Assignment 3

Short Answers

Answer the following questions with complete sentences in your own words. You are encouraged to conduct your own research online or through other methods before answering the questions. If you research online, please consult multiple sources before you write down your answers. You are expected to be able to explain your answers in detail. If you are unable to understand or answer some of the questions asked or the contents you have found, please provide the sources such as the website link or book, and mark your question in red.

1. What is OOP?
   Object oriented programming is a programming model that revolves around objects instead of functions and logic. It is better suited for large, complex, and actively updated programs. Objects can interact with other objects or can inherit traits from other objects.
2. What is a Class?
   A class is a user-defined data type that acts as a blueprint of the object. Any number of objects can be based on the same class, and classes can inherit traits from other classes.
3. What is an Object?
   An object is a basic entity in Object Oriented Programming. It is an instance of a class with specifically defined data and a collection of methods that work on that data.
4. What is the difference between pass by value and pass by reference? What is pass by object reference?
   In pass by value the method parameter values are copied and another variable is created. If one variable is modified, the other is not affected. In pass by reference, variables store a reference to a certain memory location that contains the value. If the value at that memory location is modified, both variables are modified (in the case of mutable data types). If changes are made to an immutable data type, the new data will be stored at a different memory location.
5. What is a Module?
   A module is a python script that can be imported from a different module
6. What is Encapsulation?
   Encapsulation is how to bind data members and functions into a single unit like a class. Access to methods and variables within the class can be restricted by making them private.
7. What is Inheritance?
   Inheritance is a way to create a class with features of an existing class. The new class is derived from the parent class.
8. What is Polymorphism?

Polymorphism in OOP is the ability to access objects of different types through the same interface. It's the ability of different objects to respond to identical messages/commands.

9. Why do we use lambda expressions?
   Lambda expressions are used to quickly make simple functions without having to define a function using def
10. What is recursion?
    Recursion is when a function calls itself. Recursion must have a base case in order to prevent it from running indefinitely.

Coding Practice

Write algorithms in Python to Complete following questions. Please write your own answers. You are highly encouraged to present more than one way to answer the questions. Please follow the syntax, and you need to comment on each piece of code you write to clarify your purpose. Also you need to clearly explain your logic of the code for each question. Clearly state your assumptions if you have any. You may discuss with others about the questions, but please write your own code. If you are unable to answer the questions, you can write the question in pseudocode but you need to explain your algorithm clearly in detail.

For all questions, list out your time complexity and space complexity.

1. You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have n versions [1, 2, ..., n] and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API bool isBadVersion(version) which returns whether version is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

Example 1:

Input: n = 5, bad = 4

Output: 4

Explanation:

Call isBadVersion(3) -> false

Call isBadVersion(5) -> true

Call isBadVersion(4) -> true

You can use binary search for this problem to use the minimum number of calls to the API. Check the middle value, if it's bad then check the left half, if it's not then the right half, until you find the minimum index for bad.

Time complexity for binary search is O(logn). Space complexity is O(1) because you're only storing constants (ignoring the time and space complexity of isBadVersion).

```
# Use binary search algorithm
def firstBadVersion(n, bad):
    left = 0
    right = n-1
    while (left <= right):
        middle = (left+right)/2
        if isBadVersion(middle):
            right = middle
        else:
            left = middle + 1
    return left
```

2. Given two strings and goal, return true *if you can swap two letters in* s *so the result is equal to* goal*, otherwise, return* false*.*

Swapping letters is defined as taking two indices i and j (0-indexed) such that i != j and swapping the characters at s[i] and s[j].

- For example, swapping at indices 0 and 2 in "abcd" results in "cbad".

Example 1:

Input: s = "ab", goal = "ba"

Output: true

Explanation: You can swap s[0] = 'a' and s[1] = 'b' to get "ba", which is equal to goal.

Example 2:

Input: s = "ab", goal = "ab"

Output: false

Explanation: The only letters you can swap are s[0] = 'a' and s[1] = 'b', which results in "ba" != goal.

Example 3:

Input: s = "aa", goal = "aa"

Output: true

Explanation: You can swap s[0] = 'a' and s[1] = 'a' to get "aa", which is equal to goal.

For this problem, if the lengths are not equal, then swaps can't happen. If the lengths are equal, and the strings are identical, then if there is at least one repeated character, the swap can happen. After checking for those cases, iterate through the strings s and goal to check for if there are just 2 differences, because any more than that can't be fixed with a swap. Then check if the differences are swapable.

Time complexity is O(n) because you iterate through each string once, and space complexity is O(1) because count and index are constant valued.

```
def goal(s, goal):
    count = 0
    #store indexes of the differences
    index = []
    if len(s) != len(goal):
        return False
    #if they're identical strings
    if s == goal:
        #if there is at least one duplicate pair of letters
        return len(s) - len(set(s)) >= 1
    for i in range(len(s)):
         if s[i] != goal[i]:
             count += 1
             index.append(i)
    if count != 2:
        return False
    return s[index[0]] == goal[index[1]] and s[index[1]] == goal[index[0]]
```

3. Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

Example 1:

Input: nums = [1,2,3,1]
Output: true

Example 2:

Input: nums = [1,2,3,4]
Output: false

Example 3:

Input: nums = [1,1,1,3,3,4,3,2,4,2]
Output: true

You can just compare the number of elements vs the number of unique elements (set). If they're the same, return false. Time complexity is O(1) because len() is O(1). Space complexity is also O(n) because you need to create a set.

```
def duplicate(nums):
    return len(set(nums)) != len(nums)
```

4. You are given two strings s and t.

String t is generated by random shuffling string s and then add one more letter at a random position.

Return the letter that was added to t.

Example 1:
Input: s = "abcd", t = "adcde"
Output: "e"

Example 2:
Input: s = "", t = "y"
Output: "y"

For this problem you can turn the string t into a list of characters, and remove all the characters in s from the list to find the single added character. The time complexity is O(n^2) because you iterate through s and t, but remove is also O(n). Space complexity is O(n) because you're turning t into a list of characters.

```python
def letterAdded(s, t):
    letters = list(t)
    for ch in s:
        letters.remove(ch)
    return letters[0]
```

The better way is to use XOR to find the single unique letter's ASCII value and converting it back to a character. The time complexity is O(n) and space complexity is O(1).

```python
def letterAdded1(s,t):
    value = 0
    for ch in s + t:
        value ^= (ord(ch))
    return chr(value)
```

5. Given a string s, reverse only all the vowels in the string and return it.

The vowels are 'a', 'e', 'i', 'o' and 'u', and they can appear in both cases.

Example 1:

Input: s = "hello"
Output: "holle"

Example 2:

Input: s = "assignment"
Output: "essignmant"

For this problem you can convert the string into a list of characters, and check whether each character is in the set of vowels. Then you can store the vowels and their indices, then reverse the vowel list while maintaining the indices list and replace the following index in the original list. The time complexity is O(n) because set lookup is O(1) and you loop through s once and indices once. Space complexity is O(n) because you need to store the vowels and the indices.

```python
def reverseVowels(s):
    #use set for constant time lookup
    letters = list(s)
    vowels = {'a','e','i','o','u','A','E','I','O','U'}
    vowel_list = []
    indices = []
    for i in range(len(s)):
        if s[i] in vowels:
            vowel_list.append(s[i])
            indices.append(i)
```

```
    vowel_list.reverse()
    for i in range(len(indices)):
        letters[indices[i]] = vowel_list[i]
    return ''.join(letters)
```

6. Given a string s, find the length of the longest substring without repeating characters.

Example 1:
Input: s = “abcabccbb”
Output: 3
Explanation: The answer is “abc”, with the length of 3

Example 2:
Input: s = “bbbbb”
Output: 1
Explanation: The answer is “b”, with the length of 1.

Example 3:
Input: s = “pwwkew”
Output: 3
Explanation: The answer is “wke”, with the length of 3.

For this problem you can use a dictionary to map seen characters to their last seen index. Maintain the left and right pointers for the substring, and shift the left pointer if you see a repeated character to begin a new substring. Check if each new substring is longer than the current max_length and update.

The time complexity is O(n) because you iterate through the string once with enumerate. Space complexity is O(1) because the dictionary stores repeated characters, and there are a constant number of available characters.

```
def longestSubstring(s):
    seen = {}
    left = 0
    max_length = 0
    for right, ch in enumerate(s):
        if ch in seen and left <= seen[ch]:
            left = seen[ch] + 1
        else:
            max_length = max(max_length, right - left + 1)
        seen[ch] = right
    return max_length

print(longestSubstring('adcadccbb'))
```

7. You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

Merge nums1 and nums2 into a single array sorted in non-decreasing order.

Example 1:

Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3
Output: [1,2,2,3,5,6]

Example 2:

Input: nums1 = [1], m = 1, nums2 = [], n = 0
Output: [1]

Example 3:

Input: nums1 = [0], m = 0, nums2 = [1], n = 1
Output: [1]

For this problem you can turn the list of integers into a list of characters and join them to form two strings. Then use m and n to take the nonzero parts of the two strings, add and sort them, before turning them back into a sorted list of characters. Then output that list of characters casted as integers.
The time complexity is O(n) because you loop through each list once. The space complexity is O(n) because the resulting list has size m+n.

```
def mergeNums(nums1, nums2, m, n):
    s_nums1 = ''.join([str(x) for x in nums1])
    s_nums2 = ''.join([str(x) for x in nums2])
    result = sorted(s_nums1[:m] + s_nums2[:n])
    return [int(x) for x in result]
```

8. Guessing Game

Input: int min, int max, int allowed_guesses

1. Randomly generate a target number between min and max inclusive
2. When the guess is smaller/greater than the target, output "Too Small/Big!"
3. When current guess is closer/farther to target than previous guess, output "Warmer/Colder"
4. When a correct guess is made within the allowed guesses, output "Congrats! Wanna try again?"
    If user input "yes", reset the game and ask for the three inputs again
5. When guesses exceed allowed times, output "You've lost! Correct answer is: XXXXX. Wanna try again?"
    If user input "yes", reset the game and ask for the three inputs again,

If user input "no", terminate program,

Implement a Class Guess to complete the above guess game

```
import random
class Guess():
    def __init__(self):
        self.allowed_guesses = 0
        self.guess_count = 0
        self.min = 0
        self.max = 0
        self.target = 0
        self.player_guess = 0
        self.previous_guess = 0
        self.play = 'no'
        self.game()

    def game(self):
        #keep running until it breaks
        while(True):
            # Three specified inputs
            self.min = int(input("Enter min: "))
            self.max = int(input("Enter max: "))
            self.allowed_guesses = int(input("Enter allowed guesses: "))
            #determine target from input using random.randint
            self.target = random.randint(self.min, self.max)
            #print(self.target) checking if working properly

            self.player_guess = int(input("Guess the number between {} and {}:
".format(self.min,self.max)))
            # first guess
            if self.player_guess > self.target:
                self.previous_guess = self.player_guess
                print("Too Big!")
                self.guess_count += 1
            if self.player_guess < self.target:
                self.previous_guess = self.player_guess
                print("Too Small!")
                self.guess_count += 1
            elif self.player_guess == self.target:
                self.play=input("Congrats! Wanna try again? ")
                break

            # additional guesses
            while self.player_guess != self.target:
```

```
                if self.guess_count == self.allowed_guesses:
                    self.play = input("You've lost! Correct Answer is " +
str(self.target) + ". Wanna try again? ")
                    break
                self.player_guess = int(input("Enter another number: "))
                self.previous_guess = self.player_guess
                if self.player_guess == self.target:
                    self.play=input("Congrats! Wanna try again? ")
                    break
                # warmer/colder calculated using absolute value of the difference
                elif abs(self.target - self.previous_guess) >= abs(self.target -
self.player_guess):
                    print("Warmer")
                    #print("guesses", self.guess_count) checking counter
                    self.guess_count += 1
                elif abs(self.target - self.previous_guess) < abs(self.target -
self.player_guess):
                    print("Colder")
                    #print("guesses", self.guess_count)
                    self.guess_count += 1
            #break out of while(true)
            break
        # restart game
        if self.play == 'yes':
             Guess()
```