CS 5001 Fall 2023 Final Exam Review Practice Questions

1. Search: The following code implements a simple linear search.

```
def linear_search(data, search_value):
    for index, value in enumerate(data):
        if value == search_value:
            return ***
    return -1
```

In the statement `return ***`, what should replace `***` to indicate where the search value was found?
a. `data`
b. `search_value`
c. `index`
d. None of the above

2. Dictionaries: Which of the following statements (if any) is *false*?
a. **True/False:** The following code creates the dictionary `roman_numerals`, which maps roman numerals to their integer equivalents (the value for `'X'` is intentionally wrong):

```
roman_numerals = {'I': 1, 'II': 2, 'III': 3, 'V': 5, 'X': 100}
```

b. **True/False:** The following code gets the value associated with the key `'V'` in Part (a):

```
roman_numerals['V']
```

c. **True/False:** You can update a key's associated value in an assignment statement. The following code replaces the incorrect value associated with the key `'X'` in Part (a):

```
roman_numerals['X'] = 10
```

3. Algorithms: Which of the following statements is *false*?
a. The linear search algorithm runs in $O(n)$ time.

b. The worst case in the linear search algorithm is that every element must be checked to determine whether the search item exists in the array. If the size of the array is doubled, the number of comparisons that the algorithm must perform is quadrupled.

c. Linear search can provide outstanding performance if the element matching the value searched for happens to be at or near the front of the array.

d. Linear search is easy to program, but it can be slow compared to other search algorithms. If a program needs to perform many searches on large arrays, it's better to implement a more efficient algorithm, such as the binary search.

We strongly recommend you practice the following questions first without plugging them into the terminal.

4. Lists & Operators:

After the following Python snippet executes, what are the values of the variables *i* and *even*?

```
nums = [3, 6, 8, 1]
i = 0
even = 0
while i < len(nums):
      if nums[i] % 2 == 0:
            even += 1
      i += 1
```

i = ______________ *even* = _________________

5. Recursion

How many times will the following Python program print * to the terminal?

```
def recursive_print(n):
    if n == 1:
       print('*')
    else:
       recursive_print(n-1)

def main():
    recursive_print(10)

main()
```

6. **Recursion**

Write a recursive function that sums every other item in a list (even indexes starting at 0).

7. Classes (write it up in IDLE for practice. No solution given)

Write a class to represent a color. Your class should have three attributes: red, green, and blue. Each of these attributes will be an integer in the range 0-255.

Define the class and give it three methods:

- `__init__`. Takes no parameters. Default color is gray (red, green, and blue attributes should all be initialized to 100).

- `__str__`. Takes no parameters. Returns a string that prints the RGB values in the format **`(R, G, B)`**. Ex: this method would return the string **`'(0, 0, 255)'`** for an RGB object which has red = 0, green = 0, and blue = 255.
- `greenify`. Takes no parameters. Increases the value of the green attribute by 50%, but doesn't allow it to go over 255, which is the max. The value of the green attribute must still be an integer.

For example, here's what a driver might look like using your defined class:

```
color = RGB()
print(color)

color.greenify()
print(color)

color.greenify()
print(color)

color.greenify()
print(color)

color.greenify()
print(color)
```

And it would print the following to the terminal:

```
(100, 100, 100)
(100, 150, 100)
(100, 225, 100)
(100, 255, 100)
(100, 255, 100)
```

Solutions

Question 1: Answer: c.

Question 2: Answer: All of the above statements are *true*.

Question 3:
Answer: b. Actually, if the size of the array is doubled, the number of comparisons that the algorithm must perform also is doubled.

Question 4:

$i = 4$

$odd = 2$

Question 5:

Just once

Question 6:

```
def recursive_sum_evens(num_list):
   if len(num_list) == 0:
      return 0
   elif len(num_list) == 1:
      return num_list[0]
   else:
      print(num_list[0])
      return num_list[0]  + recursive_sum_evens(num_list[2:])
```

** note: because Python “gracefully” handles slicing that in other languages would produce an IndexError, this code works – you can even take out the “if len(num_list) ==1” and it will run properly. Adding an additional length check in the else will help your code not rely on Python to handle slicing gracefully.