Spring 2023

[EC2202] Data Structures

Midterm: 1 pm, Tuesday, Apr. 18

INSTRUCTIONS

- **Do not open your exam sheet** until directed to do so, otherwise you would be considered cheating.
- You have **1 hour and 50 minutes** to complete the exam (7 problems; maximum of 100 points).
- The exam is closed book, closed notes, closed computer, and closed calculator.
- Mark your answers on the exam sheet itself and be sure to **write your answers in the space (boxes) provided**. We will not consider the answers given outside the boxes (use other space for brainstorming).

POLICIES & CLARIFICATIONS

- If you need to use the **restroom**, bring your exam sheet to the back of the room. Only one person is allowed at a time.
- You may use built-in Python functions that do not require import, such as min, max, pow, len, and abs.
- For **What Would Python Print (WWPP) problems**, write the results that would show up in the Colab environment.
- For **coding problems**, we will ignore minor grammar mistakes for evaluation (focus on the logic and flow). Moreover, your code must be indented correctly.
- Use **English** for your answers and name.

Student ID Number	Name

Q1. (15 points) What Would Python Print (WWPP)

For the code blocks given on the left, write what Python would print in the box on the right. For your answers, include ***all the resulting print outputs*** in ***the correct sequential order***. Assume that you are running the code blocks in the Colab environment.

(a) **(3 points)**

```
def func_1():
  print("EC2202")
  return 1

def func_2():
  print("I love GIST")

print(func_1(), func_2())
```

```
EC2202
I love GIST
1 None
```

(b) **(3 points)**

```
higher_order = lambda f: lambda x: f(x)
g = lambda x: x * x
print(higher_order(2)(g))
```

```
TypeError
```

(c) **(3 points)**

```
print(0 or len or True)
print(-1 and True and 2 and 1 / 0)
```

```
<built-in function len>
ZeroDivisionError
```

(d) **(3 points)**

```
lst = [5, 6, 7, 8]
lst += list(range(7, 11))
lst.pop(7)
lst.pop(6)
print(lst)
```

```
[5, 6, 7, 8, 7, 8]
```

(e) **(3 points)**

```
z = [[1, 2, 3, 4], [5, 6, 7, 8], \
     [9, 10, 11, 12]]
print([x for y in z if sum(y) > 10 \
       for x in y if x < 10])
```

```
[5, 6, 7, 8, 9]
```

Q2. (10 points) Functions

(a) (**5 points**) Implement the following function that checks if a given string is a palindrome. A palindrome is a string that remains identical when reversed. For one line (Pythonic) implementations, ***5 points*** will be given and correct implementations using more than one line will receive ***3 points***.

```
def is_palindrome(strings):
    '''
    >>> is_palindrome("tenet")
    True
    >>> is_palindrome("tenets")
    False
    >>> is_palindrome("raincar")
    False
    >>> is_palindrome("")
    True
    >>> is_palindrome("a")
    True
    >>> is_palindrome("ab")
    False
    '''
```

```
# single line implementation
return strings == strings[::-1]

# multi-line implementation
for i in range(0, int(len(strings)/2)):
  if strings[i] != str[len(strings)-i-1]:
    return False
return True
```

(b) (**5 points**) Implement the below function which takes in a number `n` and a sequence of digits `seq`. The function returns whether `n` contains `seq` as a subsequence, which does not have to be consecutive. For example, `141` contains the sequence `11` because the first digit of the sequence, `1`, is the first digit of `141`, and the next digit of the sequence, `1`, is found later in `141`.

```
def has_subseq(n, seq):
  '''
  >>> has_subseq(123, 12)
  True
  >>> has_subseq(141, 11)
  True
  >>> has_subseq(144, 12)
  False
  >>> has_subseq(144, 1441)
  False
  >>> has_subseq(1343412, 134)
  True
  '''
```

```
  if n == seq:
    return True
  if n < seq:
    return False
  without = has_subseq(n // 10, seq)
  if seq % 10 == n % 10:
    return has_subseq(n // 10, seq // 10) or without
  return without
```

Q3. (10 points) Clockwise Tower of Hanoi

Consider the following variant of the towers of Hanoi problem. As usual, we are given three pegs A, B, C. At the start, there are disks on peg A. The problem is to move them all to peg B. However, this time a disk can ***only move clockwise***: from peg A to B, from peg B to C, or from peg C to A. ***All other moves are not allowed***. For instance, it is not allowed to move a disk directly from peg A to peg C. Of course all the other rules are still valid: You can only move the top disk on each pole, and a disk is not allowed to lie on top of a smaller disk.

```python
def hanoi_clockwise(n, source, destination, spare):
  '''prints the order of moving disks
  >>> hanoi_clockwise(2, 'A', 'B', 'C')
  Move disk 1 from A to B
  Move disk 1 from B to C
  Move disk 2 from A to B
  Move disk 1 from C to A
  Move disk 1 from A to B
  '''
  if n == 1:
    print("Move disk 1 from %s to %s" % (source, destination))
  else:
    # implement this part
```

```python
    hanoi_cw(n-1, source, destination, spare)
    hanoi_cw(n-1, destination, spare, source)
    print("Move disk %d from %s to %s" % (n, source, destination))
    hanoi_cw(n-1, spare, source, destination)
    hanoi_cw(n-1, source, destination, spare)
```

Q4. (10 points) Algorithm Analysis

(a) **(5 points)** For the group of functions below, sort the functions in increasing order of asymptotic (big-Θ) complexity.

$$f_1(n) = n^{n+4} + n!$$

$$f_2(n) = n^{7\sqrt{n}}$$

$$f_3(n) = 4^{3n \log n}$$

$$f_4(n) = n^{12 + 1/n}$$

$$f_5(n) = 7^{n^2}$$

Answer: 4 -> 2 -> 1 -> 3 -> 5

Take the logarithms of the functions: $\Theta(n \log n)$, $\Theta(\sqrt{n} \log n)$, $\Theta(n \log n)$, $\Theta(\log n)$ and $\Theta(n^2)$. For f1 and f3, $f_3 = (4^{\log n})^{3n} = (n^{\log 4})^{3n} = n^{6n}$. Therefore, f3 is bigger than f1.

(b) **(5 points)** Evaluate the runtime bound in Θ for the code below. Assume the number of elements in `arr` is N and the `deepcopy` method takes Θ(M) time, where M is the number of elements copied.asaZZZham

```
import copy

def silly(arr):
  if len(arr) <= 1:
    print("You won!")
    return

  new_len = len(arr) // 2
  first_half = copy.deepcopy(arr[:new_len])
  second_half = copy.deepcopy(arr[new_len:])
  silly(first_half)
  silly(second_half)
```

$\Theta(n \log n)$

Q5. (20 points) Arrays and Strings

(a) **(5 points)** Implement the function `max_sub_sum_linear` that finds the maximum subsequence sum in ***linear time***. Implementations that require more than O(N) complexity will not receive a point.

```python
def max_sub_sum_linear(seq):
    '''
    Find the maximum subsequence sum in linear time!
    >>> max_sub_sum_linear([-2, -3, 4, -1, -2, 1, 5, -3])
    7
    >>> max_sub_sum_linear([4, -3, 5, -2, -1, 2, 6, -2])
    11
    >>> max_sub_sum_linear([1, -1, 2, -2, 3, -3, 4, -4])
    4
    >>> max_sub_sum_linear([3, 5, 7, 10, -26, 30])
    30
    >>> max_sub_sum_linear([-1])
    -1
    '''
```

```python
    seq_len = len(seq)
    max_so_far = seq[0]
    curr_max = seq[0]
    for i in range(1, seq_len):
      curr_max = max(seq[i], curr_max + seq[i])
      max_so_far = max(max_so_far, curr_max)
    return max_so_far
```

(b) **(15 points)** Given an N x N 2D matrix `mat` representing an image, `rotate_matrix` rotates the image by 90 degrees (anti-clockwise). You need to do this in place. Note that you should not create an additional array.

```python
# Function to print the matrix
def print_matrix(mat, size):
 for i in range(0, size):
   for j in range(0, size):
     print (mat[i][j], end = ' ')
   print ("")

# You just need to implement this function
def rotate_matrix(mat, size):
 '''
 >>> mat = [[1, 2, 3],
 ...        [4, 5, 6],
 ...        [7, 8, 9]]
 >>> rotate_matrix(mat, 3)
 >>> print_matrix(mat, 3)
 3 6 9
 2 5 8
 1 4 7
 >>> mat = [[1, 2],
 ...        [4, 5]]
 >>> rotate_matrix(mat, 2)
 >>> print_matrix(mat, 2)
 2 5
 1 4
 '''
```

```python
 for x in range(0, int(size / 2)):
   # Consider elements in group of 4 in current square
   for y in range(x, size-x-1):
     # store current cell in temp variable
     temp = mat[x][y]

     # move values from right to top
     mat[x][y] = mat[y][size-1-x]

     # move values from bottom to right
     mat[y][size-1-x] = mat[size-1-x][size-1-y]

     # move values from left to bottom
     mat[size-1-x][size-1-y] = mat[size-1-y][x]

     # assign temp to left
     mat[size-1-y][x] = temp
```

Q6. (20 points) Queues with a Single Sentinel

Complete the implementation of the `Queue` class using ***a single sentinel***. A sentinel is a dummy node that does not contain meaningful items; the sentinel makes the implementation clean by removing the necessity of checking exceptional cases. The `Queue` starts with a single sentinel and becomes ***circular*** after a few enqueue operations (the last node of the `Queue` points back to the sentinel node). Correct implementation of each method is worth ***5 points***.

```python
class Node:
  def __init__(self, item, prev=None, next=None):
    self.item = item
    self.prev = prev
    self.next = next

class Queue:
  '''
  >>> q = Queue()
  >>> q.is_empty()
  True
  >>> q.front()
  'Q is empty!'
  >>> q.enqueue(4)
  >>> q.front()
  4
  >>> q.is_empty()
  False
  >>> q.enqueue(5)
  >>> q.enqueue(6)
  >>> q.enqueue(7)
  >>> q.dequeue()
  4
  >>> q.front()
  5
  >>> print(q)
  5 -> 6 -> 7
  >>> q.dequeue()
  5
  >>> q.dequeue()
  6
  >>> q.dequeue()
  7
  >>> q.dequeue()
  'Q is empty!'
  >>> q.is_empty()
  True
  '''
  def __init__(self):
    self.sentinel = Node(None)
    self.sentinel.next = self.sentinel
    self.sentinel.prev = self.sentinel
```

```python
def __str__(self):
    result = []
    temp = self.sentinel.next
    while temp != self.sentinel:
      result.append(str(temp.item))
      temp = temp.next
    return " -> ".join(result)
```

```python
def is_empty(self):
    return self.sentinel.next is self.sentinel
```

```python
def front(self):
  if self.is_empty():
    return "Q is empty!"
  return self.sentinel.next.item

def dequeue(self):
  if self.is_empty():
    return "Q is empty!"
    item = self.sentinel.next.item
    self.sentinel.next = self.sentinel.next.next
    self.sentinel.next.next.prev = self.sentinel
    return item
```

```python
def enqueue(self, x):
    temp = Node(x)
    temp.prev = self.sentinel.prev
    temp.next = self.sentinel

    self.sentinel.prev.next = temp
    self.sentinel.prev = temp
```

Q7. (15 points) Binary Search

There is an integer array `nums` sorted in ascending order (with distinct values). Prior to being passed to `search`, `nums` is possibly rotated at an unknown pivot index k (1 <= k < len(`nums`)) such that the resulting array is [`nums`[k], `nums`[k+1], ..., `nums`[n-1], `nums`[0], `nums`[1], ..., `nums`[k-1]] (0-indexed). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2]. Given the array `nums` after the possible rotation and an integer `target`, `search` returns the index of `target` if it is in `nums`, or -1 if it is not in `nums`. Moreover, you should take the approach described below rather than what we've discussed in the lecture; complete the implementation of *Step 2*. The expected runtime complexity is O(log N).

```
def search(nums, target):
  '''
  >>> search([4, 5, 6, 7, 0, 1, 2], 0)
  4
  >>> search([4, 5, 6, 7, 0, 1, 2], 3)
  -1
  >>> search([1], 0)
  -1
  '''
  # Step 1: find the pivot using binary search (O(log N))
  left, right = 0, len(nums) - 1
  while left < right:
    mid = left + (right - left) // 2
    if nums[mid] > nums[right]:
      left = mid + 1
    else:
      right = mid
  pivot = left

 # Step 2. binary search for finding the target (O(log N))
```

```
  left, right = 0, len(nums) - 1
  while left <= right:
    mid = left + (right - left) // 2
    real_mid = (mid + pivot) % len(nums)
    if nums[real_mid] < target:
      left = mid + 1
    elif nums[real_mid] > target:
      right = mid - 1
    else:
      return real_mid
  return -1
```