Name- Karan Kumbhar
Id No. - 12140860

DATE

* Question 1.

```
Class CGPA
    Class Node :
        __slots__ = ("courseCode", "gradePoint", "semValue")
        def __init__(self, courseCode, semValue, gradePoint):
            self.courseCode = courseCode
            self.gradePoint = gradePoint
            self.semValue = semValue

    def __init__(self, studentData, curentSemisler)
        self.studentData = StudentData
        self.gradeToCg = {
                "A+": 10,
                "A": 10,
                "A-": 9,
                "B": 8,
                "B-": 7,
                "C": 86,
                "C-": 5,
                "D": 4,
                "F": 0
            }
        Self.curentSemisler = curentSemisler

    def SemToValue(self, data)
        semDict = {
            "m": 1,
            "w": 2,
            "s": 3
            }
        SemValue = data.split("_")
        SemValue = (int(semValue[0]) + semValue[1] + int(
                                semDict[semValue[2]]))
        return SemValue
```

```
def makeCourseData(self):
    requiredSemisler = self.semToValue(self._currentSemisler)
    self._studentData.sort(key = lambda x: x[1])
    self._courseData = []
    prev = self._studentData[0][1]
    for i in range(len(self._studentData)):
        n = self.Node(self._studentData[i][1],
            self.semValue(self._studendata[i][6]),
                    self.gradeToCg[self._studentData[i]
                    )
        If n._semValue
        If prev != self._studentData[i][1] and i != 0
                    and n._semValue <= requiredSemister:
            If n._semValue >= self._courseData[-1]._semValu
                self._courseData.pop()
                self._courseData.append(n)
            elif (n.semValue <= requiredSemister):
                prev = n._courseCode
                self._courseData.append(n)
    return self._courseData

def calculateCGPA(self):
    sum = 0
    totalCourses = 0
    for i in self._courseData
        sum += i._gradePoint
        totalCourses += 1
    return sum / totalCourses

Do If __name__ == "__main__":
    Data = [['2023-24-m', 'CS102', F] ['2023-24-W', 'CS102',
            'B']]
    semisler = "2023-24-W"
    c = CGPA(Data, semisler)
    ValidCourseData = c.makeCourseData()
    Print(c.calculateCGPA())
```

Here I used tree and referencial array and this code taking O(N) time complexity to calculate the CGPA of at end of any semister that you have enter

- restriction ⇒ semister format can't be changed required semister should be given
- I use this data structure as it takes minimum time.

Question 2

```
Class compleleA():
    # slot #
    class Node:
        __slot__ = "_data", "_left", "_right", "_equal",
                   "_endWord";
        def __init__(self, list):
            self.n = self.Node
        def __init__(self, data = None, endWord = False):
            self._data = data
            self._left = None
            self._right = None
            self._equal = None
            self._endWord = endWord

    def __init__(self, list)
        self.n = self.Node()
        self.list = list
        for w in list:
            self.add(w, self.n)

    def add(self, w, node):
        c = w[0]
        if not node._data:
            node._data = c
        if c < node._data:
            if not node._left:
                node._left = self.Node()
            self.add(w, node._left)
        elif c > node._data
            if not node._right:
                node._right = self.Node()
            self.add(w, node._right)
        else:
            If len(w) == 1
                node._endWord = True
                return
```

```
        if not node._equal:
            node._equal = self.Node()
        self.add(W[1:], node._equal)

def suffAll(self, pattern, node):
    if node._endWord:
        yield "{0}{1}".format(P, node._data)
    if node._left:
        for w in self.sufAll(P, node._left)
            yield w
    if node._right:
        for w in self.sufAll(P, node._right)
            yield w
    if node._leftequal:
        for w in self.sufAll(P+node._data, node._equ
            yield w

def finder(self, P)
    lastP = {pat: set([]) for pat in P}
    for pat in lastP.keys():
        W = self.finder2(Pat)
        If w == None:
            return None
        else:
            comp = {Y for Y in W}
            l = list(comp)
            l1 = []
            for i in range(len(self.list)):
                if self.list[i][0] in l:
                    l1.append(self.list[i][1])
            l1.sort(reverse=True)
            l2 = []
            for i in l1:
                for j in l:
                    if (j,i) in self.list:
                        l2.append(j,i)
            return l2
```

```
def finder_2 (self, P)
    node = self.Node()
    for C in P:
        while True:
            if C > node.data:
                node = node._right
            elif C < node._left.data:
                node = node._left
            else:
                node = node._equal
                break
        if not node:
            return None
    return self.suffAll(P, node)

If __name__ = "__main__":
    s = input()
    datalist = list(input())
    Pt = completeA([datalist[i][0] for i in range(len(datalist))])
    Print(Pt.finder(s))
```

Here I used tree data structure, As I want to use different children (as information of parent) attached to it. This also takes minimum time.

- Time complexity = log(N)
- Here datalist should be given by user

Question 3

```
def findCycleLength(self):
    if self._head is None
        return 0
    c1 = self._head
    c2 = self._head
    f = 0
    while (c1 and c1.next and c2 and c2.next._next)
        If c1 == c2 and f != 0:
            count = 1
            c1 = c1.next
            while (c1 != c2):
                c1 = c1.next
                count += 1
            return count, c1
        c1 = c1._next
        c2 = c2.next._next
        f = 1
    return 0

def ~~act~~ Acutallength(self):
    r, c1 = self.findlength()
    t = self._head
    count = 0
    while( t._next != c1 & c1 != self._head):
        t = t.next
        count += 1
    return (r+count)
```

node._endword = True