Date: 02.03.24
ExNo: 2

Reg No: 3122225001127
Name: Shaun Allan H

ASSIGNMENT – 2

AIM:

To solve the given problems using Python and analyze the time complexities of the problems.

Qn1:

1. (a) Develop a Python program to find unique (non-repeating) elements in a list. That is, find those elements that do not have duplicates in the list. For example, in the list $[3, 6, 9, 2, 3, 9, 1, 15, 21, 3, 1]$, the unique elements are $[6, 2, 15, 21]$. The order of elements in the output list should be the same as that in the original list.

   (b) What is the time complexity of your algorithm? You may ignore the improvements introduced by language specific implementations (say, using *set* in Python).

   [CO1,K3]

- Develop a Python code that takes as input a value n, and generates a list of n unique random values. [CO1,K3]

Psuedo Code:

```
1. find_unique_element ():
   Inputs : int[] array
   Outputs : unique_element array
   1. for each element i in array
        if (count(i) in array == 1):
            append i to new_array
        end if
    end for.
   2. return new_array
```

Date: 02.03.24
ExNo: 2

Reg No: 3122225001127
Name: Shaun Allan H

Source Code:

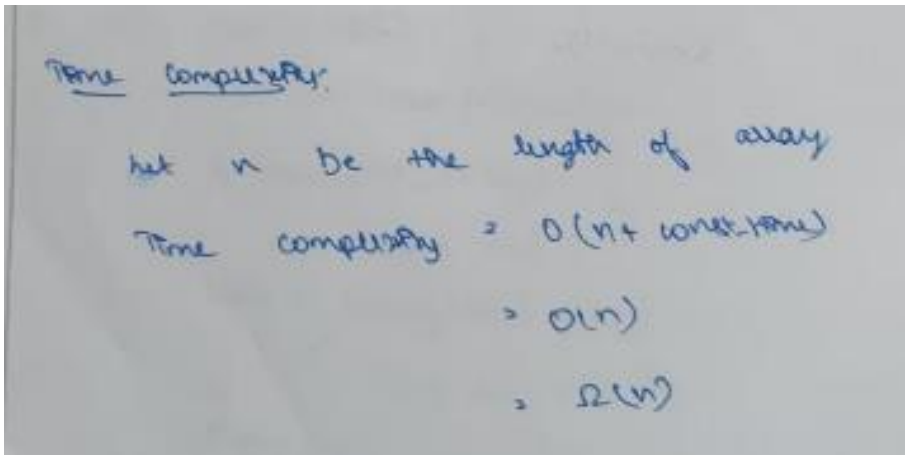
```
def getUnique(l):
    unique = []
    for i in l:
        if l.count(i) == 1:
            unique.append(i)
    return unique

l = [3,6,9,2,3,9,1,15,21,3,1]
unique = getUnique(l)
print("Unique elements: ", unique)
```

Output:

```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment2> python 1.py
Unique elements:  [6, 2, 15, 21]
```

Time Complexity:



Date: 02.03.24
ExNo: 2

Reg No: 3122225001127
Name: Shaun Allan H

Qn2:

2. (a) Develop a Python program that when given an integer n as input, prints the sum of the following series up to n terms.

$$1 + (1 + 2) + (1 + 2 + 3) + \ldots + (1 + 2 + 3 + \ldots + n)$$

(b) What is the time complexity of your code?

Psuedo Code:

2 sum_of_series()
Input : int n
Output : sum

1. s ← 0
2. for i → 1 to n:
   s ← s + i*(i+1)/2
   end for
3. return s

Source Code:

```
def sumSeries(n):
    sum = 0
    for i in range(n):
        sum += (i*(i+1))/2
    return sum

n = int(input("Enter n: "))
print("Sum of series: ", sumSeries(n))
```

Output:

```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment2> python 2.py
Enter n: 10
Sum of series:  165.0
```

Time Complexity:

Let n be the input number

Time Complexity = O(n * const_time)

= O(n)

Date: 02.03.24
ExNo: 2

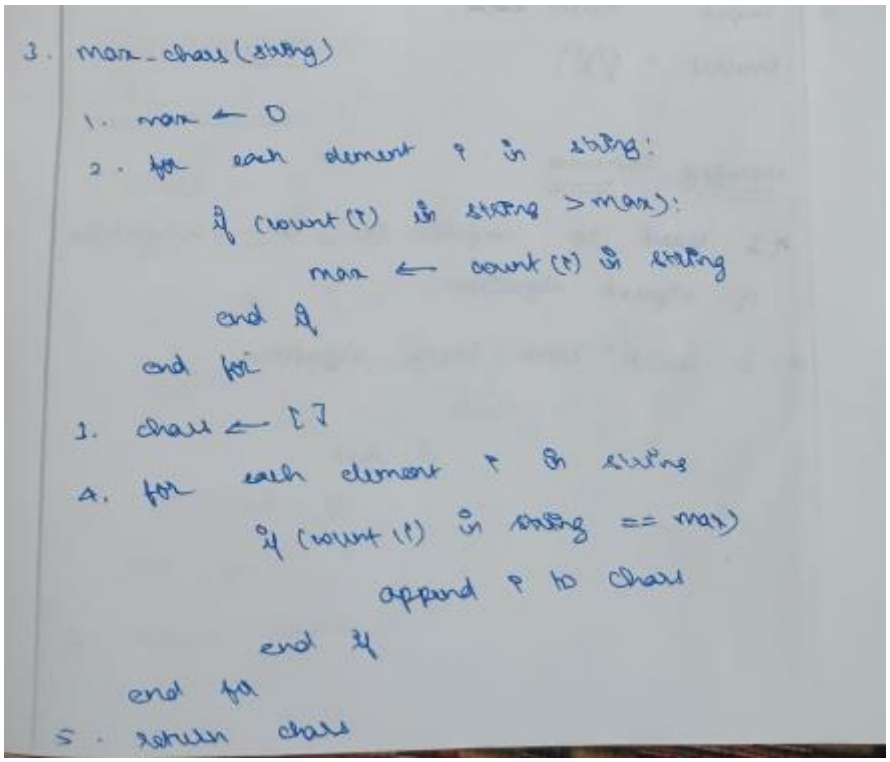
Reg No: 3122225001127
Name: Shaun Allan H

Qn3:

3. (a) Write a program to print all the most frequently occurring characters in a given string, as a list. For example, if the input string is "example",the output should be [e]. If the input string is "exist", then the output should be [e,x,i,s,t].

   (b) What is the complexity of your code?

Psuedo Code:



```
3. max_chars(string)
   1. max ← 0
   2. for each element ? in string:
        if (count(?) in string > max):
            max ← count(?) in string
        end if
   end for
   1. chars ← [ ]
   4. for each element ? in string
        if (count(?) in string == max)
            append ? to chars
        end if
   end for
   5. return chars
```

Source Code:

```
def getMostFreq(str):
    max = 1
    d = {}
    for i in str:
        if i in d:
            count = d[i] + 1
            if count > max:
                max = count
            d[i] = count
        else:
            d[i] = 1
    max_ = []
    for k,v in d.items():
        if v == max:
            max_.append(k)
```

Reg No: 3122225001127
Name: Shaun Allan H

```
    return max_

str = input("Enter string: ")
print("Mosst Frequent Characters: ", getMostFreq(str))
```

Output

```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment2> python 3.py
Enter string: example
Mosst Frequent Characters:  ['e']
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment2> python 3.py
Enter string: exist
Mosst Frequent Characters:  ['e', 'x', 'i', 's', 't']
```

Time Complexity:

Let n be the length of the input string

Time Comp = $O(n \times n \times \text{const})$

$= O(n)$

$\therefore \Omega(n) \rightarrow \Theta(n)$

Learning Outcomes:

- I learnt to analyse the time complexities of various algorithms
- I learnt how to implement various sorting and searching algorithms in Python