Assignment 2

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 a tuple?
   A tuple is an immutable, ordered data type that allows duplicate values, and can contain values of different types.
2. What's the difference between tuple and list?
   A list is mutable (so the elements within the list and the size of the list can be changed). Lists support indexing and slicing. A list must also only contain one data type.
3. How does the dictionary different from the list?
   Dictionary contains unique keys that map to values, as opposed to a list which only contains a set of values. Objects in dictionaries are retrieved by key name and cannot be sorted, while objects in lists can be retrieved by location, ordered sequence through indexing and slicing.
4. What are some set operations?
   Add() : adds a value to a set
   Remove()/discard(): removes a value from a set
   Update(): updates a set from another set
   Union/Intersection/Difference/Symmetric Difference: Find various relationships between sets
5. How to find a unique value from a list?
   There are various ways, but a simple way is to cast the list as a set.
6. What is casting?
   Casting turns an object of one data type into a different data type if they have similar structures or if the conversion makes sense.
7. Do more research on Time Complexity and Space complexity, please check the time complexity of all the methods introduced during the training, and please check the time complexity of all the algorithms you write.
   Time complexity:
   O(1) : dictionary lookup, constant operations, constant loops
   O(logn): binary search on sorted array
   O(n): iterating through a list of length n
   O(nlogn): Mergesort
   O(n^2): Iterating through an array of length n with nested for loops

8. What is the difference between stack and queue?
   A stack is LIFO (last in first out) so the most recently inserted value is the one that gets removed first.
   A queue is FIFO, (first in first out) so the first inserted value is removed first.
9. How to implement stack or queue in Python?
   You can use a list to imitate the structure of stacks and queues. Pop index -1 of the list to mimic a stack, and pop index 0 to mimic a queue.

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.Given a String s containing just the characters '(', ')', '{', '}', '[', and ']', determine if the input string is valid.

An input string if valid id:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

Example 1:

Input: s = "()"
Output: true

Example 2:

Input: s = "()[]{}"
Output: true

Example 3:

Input: s = "(]"
Output: false

This problem was solved using a stack and a dictionary. The stack contains open parentheses in the string, and the dictionary is used to look up whether the close parenthesis matches the open parenthesis. If it does match, then it is successfully popped from the stack and we can examine the next parenthesis.
Time complexity O(n) because the only nonconstant you have to loop through is the string s.
Space complexity is linear because you just need the string s and the stack.

```python
def validParenthesis(s):
    stack = []

    p = {")": "(", "]" : "[", "}" : "{"}
    for letter in s:
        if letter in ["(", "[", "{":
            stack.append(letter)
        elif letter in [")", "]", "}"]:
            # Close parenthesis without open
            if not stack:
                return False
            # Parenthesis doesn't match
            elif stack.pop() != p[letter]:
                return False
            else:
                continue
    if len(stack) == 0:
        return True
    else:
        return False
```

2. Given a string containing digits from 2 - 9 inclusive, return all possible letter combinations that the number could represent. Return the answer in any order.

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

Example1:

Input: digits = “23”

Output: [“ad”, “ae”, “af”, “bd”, “be”, “bf”, “cd”, “ce”, “cf”]

Example 2:

Input: digits = “”

Output: []

Example 3:

Input: digits = “2”

Output: [“a”, “b”, “c”]

This problem can be solved using a dictionary mapping each of the number keys to the digits that they represent. You can then simply loop through the input string of digits and append to the result array.

The time complexity is O(4^n) because every loop can create up to 4 times the number of combinations compared to the previous iteration of the loop. This problem can’t be done in a faster way than O(3^n) because the minimum number of elements in the output would be 3^n.

Space complexity is also 4^n because the largest possible output list would have that size as well

```
def letterCombo(s):
    mapping = {
        '2' : 'abc',
        '3' : 'def',
        '4' : 'ghi',
        '5' : 'jkl',
        '6' : 'mno',
        '7' : 'pqrs',
        '8' : 'tuv',
        '9' : 'wxyz'
    }
    #empty list if no digits
    result = [''] if s else []
    for digit in s:
        combinations = []
        for letter in mapping[digit]:
            for combination in result:
                combinations.append(combination+letter)
        result = combinations
    return result
```

3.You are given an array target and an integer n.

Build the target array using the follow operations:

- “Push”: Reads a new element from the beginning list, and pushes it in the array.
- “Pop”: Deletes the last element of the array.
- If the target array is already built, stop reading more elements.

Return a list of the operations needed to build target. The test cases are generated so that the answer is unique.

Example 1:

Input: target = [1, 3], n = 3
Output: [“Push”, “Push”, “Pop”, “Push”]
Explanation:

Read number 1 and automatically push in the array -> [1]
Read number 2 and automatically in the array then Pop it - > [1]
Read number 3 and automatically push in the array -> [1,3]

Example 2:

Input: target = [1, 2, 3], n = 3
Output: [“Push”, “Push”, “Push”]

Example 3:

Input: target = [1, 2], n = 4
Output: [“Push”, “Push”]
Explanation:

You only need to read the first 2 numbers and stop

For this problem you can just check for values from 1 to the largest value in the target. For every single value from 1 to the largest value in target, append “Push” to the output, and if there are empty values in between, append pop for each of those values.

The time complexity would be O(n) because you only loop through the length of the target array once. Space complexity is linear because you need to store target and output

```
def buildTarget(target, n):
    output = []
    index = 0
    for i in range(1,target[-1] + 1):
        output.append("Push")
        if i != target[index]:
            output.append("Pop")
        else:
            index += 1
```

```
    return output
```

4.Given an array of integer nums and an integer target, return indices of the two numbers such that they add up to target. You may assume that each input would have exactly one solution, and you may not use the same element twice. You can return the answer in any order.

Example 1:

Input: nums = [2, 7, 11, 15], target = 9
Output: [0, 1]
Explanation: Because nums[0] + nums[1] == 0, we return [0, 1]

Example 2:

Input: nums = [3, 2, 4], target = 6
Output: [1, 2]

Example 3:

Input: nums = [3, 3], target = 6
Output: [0, 1]

For this problem the brute force way to do it would be to loop through the nums array two times and find the sums of every value in nums with every other value, but that would be O(n^2).

For a O(n) solution, you can use a dictionary to map values i1 and i2 where i2 = target – i1, since there are only two unique values that add up to the target. If the dictionary value is empty, then you can add the new value to the dictionary. Space complexity is linear because you need to store nums and the map differences

```
def indicesSum(nums, target):
# Dictionary to map int 1 to int 2, where i2 = target - i1
    differences = {}
    for i in range(len(nums)):
        if nums[i] in differences:
            return [differences[nums[i]], i]
        else:
            differences[target-nums[i]] = i
```

5.Given two strings s and t, determine if they are isomorphic.

Two strings s and t are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character, but a character may map to itself.

Example 1:
Input: s = “egg”, t = “add”
Output: true

Example 2:
Input: s = “foo”, t = “bar”
Output: true

Example 3:
Input: s = “paper”, t = “title”
Output: true

The simplest way to do this would be to use the find functions in python to check if the indices are the same for the characters in each string.
The time complexity would be O(n^2) because the find function is O(n) and the for loop is also O(n). Space complexity is linear because you still only store s and t

```
def isomorphic(s,t):
    return [s.find(i) for i in s] == [t.find(j) for j in t]
```

6. Given a string s, find the first non-repeating character in it and return its index. If it does not exist, return -1.

Example 1:
Input: s = “python”
Output: 0

Example 2:
Input: s = “pythonprogram”
Output: y

Example 1:
Input: s = “aabb”
Output: -1

This solution uses a set to get a set of all the characters in the string. For each of those characters, if the count is 1, then store the index of the character. Return the value of the character at the minimum index stored in the indices list.

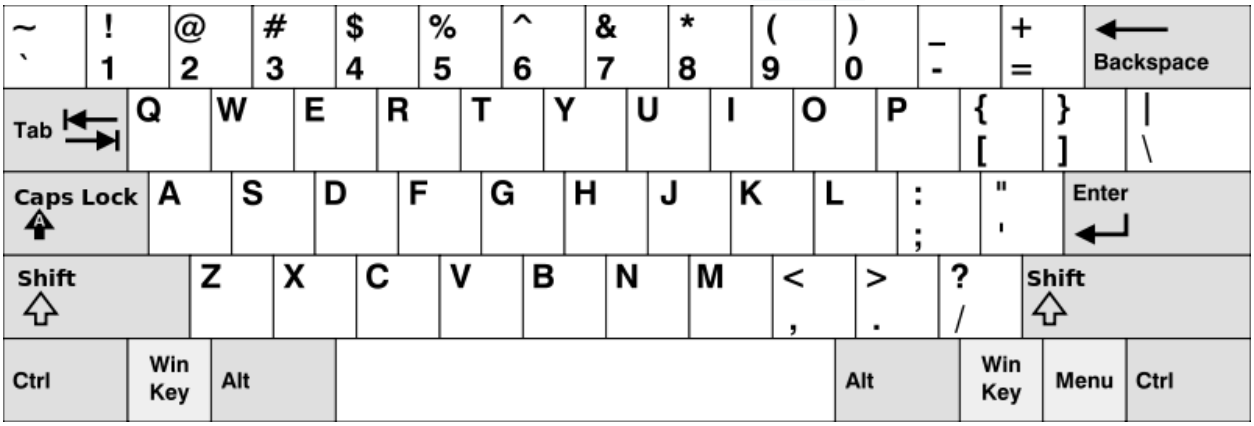
The time complexity of this solution is O(n^2) because count is O(n) and we loop through the set(s) once. Space complexity is linear because we have s, set(s) and indices[ ].

```
def nonRepeat(s):
    indices = []
    for i in set(s):
        if s.count(i) == 1:
            indices.append(s.index(i))
    if len(indices) > 0:
        return s[min(indices)]
    else:
        return -1
```

7. Given an array of string words, return the words that can be typed using letters of the alphabet on only one row of the American keyboard like the image below.

In American keyboard:

- The first row consists if the characters "`qwertyuiop`"
- The second row consists if the characters "`asdfghjkl`"
- The second row consists if the characters "`zxcvbnm`"



Example 1:

Input: words = ["Hello", "Alaska", "Dad", "Peace"]
Output: ["Alaska", "Dad"]

Example 2:

Input: words = ["omk"]
Output: []

Example 3:

Input: words = ["adsdf", "sfd"]
Output: ["adsdf", "sfd"]

For this problem, you can map each of the letters of the alphabet to the row on the keyboard.

We can check if all the letters in a word are in the same row by using a set containing the values that the characters are mapped to. If the set only contains 1, 2, or 3, then all of the letters in that word belong to row 1, 2 or 3, respectively.

The time complexity for this solution is O(n^2) because it goes through every letter of every word. Realistically, if the lengths of words were considered a constant (if words have a maximum length), then it would be O(n). Space complexity would be linear from words and output lists.

```
def oneRow(words):
# map the letters to the row
    mapping = { 'q':1,'w':1,'e':1,'r':1,'t':1,'y':1,'u':1,'i':1,'o':1,'p':1,
                'a':2,'s':2,'d':2,'f':2,'g':2,'h':2,'j':2,'k':2,'l':2,
                'z':3,'x':3,'c':3,'v':3,'b':3,'n':3,'m':3 }
    output = []
    for word in words:
        same_row = set()
        for letter in word:
            same_row.add(mapping[letter.lower()])
        if len(same_row) == 1:
            output.append(word)
    return output
```

8. International Morse Code defines a standard encoding where each letter is mapped to a series of dots and dashes, as follows:

- 'a' maps to ".-",
- 'b' maps to "-...",
- 'c' maps to "-.-.", and so on.

For convenience, the full table for the 26 letters of the English alphabet is given below:

[".-","-...","-.-.","-..",".","..-.","--.","....","..",".---","-.-",".-..","--","-.","---",".--.","--.-",".-.","...","-","..-","...-",".--","-..-","-.--","--.."]

Given an list of string words where each word can be written as a concatenation of the Morse code of each letter.

- For example, "cab" can be written as "-.-..--...", which is the concatenation of "-.-.", ".-", and "-...". We will call such a concatenation the transformation of a word.

Return *the number of different transformations among all words we have*.

Example 1:

Input: words = ["gin","zen","gig","msg"]

Output: 2

Explanation: The transformation of each word is:

"gin" -> "--...-."

"zen" -> "--...-."

"gig" -> "--...--."

"msg" -> "--...--."

There are 2 different transformations: "--...-." and "--...--.".

For this problem, you could also do the mapping method for each morse code character, similar to question 7, but there's a more concise way of doing it. First you can create a list of the words in morse code form by using the join function and finding the right index of the morse code list using ascii subtraction for each letter in each word. Then you can cast it as a set to find how many unique transformations there are and return the length of that. The time complexity would be O(n^2) because it has to loop through every letter in every word to join them into the morse code representation. Space complexity is linear for the morse_w and words lists.

```
def morse(words):
    morse_code = [".-","-...","-.-.","-..",".","..-.","--.",
                  "....","..",".---","-.-",".-..","--","-.",
                  "---",".--.","--.-",".-.","...","-","..-",
                  "...-",".--","-..-","-.--","--.."]
    morse_words = []
    for word in words:
        # get the right index by subtracting ascii values from ord
        morse_w = ''.join(morse_code[ord(letter)-ord('a')] for letter in word)
        morse_words.append(morse_w)
    #set for unique transformations
    return len(set(morse_words))
```

9. Given two strings s and t, return true of t is an anagram of s, and false otherwise.

An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:

Input: s = "anagram", t = "nagaram"

Output: true

Example 2:

Input: s = “rat”, t = “car”

Output: false

To check if two strings are anagram, just sort the characters and see if they’re equal
The time complexity is O(nlogn) because the python sorted function sorts in O(nlogn)
Space complexity is linear because you need 2 lists.

```
def anagram(s,t):
    return sorted(s) == sorted(t)
```

You could also use the all() function, which returns true if all items are true.
The time complexity is O(n) because count is O(n) and the alphabet is constant sized, so the for loop and the all function will also scan through a list of maximum size 26.
Space complexity is linear because you need 2 lists

```
def anagram1(s,t):
    return all([s.count(l) == t.count(l) for l in 'abcdefghijklmnopqrstuvwxyz'])
```