CSCI046 Final, Spring 2021

Collaboration policy:

You may not:

1. discuss the exam with any human other than Mike; this includes:

   (a) asking your friend for clarification about what a problem is asking

   (b) asking your friend if they've completed the exam

You may:

1. take as much time as needed

2. use any written notes / electronic resources you would like

3. ask Mike to clarify questions via email

Name:

Problem 1. (3 pts) Complete each equation below by adding the symbol O if $f = O(g)$, Ω if $f = \Omega(g)$, or Θ if $f = \Theta(g)$. The first row is completed for you as an example.

You will lose one point for each incorrect answer.

f(n)		g(n)
1	$=$	$O(n)$
n^2	$=$	$\Omega \quad n^3$
$1/n$	$=$	$\Omega \quad n^{-2}$
$\frac{n}{\log n}$	$=$	$O \quad \left(\frac{n}{\log n}\right)^2$
2^n	$=$	$O \quad 3^n$
$\log(2^n)$	$=$	$\Theta \quad \log(3^n)$
$\log n$	$=$	$O \quad n^{0.0001}$
$\log(3n)$	$=$	$\Theta \quad \log(4n)$
n	$=$	$\Theta \quad 4n$
2^n	$=$	$\Omega \quad n^2$

Problem 2. (20 pts) For each question below, circle either True or False. Each correct answer will result in +1 point, each incorrect answer will result in -1 point, and each blank answer in 0 points.

1. **TRUE**   False   Any procedure with worst-case runtime $\Theta(n^2)$ is guaranteed to have best case runtime $O(n^2)$.

2. True   **FALSE**   TimSort is asymptotically faster than Merge sort on worst case input.

3. **TRUE**   False   TimSort is asymptotically faster than insertion sort on worst case input.

4. **TRUE**   False   When Python's built-in `sorted` function is called on a list, it returns a new list object and leaves the original list unchanged.

5. True   **FALSE**   Binary search can be efficiently implemented using Python's built-in `deque` data structure.

6. **TRUE**   False   Tuples cannot be modified in python after they are created.

7. **TRUE**   False   Python's built-in `enumerate` function allocates $\Theta(1)$ memory.

8. True   **FALSE**   Python's built-in `range` function is implemented natively in python using the `yield` keyword.

9. **TRUE**   False   In Python 3, all classes that implement the `__hash__` magic method are assumed to be immutable.

10. **TRUE**   False   The worst case runtime for inserting into a Binary Heap is better than the worst case runtime for inserting into a BST.

11. True   **FALSE**   Python sets implement the `__hash__` magic method.

12. **TRUE**   False   The height of every binary heap is guaranteed to be $\Theta(\log n)$.

13. **TRUE**   False   The height of every AVL tree is guaranteed to be $\Theta(\log n)$.

14. True   **FALSE**   In Python 3, a `deque` variable can be used as the key for a dictionary.

15. **TRUE**   False   An in-order traversal of a `BST` will traverse the elements in sorted order.

16. True   **FALSE**   The Linux kernel is licensed using the GPL *version 3*.

17. **TRUE**   False   If you find a github repo using the BSD3 license, you may legally use code from this repo in a GPL licensed project.

18. **TRUE**   False   Given the string "César Chávez", an NFC-normalized UTF-8 encoding will require fewer bytes than a NFD-normalized UTF-8 encoding.

19. **TRUE**   False   Given any string in NFC form, normalizing to NFD and back to NFC is guaranteed to be an idempotent operation (i.e. you will get the same string back.)

20. **TRUE**   False   Given any string in NFKD form, normalizing to NFD and back to NFKD is guaranteed to be an idempotent operation (i.e. you will get the same string back.)

Problem 3. Consider the following three functions:

```python
1  def print_container_1(xs):
2      for i in range(len(xs)):
3          print(xs.pop())
4
5  def print_container_2_list(xs):
6      for i in range(len(xs)):
7          print(xs.pop(0))
8
9  def print_container_2_deque(xs):
10     for i in range(len(xs)):
11         print(xs.popleft())
```

Notice that the first function will work whether the `xs` parameter is a list or a deque (i.e. it is *polymorphic*). The other two functions, however, are not polymorphic and will throw errors when run with the wrong input type.

1. (1 pt) Assume we have a **list** with **n** elements. What is the runtime of `print_container_1` on this list? Your answer must be tight and in big-O notation.

   $O(n)$

2. (1 pt) Assume we have a **list** with **n** elements. What is the runtime of `print_container_2_list` on this list? Your answer must be tight and in big-O notation.

   $O(n^2)$

3. (1 pt) Assume we have a **deque** with **n** elements. What is the runtime of `print_container_1` on this deque? Your answer must be tight and in big-O notation.

   $O(n)$

4. (1 pt) Assume we have a **deque** with **n** elements. What is the runtime of `print_container_2_deque` on this deque? Your answer must be tight and in big-O notation.

   $O(n)$

Problem 4. The code below is my merge sort implementation for the week 5 homework. Recall that we analyzed the runtime of merge sort when the input variable `xs` was a **list** of length n, and showed that the runtime was $\Theta(n \log n)$. In this problem, we will compute the runtime when the input variable `xs` is a **deque**.

```
1   def merge_sorted(xs, cmp=cmp_standard):
2       if len(xs) <= 1:
3           return xs
4       else:
5           mid = len(xs) // 2
6           left = xs[:mid]
7           right = xs[mid:]
8           return _merged(
9               merge_sorted(left, cmp),
10              merge_sorted(right, cmp),
11              cmp
12              )
13
14  def _merged(xs, ys, cmp=cmp_standard):
15      zs = []
16      i = 0
17      j = 0
18      while i < len(xs) and j < len(ys):
19          if cmp(xs[i], ys[j]) == -1:
20              zs.append(xs[i])
21              i += 1
22          else:
23              zs.append(ys[j])
24              j += 1
25      while i < len(xs):
26          zs.append(xs[i])
27          i += 1
28      while j < len(ys):
29          zs.append(ys[j])
30          j += 1
31      return zs
```

Note that lines 6 and 7 above take a slice of the `xs` variable. The built-in `collections.deque` class does not by default support the ability to take slices, and so the code above will generate an error message. It is possible, however, to efficiently add the ability to take slices to the `deque` class. (You do not need to understand the details of how to do this, buth this stackoverflow link explains them: `https://stackoverflow.com/questions/10003143/how-to-slice-a-deque`.) For this problem, you should assume that the above technique has been used to make the input deque slicable. The runtime of computing the slice is $O(n)$, where n is the size of the deque.

1. (2 pts) Compute the recurrence relation that describes the runtime of the `merge_sorted` function when the input variable `xs` is a **deque**.

> **Solution:**
> $$T(n) = 2T(n/2) + n^2$$
> The key observation is that `_merged` takes time $O(n^2)$ because it indexes into a deque.

2. (2 pts) Solve the recurrence relation in part 1 above, and report your answer in Θ notation. Note that if you do not get the answer to part 1 above correct, you can get partial credit on this problem, but you will not get full credit.

> **Solution:**
> $$\Theta(n^2)$$

Problem 5. In the questions below, assume that all data structures from the homeworks are correctly implemented.

1. (2 pts) What is the worst-case runtime of the `heap_sorted` function defined below in terms of the length n of the input list? (Use Θ notation.)

```
1  from containers.Heap import Heap
2
3  def heap_sorted(xs):
4      heap = Heap(xs)
5      ret = []
6      while len(heap) > 0:
7          ret.append(heap.find_smallest())
8          heap.remove_min()
9      return ret
```

$\Theta(n \log n)$

2. (2 pts) What is the runtime of the code below in terms of n? (Use Θ notation.)

```
1  from containers.BST import BST
2
3  bst = BST()
4  for i in range(n**2):
5      bst.insert(i)
```
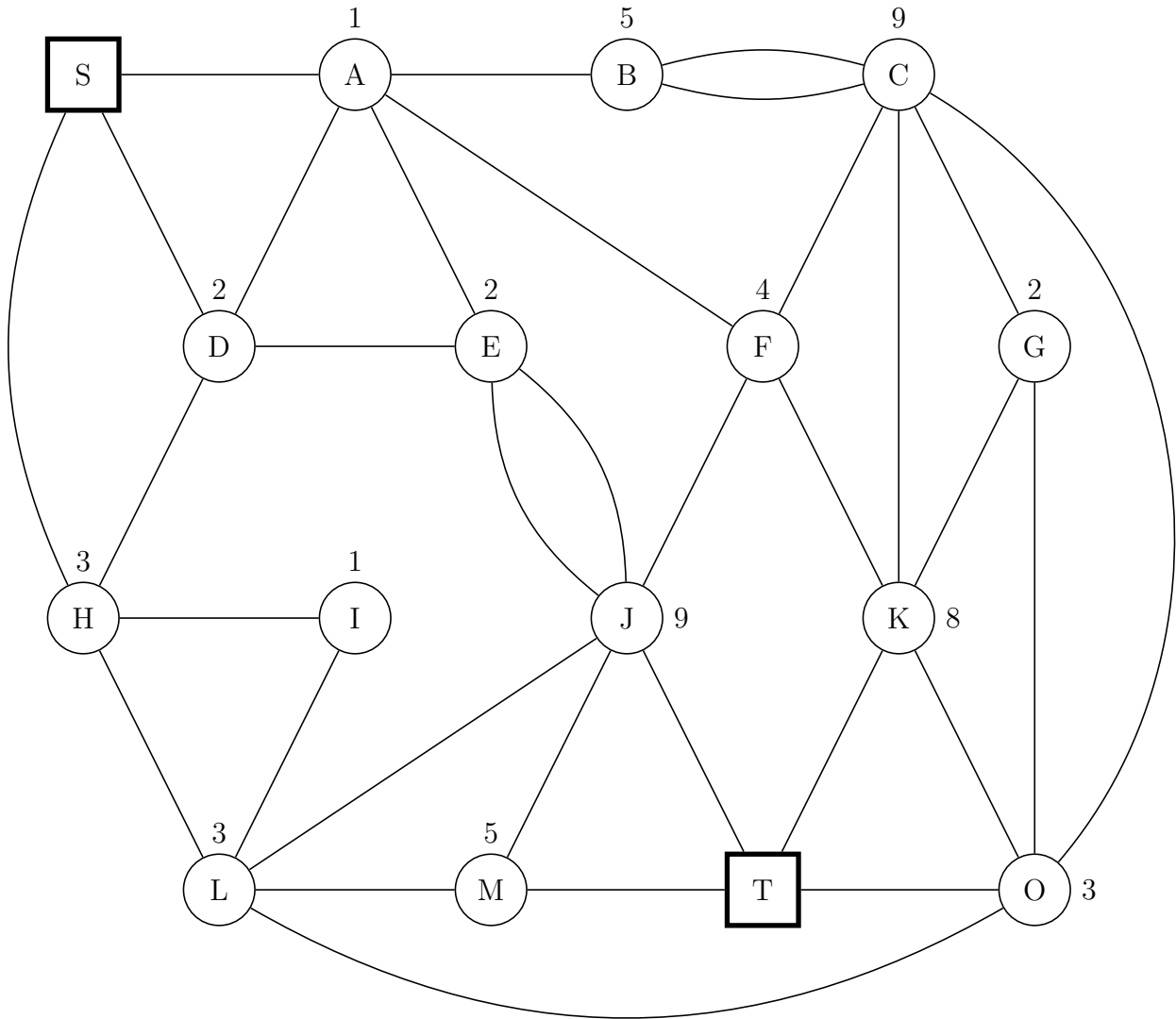
$\Theta(n^4)$

3. (2 pts) What is the runtime of the code below in terms of n? (Use Θ notation.)

```python
from containers.AVLTree import AVLTree

avl = AVLTree()
for i in range(n):
    avl.insert(i)
```

$\Theta(n \log n)$

Problem 6. (3 pts) List all of the shortest paths from node S to node T in the following graph. If no path exists between these two nodes, say so.

Solution: SHLOT