UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación– Algoritmos y Estructura de Datos I
Segundo parcial – 18/11/2016

Nro. de orden:
LU:
Apellidos:
Nombres:

A

1	2	3	4	TOTAL
18	30	35	15	98

Aclaraciones: El parcial NO es a libro abierto. Cualquier decisión de interpretación que se tome debe ser aclarada y justificada. Para aprobar se requieren al menos 60 puntos. **Entregar cada ejercicio en hoja separada.**

Ejercicio 1. [20 puntos]

Dado el siguiente programa, y asumiendo que se cuenta con el siguiente `requiere`: $|a| == n \land n \ mod \ 2 == 0$

a) Dar la especificación del ciclo (P_c, Q_c, I, f_v y cota)

b) Demostrar que $(I \land f_v \leq c) \Rightarrow \neg B$

```
int contarIgualesEnEspejo(int[] a, int n) {
  int respuesta = 0;
  int i = n/2;
  while (i < n) {
    if(a[i]==a[n-i-1]) {
      respuesta++;
    }
    i++;
  }
  return respuesta;
}
```

Ejercicio 2. [30 puntos]

Dado el siguiente código, que contiene un único ciclo cuya precondición y postcondición son:
$P_c : i == n - 1 \land a == pre(a)$
$Q_c : |a| == |pre(a)| \land i == -1 \land$
$(\forall j \leftarrow [0..|a|))a_j == pre(a)_j * n.$
Asumir que se cuenta con el siguiente `requiere`: $|a| == n$

```
void multiplicandoPorN(int[] a, int n) {
  int i = n-1;
  while (i >= 0) {
    a[i] = a[i]*n;
    i = i-1;
  }
}
```

a) Determinar si el siguiente invariante es correcto para el ciclo dado:
$I : 0 \leq i \leq n \land |a| == |pre(a)| \land$
$(\forall j \leftarrow [0..i))a_j == pre(a)_j * n \land$
$(\forall j \leftarrow [i+1..n))a_j == pre(a)_j.$
Si el invariante es correcto realizar las siguientes demostraciones. Si no lo es, presentar una versión corregida y usar esa versión para las siguientes demostraciones:

b) Demostrar $P_c \Rightarrow I$

c) Demostrar $(I \land \neg B) \Rightarrow Q_c$

d) Demostrar que el cuerpo del ciclo preserva el invariante

Ejercicio 3. [35 puntos] A partir de la especificación del problema **primoMayor**, que dado un arreglo y un entero n devuelve el elemento del arreglo que es primo y es mayor a todos los otros elementos primos del arreglo, y la función auxiliar **esPrimo**:

```
problema primoMayor (a: [ℤ], n: ℤ) = res : ℤ {
    requiere |a| == n ∧ (∀i ← [0..n)) a_i > 1;
    requiere (∃i ← [0..n)) esPrimo(a_i);
    asegura (∃i ← [0..n)) a_i == res ∧ esPrimo(res) ∧ (∀i ← [0..n), esPrimo(a_i)) res ≥ a_i;
    aux esPrimo (n: ℤ) : Bool = (∀j ← [2..n)) n mod j ≠ 0;
}
```

a) Implementar un programa en C++ que respete la especificación dada y que tenga como signatura **int** primoMayor(**int**[] a, **int** n).

b) Indicar la complejidad del programa implementado.

c) ¿Es posible hacer un programa de menor complejidad, con respecto al realizado en el punto a, si se agrega la siguiente precondición?
`requiere`: $(\forall i \leftarrow [0..n))\ esPrimo(a_i)$. En caso afirmativo, escribir el programa e indicar su complejidad.

d) ¿Es posible hacer un programa de menor complejidad, con respecto a los puntos anteriores, si se cuenta con la siguiente precondición?
`requiere`: $(\forall j \leftarrow [0..n-1))a_j > a_{j+1}$ **(no considerar el requiere del punto c)**
En caso negativo, justificar. En caso afirmativo, escribir el programa e indicar su complejidad.

e) ¿Es posible hacer un programa de menor complejidad, con respecto a los puntos anteriores, si se agregan ambas precondiciones?
`requiere`: $(\forall i \leftarrow [0..n))esPrimo(a_i)$
`requiere`: $(\forall j \leftarrow [0..n-1))\ a_j > a_{j+1}$
En caso negativo, justificar. En caso afirmativo, escribir el programa e indicar su complejidad.

Ejercicio 4. [15 puntos] Implementar un programa en C++ que dado un entero positivo k, un arreglo de enteros positivos a y un entero n que representa su longitud, devuelva un entero que sea el producto del máximo valor hallado en los índices *divisibles* por k y el mínimo valor también en los índices divisibles por k. Asumir que $-n < k < n$. Respetar la siguiente signatura: **int** minMaxK(**int** k, **int**[] a, **int** n).

↓
y $k \neq 0$

2do PARCIAL - ALGORITMOS I

HOJA N° 1

FECHA 18/11/2016

Ejercicio 1

a) P_c : respuesta == 0 ∧ i == n/2 ∧ |a| == n ∧ n mod 2 == 0 ✓

Q_c : respuesta == $|[j \mid j \leftarrow [n/2 .. n), a_j == a_{n-j-1}]|$ ∧ i == ...

I : $n/2 \leq i \leq n$ ∧ respuesta == $|[j \mid j \leftarrow [n/2 .. i), a_j == a_{n-j-1}]|$ ∧ |a| == n ∧ n mod 2 == 0 ✓

f_v : n − i

cota : 0 ✓

b) qvq $(I \wedge f_v \leq c) \Rightarrow \neg B$; B : i < n ; ¬B : ¬(i < n) ✓

$(I \wedge f_v \leq c)$: $n/2 \leq i \leq n$ ∧ respuesta == $|[j \mid j \leftarrow [n/2 .. i), a_j == a_{n-j-1}]|$ ∧ |a| == n ∧ n mod 2 == 0 ∧ n − i ≤ 0

• $(I \wedge f_v \leq c) \Rightarrow n - i \leq 0 \Leftrightarrow n \leq i \Leftrightarrow \neg(i < n) \Leftrightarrow \neg B$. ■ ✓

NOTA

Ejercicio 2

a) El invariante es incorrecto, ya que $0 \leq i < n$ no vale en la última iteración y las últimas dos condiciones no valen en general.

Nuevo I: $\underbrace{-1 \leq i \leq n-1}_{I_1} \wedge \underbrace{|a| == |pre(a)|}_{I_2} \wedge \underbrace{((\forall j \leftarrow [0..i])\ a_j == pre(a)_j)}_{I_3} \wedge$
$\underbrace{((\forall j \leftarrow (i..n))\ a_j == pre(a)_j * n)}_{I_4} \wedge \underbrace{n == |a|}_{I_5}$

Además, agregué que $n == |a|$, porque sin este dato no puedo demostrar $(I \wedge \neg B) \Rightarrow Q_c$. $n-1$ es una cota más ajustada para i, aunque n no es una cota incorrecta. (no, pero te vuela de rango)

b) qvq $P_c \Rightarrow I$.

- P_c: $i == n-1 \wedge a == pre(a)$

además, por requiere, $n == |a|$. (¡lo cual implica I_5!)

- $i == n-1 \Rightarrow i \leq n-1$
- $n == |a| \Rightarrow n \geq 0 \Rightarrow n-1 \geq -1 \Rightarrow i \geq -1$ (con $i == n-1$)

$\Rightarrow -1 \leq i \leq n-1 \Leftrightarrow$ vale I_1.

- $a == pre(a) \Rightarrow |a| == |pre(a)| \Leftrightarrow$ vale I_2. ($|a| == n$)
- $a == pre(a) \Leftrightarrow ((\forall j \leftarrow [0..|a|))\ a_j == pre(a)_j) \Leftrightarrow ((\forall j \leftarrow [0..n))\ a_j == pre(a)_j) \Leftrightarrow$
$\Leftrightarrow ((\forall j \leftarrow [0..n-1])\ a_j == pre(a)_j) \Leftrightarrow ((\forall j \leftarrow [0..i])\ a_j == pre(a)_j) \Leftrightarrow$ vale I_3. ($i == n-1$)

- $i == n-1 \Rightarrow (i..n) == (n-1..n) \Leftrightarrow (i..n) == [n..n-1] \Rightarrow (i..n) == [] \Rightarrow$
$\Rightarrow ((\forall j \leftarrow (i..n))\ a_j == pre(a)_j * n) \Leftrightarrow$ vale I_4.

($(\forall j \leftarrow [])\ \langle cond \rangle$ es siempre verdadero.)

∴ vale I.

c) qvq $(I \wedge \neg B) \Rightarrow Q_c$. $\quad \neg B: i < 0 \Leftrightarrow \neg B: i \leq -1$.

- $I \wedge \neg B$: $-1 \leq i \leq n-1 \wedge |a| == |pre(a)| \wedge ((\forall j \leftarrow [0..i])\ a_j == pre(a)_j) \wedge$
$((\forall j \leftarrow (i..n))\ a_j == pre(a)_j * n) \wedge n == |a| \wedge i \leq -1$.

- $(-1 \leq i \wedge i \leq -1) \Leftrightarrow i == -1$
- $(i == -1 \wedge ((\forall j \leftarrow (i..n))\ a_j == pre(a)_j * n)) \Rightarrow ((\forall j \leftarrow (-1..n))\ a_j == pre(a)_j * n) \Leftrightarrow$
$\Leftrightarrow ((\forall j \leftarrow [0..n))\ a_j == pre(a)_j) \Rightarrow ((\forall j \leftarrow [0..|a|))\ a_j == pre(a)_j * n)$. ($|a| == n$)

∴ vale Q_c.

NOTA Continúa en la hoja siguiente

2) d) qvq I vale en E_2. Primero veo la transformación de estados del ciclo:

```
while (i>=0) {
  //estado E0;                                   => B
  //vale I ∧ B: 0≤i≤n-1 ∧ |a|==|pre(a)| ∧ ((∀j ∈ [0..I]) a_j == pre(a)_j) ∧
  //((∀j ∈ (i..n)) a_j == pre(a)_j * n) ∧ n == |a|;
  a[i;I] = a[i] * n;
  //estado E1;
  //vale i == i@E0 ∧ n == n@E0 ∧ |a| == |a@E0| ∧ a_i == a@E0_i * n ∧
  //((∀j ∈ [0..|a|), j≠i) a_j == a@E0_j );
  i = i-1;
  //estado E2;
  //vale a == a@E1 ∧ n == n@E1 ∧ i == i@E1 - 1;
  //implica i == i@E0 - 1 ∧ n == n@E0 ∧ |a| == |a@E0| ∧ a_{i@E0} == a@E0_{i@E0} * n@E0 ∧
  //((∀j ∈ [0..|a|), j≠i@E0) a_j == a@E0_j ); se deduce de E1.
  //implica i == i@E0 - 1 ∧ n == |a| ∧ |a| == |pre(a)| ∧ a_{i@E0} == a@E0_{i@E0} * n ∧
  //((∀j ∈ [0..|a|), j≠i@E0) a_j == a@E0_j ); se deduce de E0.
}
```

Ahora demuestro que cada parte del invariante vale en E_2:

I_1 vale en E_2:

$i == i@E_0 - 1$

- $0 \le i@E_0 \le n-1 \Rightarrow -1 \le i@E_0 - 1 \le n-1 \Leftrightarrow -1 \le i \le n-1 \Leftrightarrow$ vale I_1. ✓

I_2 vale en E_2: trivial, pues $|a| == |pre(a)|$. ✓

I_3 vale en E_2:

$[k \mid k \in [0..i@E_0-1]] \subset [k \mid k \in [0..|a|), k \neq i@E_0]$ pues $i@E_0 - 1 < n == |a|$

- $((\forall j \in [0..|a|), j \neq i@E_0)\ a_j == a@E_{0j}) \Rightarrow ((\forall j \in [0..i@E_0-1])\ a_j == a@E_{0j}) \Rightarrow$
$\Rightarrow ((\forall j \in [0..i@E_0-1])\ a_j == pre(a)_j) \Rightarrow ((\forall j \in [0..i])\ a_j == pre(a)_j) \Leftrightarrow$ vale I_3.

se deduce de E_0; $i == i@E_0 - 1$. ✓

I_4 vale en E_2:

$n == |a|$; $[k \mid k \in (i@E_0..n)] \subset [k \mid k \in [0..n), k \neq i@E_0]$

- $((\forall j \in [0..|a|), j \neq i@E_0)\ a_j == a@E_{0j}) \Rightarrow ((\forall j \in [0..n), j \neq i@E_0)\ a_j == a@E_{0j}) \Rightarrow$
$\Rightarrow ((\forall j \in (i@E_0..n))\ a_j == a@E_{0j}) \Rightarrow ((\forall j \in (i@E_0..n))\ a_j == pre(a)_j * n)$

porque en E_0 vale que $((\forall j \in (i@E_0..n))\ a_j == pre(a)_j * n)$. ✓

- $a@E_{0\,i@E_0} == a@E_{0\,i@E_0} * n \Rightarrow a_{i@E_0} == pre(a)_{i@E_0} * n$

porque en E_0 vale que $((\forall j \in [0..i@E_0 I])\ a_j == pre(a)_j) \Rightarrow$ en particular vale para $a_{i@E_0}$.

- $(((\forall j \in (i@E_0..n))\ a_j == pre(a)_j * n) \wedge a_{i@E_0} == pre(a)_{i@E_0} * n) \Rightarrow$
$\Rightarrow ((\forall j \in [i@E_0..n))\ a_j == pre(a)_j * n) \Leftrightarrow ((\forall j \in (i@E_0 - 1..n))\ a_j == pre(a)_j * n) \Rightarrow$
$\Rightarrow ((\forall j \in (i..n))\ a_j == pre(a)_j * n) \Leftrightarrow$ vale I_4. ✓

I_5 vale en E_2: trivial, pues $|a| == n$. ✓

$\therefore$ I vale en $E_2 \Rightarrow$ el cuerpo del ciclo preserva I. ■

NOTA

Ejercicio 3

```
bool esPrimo (int k) {
  bool res = true;                          O(1)
  if (k<=1) {                               O(1)
    res = false;                            O(1)
  }
  int i=2;                                  O(1)
  while (i*i<=k) {                          O(√k-2+1) ∈ O(√k)
    if (k%i==0) {                           O(1)
      res = false;                          O(1)
    }
    i++;                                    O(1)
  }
  return res;                               O(1)
}
```

O(1)

O(1) + O(1) ∈ O(1)
O(1) + O(1) ∈ O(1)
O(1) + O(1) ∈ O(1)
O(1) + O(1) ∈ O(1)
O(1) + O(√k) ∈ O(√k)

O(1)
O(1) + O(1) ∈ O(1)
O(1) + O(1) ∈ O(1)

O(√k) + O(1) ∈ O(k)
∴ esPrimo(k) ∈ O(√k).

Sea t el número de veces que se ejecuta la instrucción i++; hasta llegar a i*i > k, negar la guarda y salir del ciclo. Como i empieza valiendo 2, tenemos que

$2+t^2 > k \iff t^2 > k-2 \iff t > \sqrt{k-2} \iff t \geq \sqrt{k-2}+1$

t es un número de ejecuciones ⇒ t ≥ 0

∴ esPrimo(k) ∈ O(√k).

```
int primoMayor(int[] a, int n) {
  int i=0;                                  O(1)
  //busco el 1er primo que aparece
  while (i<n && !esPrimo(a[i])) {           O(Σ_{i=0}^{n-1} √a[i]) · O(1)
    i++;                                    O(1)   O(√a[i])
  }
  int res = a[i];                           O(1)
  //busco alguno mayor en el resto del arreglo
  i++;                                      O(1)   O(1)
  while (i<n) {                             O(Σ_{i=0}^{n-1} √a[i])
    if (esPrimo(a[i]) && a[i]>res) {        O(√a[i])
      res = a[i];                           O(1)
    }
    i++;                                    O(1)
  }
  return res;                               O(1)
}
```

O(1)

le llamo m

O(1) + O(Σ_{i=0}^{n-1} √a[i]) ∈ O(m)
O(1)

O(m) + O(1) ∈ O(m)

O(m) + O(1) ∈ O(m)
O(m) + O(m) ∈ O(m)

O(√a[i])
O(√a[i]) + O(1) ∈ O(√a[i])

O(√a[i]) + O(1) ∈ O(√a[i])

O(m) + O(1) ∈ O(m)
∴ primoMayor ∈ O(m).
↳ (a,n)

Por cada elemento a[i] ejecuto la función esPrimo(a[i]), de complejidad O(√a[i]). Así, al chequear todos los a[i], ejecuté $\sum_{i=0}^{n-1} \sqrt{a[i]}$ instrucciones. Recorro todos los a[i] porque i empieza valiendo 0, y debo aumentarlo n veces; en la última iteración, i empieza valiendo n-1 y ahí veo el último elemento.

∴ primoMayor ∈ $O(\sum_{i=0}^{n-1} \sqrt{a[i]})$.

continúa en la carilla siguiente.

c) Sí, porque como ya sé que todos los elementos son primos, alcanza con buscar el máximo de forma lineal, sin ejecutar un ciclo de complejidad $O(\sqrt{k})$.

```
int primoMayor2 (int[] a, int n) {
  int res = a[0]; O(1)                          O(1)
  int i = 1; O(1)                               O(1)+O(1) ∈ O(1)
(*) while (i<n) {(n-1) O(1) ∈ O(n)              O(1)+O(n) ∈ O(n)
      if (a[i] > res) {                         |
        res = a[i]; O(1)                        | O(1)
      }                                         | O(1)+O(1) ∈ O(1)
      i++; O(1)                                 |
    }                                           | O(1)+O(1) ∈ O(1)
  return res; O(1)                              O(n)+O(1) ∈ O(n)
}                                               ∴ primoMayor2 (a,n) ∈ O(n)
```

(*) Otra vez busco la cantidad t de instrucciones necesarias para negar la guarda cuando i empieza en 0:

$1 + t = n \iff t = n - 1$

Observemos que esta complejidad es menor que la anterior, porque

$$((\forall i \in [0..n)) a_i > 1) \Rightarrow \sum_{i=0}^{n-1} \sqrt{a[i]} > \sum_{i=0}^{n-1} \sqrt{1} = n - \sqrt{1} = n \Rightarrow O(n) \subset O\left(\sum_{i=0}^{n-1} \sqrt{a[i]}\right).$$

(requiere)

~~d) Sí, porque como cada elemento es mayor que todos los siguientes, el 1er primo que encontremos~~

d) No. Aunque podríamos buscar el máximo absoluto en O(1), si queremos el mayor primo vamos a tener que verificar que cada elemento de la lista sea primo, y esto es $O(\sqrt{k})$. En el peor caso, el único primo está al final de la lista, y el orden no nos da ninguna ventaja pues nuevamente es $O\left(\sum_{i=0}^{n-1} \sqrt{a[i]}\right)$.

e) Sí, porque si sabemos que todos los elementos son primos y que $a_0 > a_1 > \dots > a_{n-1}$, alcanza con devolver a_0, que es primo y mayor que todos.

```
int primoMayor3 (int[] a, int n) {
  return a[0];
}
```

(asumimos que nadie va a intentar hacer esto con un arreglo vacío. En realidad ni siquiera hace falta pasar n como parámetro)

NOTA

Corrigió: Mauro

2do PARCIAL - ALGORITMOS I

HOJA Nº 5

FECHA 13/11/2016

Ejercicio 4

```
int minMaxK (int k, int[] a, int n) {
  int minActual = a[0];
  int maxActual = a[0];
  int i = 0;

  while (i+k < n) {
    i += k;
    int actual = a[i];

    if (actual < minActual) {
      minActual = actual;
    } else if (actual > maxActual) {
      maxActual = actual;
    }
  }

  int res = minActual * maxActual;
  return res;
}
```

Podés devolver esto directamente :)

NOTA