            print("i+j = ", i+j)
```

Coding (45 minutes)

1. Write a function that rotates a list by k positions to the left in $\Theta(n)$ time. Do not create a new list, and do not use list methods like `pop` or `append`.

```
def rotate_list(lst, k):
    """
    : lst type: list[]
    : k type: int
    : return type: None
    """
```

2. Write a function to move all even numbers to the beginning of a list of integers, maintaining the order of even and odd numbers. The solution must be in-place and run in $\Theta(n)$ time. Do not use list methods like `pop` or `append`.

```
def move_evens(nums):
    """
    : nums type: list[int]
    : return type: None
    """
```

3. Write a function that performs binary search on a sorted list of unique integers and returns the index of a given integer, or -1 if not found. The solution must not modify the list and must run in $\Theta(\log(n))$ time.

```
def binary_search_integer(lst, num):
    """
    : lst type: list[int]
    : num type: int
    : return type: int
    """
```

Part B: Recursion (4:00 PM - 5:50 PM)

Coding (65 minutes)

1. Write a recursive function that returns the product of all numbers from 1 to n.

```
def product_to(n):
    """
```

```
    : n type: int
    : return type: int
    """
```

2. Give a recursive implementation for a linear search algorithm.

```
def linear_search(lst, low, high, val):
    """
    : lst type: list[int]
    : val type: int
    : low type, high type: int
    : return type: int
    """
```

3. Write a recursive function to find the minimum element in a non-empty, non-sorted list of numbers.

   - Determine the runtime of the following implementation:

```
def find_min(lst):
    if len(lst) == 1:
        return lst[0]
    #base case
    prev = find_min(lst[1:])
    if prev < lst[0]:
        return prev
    return lst[0]
```

   - Update the function parameters to include `low` and `high` to specify the range of indices to consider. The runtime must be linear.

```
def find_min(lst, low, high):
    """
    : lst type: list[int]
    : low, high type: int
    : return type: int
    """
```

4. Write a recursive function that returns a tuple containing the number of uppercase and lowercase letters in a given string. The runtime must be linear.

```
def uplow_count(word, low, high):
    """
```

```
    : word type: str
    : low, high type: int
    : return type: tuple (int, int)
    """
```

5. Write a recursive function that checks if a given string is a palindrome (reads the same forwards and backward), ignoring spaces and punctuation.

```
def is_palindrome(text):
    """
    : text type: str
    : return type: bool
    """
```

6. Write a recursive function that calculates the nth Fibonacci number.

```
def fibonacci(n):
    """
    : n type: int
    : return type: int
    """
```

7. Write a recursive function that reverses a string.

```
def reverse_string(text):
    """
    : text type: str
    : return type: str
    """
```

Vitamins (15 minutes)

Trace the execution and analyze the running time of each code snippet, including drawing the recursion tree and determining the asymptotic runtime.

```
a. def func1(n):
    # Draw out func1(16)
    if (n <= 1):
        return 1
    else:
        return 2 * func1(n-1)
```

b.
```
def func2(n):
    # Draw out func2(16)
    if (n <= 1):
        return 10
    else:
        return 1 + func2(n//4)
```

c.
```
def func3(lst) # Draw out func3([1, 2, 3, 4, 5, 6, 7, 8])
    if (len(lst) == 1):
        return lst[0]*2
    else:
        return lst[0] + func3(lst[1:])
```