PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

ESTRUCTURA DE DATOS Y PROGRAMACIÓN METÓDICA

1ra práctica (tipo a)
(Segundo semestre de 2017)

Horario 0581: prof. V.Khlebnikov
Horario 0582: prof. A.Bello R.

Duración: 1 hora 50 min.
Nota: No se puede usar ningún material de consulta.
La presentación, la ortografía y la gramática influirán en la calificación.
Puntaje total: 20 puntos

Pregunta 1 (5 puntos - 25 min.) Compararemos algunos programas asumiendo que las cadenas $s1$ y $s2$ son de la misma longitud y ambas están compuestas de los símbolos del conjunto de 26 caracteres alfabéticos en minúscula.

a) (1 punto - 5 min.) Dado el siguiente programa:

```
def anagram1(s1,s2):
    l1= list(s1); l2= list(s2)

    l1.sort(); l2.sort()

    i= 0; matches= True
    while i < len(s1) and matches:
        if l1[i] == l2[i]:
            i += 1
        else:
            matches= False

    return matches
```

¿Cuál será su estimación de este programa en términos de $\mathcal{O}()$?

b) (1 punto - 5 min.) Otra propuesta, para determinar si una cadena es un anagrama de otra cadena, es simplemente generar una lista de todas las cadenas posibles usando los caracteres de `s1` y después verificar si la cadena `s2` está entre estas. ¿Cuántas operaciones de comparación habría que hacer?

c) (1,5 puntos - 8 min.) La otra propuesta es:

```
def anagram3(s1,s2):
    c1= [0]*26; c2= [0]*26

    for i in range(len(s1)):
        j= ord(s1[i])-ord('a')
        c1[j] += 1

    for i in range(len(s2)):
        j= ord(s2[i])-ord('a')
        c2[j] += 1

    i= 0; stillOK= True
    while i<26 and stillOK:
```

```
        if c1[i] == c2[i]:
            i += 1
        else:
            stillOK= False

    return stillOK
```

Presente el cálculo de la cantidad de operaciones $T(n)$ que se hacen en este programa. ¿Cómo se estima este $T(n)$ con $\mathcal{O}()$?

d) (1,5 puntos - 8 min.) Y la última propuesta es la siguiente:

```
def anagram4(s1,s2):
    l= list(s2)

    i1= 0; stillOK= True
    while i1<len(s1) and stillOK:
        i2= 0; found= False
        while i2<len(s2) and not found:
            if s1[i1] == l[i2]:
                found= True
            else:
                i2 += 1

        if found:
            l[i2]= None
        else:
            stillOK= False

        i1 += 1

    return stillOK
```

¿Cuál será la cantidad de operaciones $T(n)$? ¡Cuidado con el cálculo! Tome como ejemplo las cadenas "python" y "typhon", esto le puede ayudar. Estime $T(n)$ con $\mathcal{O}()$.

Pregunta 2 (5 puntos - 25 min.) Answer the following questions:

a) (2 puntos - 10 min.) Use the definition of **Big-Oh** to prove that $0.001n^3 - 1000n^2 log\ n - 100n + 5$ is $\mathcal{O}(n^3)$.

b) (1 punto - 5 min.) One of the two software packages, **A** or **B**, should be chosen to process data collections, containing each up to 10^9 records. Average processing time of the package **A** is $T_A(n) = 0.001n$ milliseconds and the average processing time of the package **B** is $T_B(n) = 500\sqrt{n}$ milliseconds. Which algorithm has better performance in a "Big-Oh" sense? Work out exact conditions when these packages outperform each other.

c) (2 punto - 10 min.) Consider the following recursive algorithm for computing the sum of the first n cubes: $S(n) = 1^3 + 2^3 + ... + n^3$.

```
ALGORITHM S(n)
//Input: A positive integer n
//Output: The sum of the first n cubes
if n = 1 return 1
else return S(n - 1) + n * n * n
```

Set up and solve a recurrence relation for the number of times the algorithm's basic operation is executed.

Pregunta 3 (5 puntos - 25 min.) Extienda la especificación *FIGURAS* con las siguientes operaciones:

a) (2 puntos - 10 min.) `angulos90`, que toma una *figura* y retorna un número natural que indica la cantidad de ángulos rectos que posee la *figura*. Por ejemplo, para *f*, mostrada en Figura 1, `angulos90`(f) debe devolver 5. No puede emplear operaciones auxiliares.

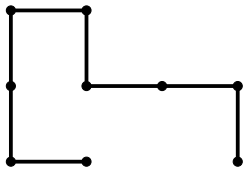

Figura 1: elemento f del tipo *figura*

b) (3 puntos - 15 min.) `es_extension_de?` toma dos elementos del tipo *figura* y retorna *true* si, a partir del último punto de la primera *figura*, se puede llegar a la segunda. Es decir, si la segunda es una extensión de la primera. Devolverá *falso* en caso contrario. Se puede emplear operaciones auxiliares.

Pregunta 4 (5 puntos - 25 min.) Considere el siguiente programa:

```
class Node:
    def __init__(self,initdata):
        self.data = initdata
        self.next = None

    def getData(self):
        return self.data

    def getNext(self):
        return self.next

    def setData(self,newdata):
        self.data = newdata

    def setNext(self,newnext):
        self.next = newnext


class UnorderedList:

    def __init__(self):
        self.head = None

    def isEmpty(self):
        return self.head == None

    def add(self,item):
        temp = Node(item)
        temp.setNext(self.head)
        self.head = temp
```

```
    def length(self):
        current = self.head
        count = 0
        while current != None:
            count = count + 1
            current = current.getNext()

        return count

    def search(self,item):
        ...

    def remove(self,item):
        ...
```

a) (2 puntos - 10 min.) Complete el método `search`.

b) (3 puntos - 15 min.) Complete el método `remove`.

La práctica fue preparada por AB(2,3) y VK(1,4)
con LaTeX en Linux Mint 18.2 Sonya

Profesores del curso: V.Khlebnikov
A.Bello R.

Pando, 7 de septiembre de 2017