Supplemental Instruction Fall 2021
CSCI 141 Practice Exam 2

1. A stack works like a line of cars at the gas station. [TRUE / **FALSE**]
2. You can use a Python list to implement a stack by making the end of the list the top of your stack [**TRUE** / FALSE]
3. A queue is a LIFO structure [TRUE / **FALSE**]
4. The *top* function mutates a stack. [TRUE / **FALSE**]
5. The undo button operates like a stack. [**TRUE** / FALSE]
6. The keys in a dictionary can be mutable. [TRUE / **FALSE**]
7. Follow the stack!

   stk = mkEmptyStack()

   push(stk, “A”)

   push(stk, “B”)

   push(stk, “C”)

   pop(stk)

   push(stk, “D”)

   push(stk, “E”)

   pop(stk)

   pop(stk)

   push(stk, “F”)

   FBA

8. Follow the queue!

   q = mkEmptyQueue()

   enqueue(q, 3)

   enqueue(q, 2)

   enqueue(q, 1)

   print(front(q))

   print(back(q))

   print(dequeue(q))

   dequeue(q)

   dequeue(q)

   print(emptyQueue(q))

   enqueue(q, “cue”)

   cue

9. Which of the following Python collections are immutable?

   - Set
   - List
   - **Tuple**

10. Match the following:

__4__ | < 1, 2, 3, 4, 5, 6 >	1) List
__3__ | { 1, 2, 3, 4, 5, 6 }	2) Tuple
__2__ | (1, 2, 3, 4, 5, 6)	3) Set
__4__ | 1 2 3 4 5 6	4) Doesn’t match any types
__1__ | [1, 2, 3, 4, 5, 6]	
__4__ | 1, 2, 3, 4, 5, 6	

11. More matching:

__**3**__ | Divide the list in half, recursively sort each half, then figure out how to interweave the two sorted halves such that you have one sorted list at the end

__**4**__ | Using some value already in the list, split up the list into the values that are smaller than, equal to, and greater than the value. Repeat the process on the first and third parts recursively, and then concatenate all the pieces together.

__**2**__ | Repeatedly take the values of the unsorted end of the list and find the right spot for them in the sorted end. Repeat until the entire list is sorted.

__**1**__ |Start from the left of the unsorted list and then compare that value to the value left of it. If the current value is less than the one you're comparing it to, swap the values and keep swapping until your value is greater than the one to the left.

1) Insertion Sort
2) Selection Sort
3) Merge Sort
4) Quick Sort

12. Fill in the Time Complexity Table:

	Best Case	Average Case	Worst Case
Merge Sort	O(n log n)	O(n log n)	O(n log n)
Quick Sort	O(n log n)	O(n log n)	O(n^2)
Insertion Sort	O(n)	O(n^2)	O(n^2)
Selection Sort	O(n^2)	O(n^2)	O(n^2)
Binary Search	O(1)	O(log n)	O(log n)
Linear Search	O(1)	O(n)	O(n)

13. Searching for a value in a sorted list takes ___**O(log n)**____ time.

14. Searching for a value in an unsorted list takes ____**O(n)**____ time.

15. Accessing the last element in a Python list takes ____**O(1)**____ time.

16. Accessing a value in a Python dictionary takes ____**O(1)**_____ time.

17. Create a Data structure named *Student* that has a name, GPA, ID number, and major. Create two unique students using your new data structure

```
from dataclasses import dataclass
@dataclass
class Student:
    name: str
    GPA: float
    id: int
    major: str
s1 = Student('Chad', 4.0, 73478754, 'CS')
s2 = Student('Alex', 0.7, 1, 'SI')
```

18. How can you check to see if 'instant_ramen' exists in a dictionary, and if it doesn't how would you add it? Hint: 'instant_ramen' is a key

```
cookbook = dict()
if 'instant_ramen' in cookbook.keys():
# if cookbook['instant_ramen'] is None:
    print(meal of the gods)
else:
    cookbook['instant_ramen'] = 'the nectar of the gods'
```

19. Use the following data structure to finish the function:

```
@dataclass(frozen=False)
class LinkedNode:
    value: Any
    next: Union[None, 'LinkedNode']


def insert_in_order(value, lnk):
```