AYED II – PRÁCTICA 1

Preliminares y verificación de programas

EJERCICIO 1.1. Sea f_n la sucesión de Fibonacci, definida mediante $f_0 = 0$, $f_1 = 1$ y $f_n = f_{n-1} + f_{n-2}$, para $n \geq 2$. Dé los valores de las siguientes expresiones:

$$a.\ \sum_{k=1}^{0} f_k \qquad b.\ \sum_{0 \leq k^2 \leq 5} f_k \qquad c.\ \sum_{0 \leq k^2 \leq 5} f_{k^2} \qquad d.\ \sum_{0 \leq k^2 \leq 5} f_k^2$$

EJERCICIO 1.2. Considere la suma $\sum_{1 \leq i \leq j \leq k \leq 4} a_{ijk}$. Despliéguela sumando:

a) Primero en k, luego en j y luego en i.

b) Primero en i, luego en j y luego en k.

EJERCICIO 1.3. Refute o valide cada una de las siguientes igualdades:

$$\left(\sum_{k=1}^{n} a_k\right)\left(\sum_{j=1}^{n} 1/a_j\right) = \sum_{k=1}^{n}\sum_{j=1}^{n} a_k/a_j = \sum_{k=1}^{n}\sum_{k=1}^{n} a_k/a_k = \sum_{k=1}^{n} n = n^2.$$

EJERCICIO 1.4. Sea s_n la suma de los primeros n términos de la serie aritmética $a, a+b, a+2b, \ldots$. Exprese s_n utilizando el símbolo Σ y luego pruebe que s_n es exactamente $an + n(n-1)b/2$.

EJERCICIO 1.5. Sea s_n la suma de los primeros n términos de la serie geométrica $a, ar, ar^2, \ldots$. Pruebe que si $r \neq 1$ entonces s_n es exactamente $\frac{a(1-r^n)}{1-r}$.

EJERCICIO 1.6. Probar por inducción que $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$.

EJERCICIO 1.7. Pruebe que $\sum_{k=0}^{n} \binom{n}{k} = 2^n$.

EJERCICIO 1.8. Pruebe que $\sum_{k=1}^{n} k\binom{n}{k} = n2^{n-1}$. (Ayuda: Derive ambos miembros de $(1+x)^n = \sum_{i=0}^{n} \binom{n}{i} x^i$).

EJERCICIO 1.9. Considere n rectas no paralelas dos a dos y sin puntos de intersección múltiples. Sea L_n el número de regiones en las que queda dividido el plano. Pruebe que $L_n = 1 + n(n+1)/2$.

EJERCICIO 1.10. Considere los siguientes programas. Para cada uno de ellos, escriba formalmente la postcondición y una adecuada precondición, y luego establezca un invariante que permita verificar el ciclo.

(1) **while** $x \neq y$
  if $x > y$ **then** $x := x - y$
  if $x < y$ **then** $y := y - x$

(2) $q, r := 0, x$
  while $r \geq y$ **do** $q, r := q + 1, r - y$

EJERCICIO 1.11. Un polinomio $a_0 + a_1 t + \cdots + a_{N-1} t^{N-1}$ se puede representar con el arreglo $[a_0, a_1, \ldots, a_{N-1}]$. Hacer lo mismo que en Ejercicio 1.10 para los siguientes programas que evalúan, de dos maneras distintas, un polinomio en x.

(1) **var** i : **int**
$i, r := 0, 0$
while $i \neq N$ **do** $r, i := rx + X[N - i - 1], i + 1$

(2) **var** i, y : **int**
$i, r, y := 0, 0, 1$
while $i \neq N$ **do** $r, i, y := r + X[i]y, i + 1, yx$

EJERCICIO 1.12. Verifique la corrección de los siguientes algoritmos recursivos, especificando previamente la precondición y la postcondición.

(1) **var** X : **array**$[0..n)$ **of int**
func $suma(X$: **array**, n : **nat**$)$ **dev: int**
if $n = 0$ **then** $s := 0$
if $n > 0$ **then**
$s := suma(X, n - 1)$
$s := s + X[n - 1]$
return s

(2) **func** $dividir(a, b$: **nat**$)$ **dev: nat**, **nat**
if $a < b$ **then** $q, r := 0, a$
if $a \geq b$ **then**
$q, r := dividir(a - b, b)$
$q := q + 1$
return q, r

AYED II – PRÁCTICA 2

Análisis de algoritmos

EJERCICIO 2.1. Hallar de manera exacta y lo más simple posible el tiempo de ejecución de los siguientes programas (por supuesto, las sumatorias deben ser eliminadas).

```
t := 0;
for i := 1 to n do
    for j := 1 to n^2 do
        for k := 1 to n^3 do t := t + 1
```

```
t := 0;
for i := 1 to n do
    for j := 1 to i do
        for k := j to n do t := t + 1
```

EJERCICIO 2.2. Determinar cuáles de las siguientes afirmaciones son válidas. Justificar.

a) $\mathcal{O}(f+g) = \mathcal{O}(max(f,g))$.
b) Si $s \in \mathcal{O}(f)$ y $r \in \mathcal{O}(g)$ entonces $s + r \in \mathcal{O}(f+g)$.
c) Si $s \in \mathcal{O}(f)$ y $r \in \mathcal{O}(g)$ entonces $s - r \in \mathcal{O}(f-g)$.
d) $2^{n+1} \in \mathcal{O}(2^n)$.
e) $(m+1)! \in \mathcal{O}(m!)$.

EJERCICIO 2.3. Ordenar intercalando "=" o "$\subset$" los $\mathcal{O}$ de las siguientes funciones, donde $0 < \varepsilon < 1$:

$$n^8, \qquad nlog(n), \qquad n^{1+\varepsilon}, \qquad (1+\varepsilon)^n, \qquad n^2/log(n), \qquad (n^2-n+1)^4.$$

EJERCICIO 2.4. Pruebe o refute las relaciones $f \in \mathcal{O}(g)$, $f \in \Omega(g)$ y $f \in \Theta(g)$ para los siguientes pares:

a) $100n + log(n),\ n + (log(n))^2$.
b) $n^2/log(n),\ n(log(n))^2$.
c) $n2^n,\ 3^n$.

EJERCICIO 2.5. Pruebe que $log(n!) \in \Theta(nlog(n))$. (Ayuda: use la fórmula de Stirling, $n! = \sqrt{2\pi n}\,(n/e)^n$)

EJERCICIO 2.6. Dé soluciones exactas para las siguientes recurrencias:

(1) $t(n) = t(n-1) + n/2,\ t(1) = 1$.
(2) $t(n) = 8t(n-1) - 15t(n-2),\ t(1) = 1,\ t(2) = 4$.
(3) $t(n) = 3t(n-2) - 2t(n-3),\ t(1) = 0,\ t(2) = 0,\ t(3) = 1$.
(4) $t(n) = 8t(n-1) - 15t(n-2), t(1) = 1, t(2) = 4$.

EJERCICIO 2.7. Demuestre que:

(1) $\sum_{i=1}^{n} i^k \in \mathcal{O}(n^{k+1})$, (2) $\sum_{i=1}^{n} i^k log(i) \in \mathcal{O}(n^{k+1}log(n))$.

EJERCICIO 2.8. Determine cuales de las siguientes funciones son uniformes:

$$n, \qquad nlog(n), \qquad n^k, \qquad n^{log(n)}, \qquad 2^n, \qquad n!\,.$$

EJERCICIO 2.9. Sea $t(n) = 2t(\lfloor n/2 \rfloor) + 2nlog(n),\ t(1) = 4$. Pruebe que $t(n) \in \mathcal{O}(nlog^2(n))$.

EJERCICIO 2.10. Para cada uno de los siguientes algoritmos escriba una ecuación recursiva asintótica para $T(n)$. Dé el orden exacto de T expresándolo de la forma más simple posible.

proc $DC(n : \textbf{nat})$
  if $n \leq 1$ **then skip**
  if $n > 1$ **then**
    for $i := 1$ **to** 8 **do** $DC(n$ **div** $2)$
    for $i := 1$ **to** n^3 **do** $dummy := 0$

proc $waste(n : \textbf{nat})$
  for $i := 1$ **to** n **do**
    for $j := 1$ **to** i **do** $write\ i, j, n$
  if $n \leq 0$ **then skip**
  if $n > 0$ **for** $i := 1$ **to** 4 **do** $waste(n$ **div** $2)$

EJERCICIO 2.11. Ordenar según $\subseteq$ los $\mathcal{O}$ de las siguientes funciones. No calcular límites, utilizar las propiedades algebraicas.

(1) $n \log 2^n$
(2) $2^n \log n$
(3) $n! \log n$
(4) 2^n

EJERCICIO 2.12. Resolver la siguiente recurrencia

$$t_n = \begin{cases} 5 & n = 1 \\ 3t_{n-1} - 2^{n-1} & n > 1 \end{cases}$$

EJERCICIO 2.13. Resolver la siguiente recurrencia

$$t_n = \begin{cases} 0 & n = 0 \vee n = 1 \\ 1 & n = 2 \\ 3t_{n-1} - 4t_{n-3} & c.c. \end{cases}$$

EJERCICIO 2.14. Calcular el orden del tiempo de ejecución del siguiente algoritmo

proc $p(n : \textbf{nat})$
  for $j := 1$ **to** 6 **do**
    if $n \leq 1$ **then skip**
    if $n \geq 2$ **then**
      for $i := 1$ **to** 3 **do** $p(n$ **div** $4)$
      for $i := 1$ **to** n^4 **do write** i

¿Qué operación u operaciones está considerando como elementales? Justificar.

EJERCICIO 2.15. Calcular el orden exacto del tiempo de ejecución de cada uno de los siguientes algoritmos:

(1) $t := 0;$
  for $i := 1$ **to** n **do**

```
        for j := 1 to i do
            for k := j to j + 3 do t := t + 1
(2)  proc p(n : nat)
        if n ≥ 2 then
            for i := 1 to 16 do p(n div 4)
            for i := 1 to n do write i
            for i := 1 to n do
                for j := 1 to n do write j
```

EJERCICIO 2.16. Ordenar las siguientes funciones según $\subset$ (incluido *estricto*) e $=$ de sus 's.

(1) $n^4 + 2\log n$
(2) $\log(n^{n^4})$
(3) $2^{4\log n}$
(4) 4^n
(5) $n^3 \log n$

Justificar sin utilizar la regla del límite.

EJERCICIO 2.17. Sea p el procedimiento dado por

```
proc p(n : nat)
    if n ≤ 1 then skip
    if n ≥ 2 then
        for i := 1 to C do p(n div D)
        for i := 1 to n^4 do write i
```

Determine posibles valores de la constante C y D de manera que el procedimiento p tenga orden

(1) $\Theta(n^4 \log n)$
(2) $\Theta(n^4)$
(3) $\Theta(n^5)$

En los casos en que

(a) $D = 2$
(b) $C = 64$

EJERCICIO 2.18. Dar algoritmos cuyos tiempos de ejecución tengan los siguientes órdenes:

(1) $n^2 + 2\log n$
(2) $n^2 \log n$
(3) 3^n

No utilizar potencia, logaritmo ni multiplicación en los programas.

EJERCICIO 2.19. Dar el orden exacto de f. Justificar.

```
func f(i, k : int) dev: int
  {pre: i ≤ k }
    if i < k then
        j := (i + k) div 2
        m := max(f(i, j), f(j + 1, k))
    else m := a[i]
    return m
```

EJERCICIO 2.20. Resolver de manera exacta la siguiente recurrencia
$2t(n) = t(n-1) + t(n-2) + 12n - 16 \qquad t(0) = 1,\ t(1) = 1.$

EJERCICIO 2.21. Calcular el orden exacto del tiempo de ejecución del siguiente algoritmo:

$p := 1$
while $p < n$ **do** $p := p * 3$

EJERCICIO 2.22. Una secuencia $x_1, ..., x_n$ se dice que tiene *orden cíclico* si existe un x_i tal que la secuencia $x_i, x_{i+1}, ..., x_n, x_1, ..., x_{i-1}$ es estríctamente creciente. Supongamos que se recibe una secuencia con orden cíclico almacenada en un arreglo X definido en $[1, n]$. Utilice una idea similar a la de búsqueda binaria para diseñar e implementar un algoritmo que encuentre el menor elemento de la secuencia en tiempo $\mathcal{O}(log\ n)$. Analice en forma detallada la complejidad.

2.1. Ejercicios adicionales

EJERCICIO 2.23. Dado un arreglo de enteros $A[1, n]$ se pide "ordenar" A sin modificarlo. Para ello, el algoritmo deberá crear un arreglo $I[1, n]$, inicializarlo con los valores $I = [1, \ldots, n]$ y realizar las modificaciones necesarias en I para que al finalizar se tenga $A[I[1]] \leq A[I[2]] \leq \ldots \leq A[I[n]]$. El algoritmo deberá devolver el arreglo I. Calcular la complejidad.

EJERCICIO 2.24. Una cima de $A[0, N)$ es un entero k en el intervalo $[0, N-1]$ que satisface que $A[0, k]$ es estrictamente creciente y $A[k, N)$ es estrictamente decreciente. Dar un algoritmo lo más eficiente posible que, asumiendo que la cima existe, la encuentra.

EJERCICIO 2.25. Resuelva la siguiente recursión:

$$t(n) = n + \sum_{i=1}^{n-1} t(i), \quad t(1) = 1.$$

EJERCICIO 2.26. Pruebe que el tiempo de ejecución de las siguientes versiones de búsqueda binaria es en ambos casos $\mathcal{O}(log(n))$.

func $bs1(X$: **array**, x : **int**, i, j : **nat**$)$ **dev: nat**
if $i = j$ **then** $k := i$
if $i < j$ **then**
    var m : **nat**
    $m := \lceil (i + j)/2 \rceil$
    if $x < X[m]$ **then** $k := bs(X, x, i, m - 1)$
    if $x = X[m]$ **then** $k := bs(X, x, m, m)$
    if $x > X[m]$ **then** $k := bs(X, x, m + 1, j)$
return k

func $bs2(X$: **array**, x : **int**, i, j : **nat**$)$ **dev: nat**
while $i < j$ **do**
    $k := (i + j)$ **div** 2
    if $x < X[k]$ **to** $j := k - 1$
    if $x = X[k]$ **to** $i, j := k, k$
    if $x > X[k]$ **to** $i := k + 1$
return k

EJERCICIO 2.27. Resuelva de manera exacta las siguientes recurrencias. Exprese la respuesta usando notación Θ, de la manera más simple posible.

(1) $t(n) = a$, si $n = 0, 1$,
(2) $t(n) = t(n-1) + t(n-2) + c$, si $n \geq 2$.
(3) $t(n) = a$, si $n = 0, 1$,
(4) $t(n) = t(n-1) + t(n-2) + cn$, si $n \geq 2$.

EJERCICIO 2.28. Ordenar según $\subset$ y $=$ los de las siguientes funciones. No calcular límites, utilizar las propiedades algebraicas.

(1) $(\log_3 2)^n$
(2) $(\log_5 2)^n$
(3) $n^{\log_3 2}$
(4) $n^{\log_5 2}$
(5) n

EJERCICIO 2.29. Resolver la recurrencia $f(n) = 8f(n-1) - 15f(n-2)$, $f(1) = 1$, $f(2) = 4$.

EJERCICIO 2.30. Dadas dos funciones $f, g : \mathbb{N} \to \mathbb{R}^+$ ¿Cuáles de las siguientes condiciones son equivalentes a $f(n) \in \mathcal{O}(g(n))$? Responder confeccionando dos listas: una que enumera aquéllas que sí son equivalentes y otra que enumera aquéllas que no lo son.

(a)	$4f(n) \in \mathcal{O}(\sqrt{3}g(n))$	(b)	$\mathcal{O}(g(n)) \subseteq \mathcal{O}(f(n))$	(c)	$\Omega(g(n)) \subseteq \Omega(f(n))$
(d)	$\Omega(f(n)) \subseteq \Omega(g(n))$	(e)	$f(n) \in \Omega(g(n))$	(f)	$g(n) \in \Omega(f(n))$
(g)	$\mathcal{O}(f(n)) \subseteq \mathcal{O}(g(n))$	(h)	$\Theta(f(n)) \subseteq \Theta(g(n))$	(i)	$\Omega(f(n)) \subseteq \mathcal{O}(g(n))$
(j)	$1/g(n) \in \Omega(1/f(n))$	(k)	$\mathcal{O}(f(n)) \subset \mathcal{O}(g(n))$	(l)	$f(n)^2 \in \mathcal{O}(g(n)^2)$

EJERCICIO 2.31. El siguiente algoritmo obtiene el mínimo elemento de un arreglo a : **array**$[1..n]$ mediante la técnica de programación "divide y vencerás". Determinar el tiempo de ejecución de *minimo*$(1, n)$.

func *minimo*$(i, k$: **int**$)$ **dev: int**
{pre: $i \leq k$*}*
if $i = k$ **then** $m := a[i]$
else
$j := (i + k)$ **div** 2
$m := \min($*minimo*$(i, j),$ *minimo(j+1,k)*$)$
{post: m *es el mínimo de* $a[i, k]$*}*

EJERCICIO 2.32. Escriba procedimientos p que tengan órdenes

(1) $\Theta(n^2 \log n)$
(2) $\Theta(n^3)$
(3) $\Theta(n^2 \log n \log n)$

EJERCICIO 2.33. Escribir un algoritmo cuyo orden exacto es $\log^2 n$. Las únicas operaciones aritméticas permitidas son: suma, resta, multiplicación y división.

EJERCICIO 2.34. Escribir un algoritmo cuyo orden exacto es $n!$. Las únicas operaciones aritméticas permitidas son: suma, resta, multiplicación y división.