Práctica 4: Estructuras de Datos y Algoritmos II UPE 2015

Ejercicio 1.
Para cada uno de los siguientes fragmentos de código, determine el tiempo de ejecución y su orden:

```
for i in range(0,n):
  sum+=1


for i in range(0,n,2):
  sum +=1


for i  in range(0,n):
  for j in range(0,n):
    sum += 1


for i in range(0,n):
  for j in range (0,n*n):
      sum+=1
```

Ejercicio 2.

Calcule el T(n).

i)

```
c = 1;
 while ( c < n ):
    algo_ de _ O(1)
    c = c * 2
```

ii)

```
 c = i;
 while ( c > 1 ):
   algo_ de _ O(1)
   c = c / 2
```

Ejercicio 3.

i)

```
def  uno (n):
   a=[range(n)for i in range(n)]
   b=[range(n)for i in range(n)]
   c=[range(n)for i in range(n)]
   for i in range(n):
     for j in range(i+1,n):
          for k in range(j+1):
               c[i][j]=c[i][j]+a[i][j]*b[i][j]
```

ii)

```
def dos(n):
    x = 0
    y = 0
    for i in range(n):
      if n % 2 == 1:
          for j in range(i,n+1):
            x +=1
          for j in range(i+1):
              y +=1
```

iii)

```
def tres(n):
    sum = 0
    for i in range(n+1):
       for j in range(i*i):
         for k in range(j+1):
            sum += 1
```

Para cada uno de los algoritmos presentados:

a) Determinar cuál es el peor caso. Fundamentar su respuesta.

b) Expresar en función de n el tiempo de ejecución del peor caso

Ejercicio 4

```
class Recurrencia():
  @staticmethod
  def rec1(n):
    if n <= 1 :
      return 1
    else:
      return Recurrencia.rec1(n-1) + Recurrencia.rec1(n-1)

  @staticmethod
  def rec2(n):
    if n <= 1:
      return 1
    else:
      return 2 * Recurrencia.rec2(n-1)

 @staticmethod
  def rec3(n):
    if n == 0:
      return 0
    elif n == 1:
      return 1
    else:
      return Recurrencia.rec3(n-2) * Recurrencia.rec3(n-2)
```

```
@staticmethod
def potencia_iter(x,n):
  if n == 0:
    potencia = 1
  elif n == 1:
   potencia = x
  else:
    potencia = x
    for i in range(2,n+1):
      potencia *= x
  return potencia

@staticmethod
def potencia_rec(x,n):
  if n == 0:
    return 1
  elif n == 1:
    return x
  elif n % 2 == 0:
    return Recurrencia.potencia_rec(x*x, n/2)
  else:
    return Recurrencia.potencia_rec (x * x, n / 2) * x
```

a) Para cada uno de los métodos presentados:
- Expresar en función de n el tiempo de ejecución.
- Analizar y resolver la correspondiente recurrencia.
- Determinar el orden de las funciones obtenidas

b) Comparar el tiempo de ejecución del método ‘rec2’ con el del método ‘rec1’.

c) Implementar un algoritmo más eficiente que el del método rec3 (es decir que el T(n) sea menor)**.**

Ejercicio 5

Resolver las siguientes recurrencias:

a) $T(n) = \begin{cases} 1 & \text{si } n = 1 \\ 8\,T(n/2) + n^3 & \text{si } n >= 2 \end{cases}$	b) $T(n) = \begin{cases} c & \text{si } n = 0 \\ d & \text{si } n = 1 \\ 2T(n/2) + n^2 & \text{si } n >= 2 \end{cases}$
c) $T(n) = \begin{cases} 2 & \text{si } n = 1 \\ T(n-1) + n & \text{si } n >= 2 \end{cases}$	d) $T(n) = \begin{cases} 2 & \text{si } n = 1 \\ T(n-1) + n/2 & \text{si } n >= 2 \end{cases}$

Ejercicio 6

Dado el siguiente método, plantear y resolver la función de recurrencia:

```
def funcion(n):
  x = 0
  if n <= 1:
    return 1
  else:
    for i in range(1,n):
      x = 1
      while x < n :
        x = x*2
   return funcion(n/2) + funcion(n/2)
```