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. [30 puntos]

a) Dada la especificación del problema `dondeEmpiezaOrden` y una posible implementación para el mismo, dar la especificación del ciclo (Pc, I, Qc, fv y cota) de forma tal que se pueda demostrar la correctitud del ciclo y de la función (NO se pide realizar ninguna demostración):

b) Demostrar que $(I \wedge \neg B) \Rightarrow Qc$

c) Demostrar que $fv < cota \Rightarrow \neg B$

```
problema dondeEmpiezaOrden (a: [ℤ], n: ℤ) = res : ℤ {
    requiere n == |a| ∧ n > 0;
    asegura enRango : 0 ≤ res ∧ res < n;
    asegura antesDesordenada : (n > 1 ∧ res > 0) ⇒ a_{res-1} ≥ a_{res};
    asegura despuesOrdenada : (∀j ← [res..n − 1))a_j < a_{j+1};
}
```

```
int dondeEmpiezaOrden(int[] a, int n) {
    int i = n - 1;
    while (i > 0 && a[i] > a[i-1]) {
        i--;
    }

    return i;
}
```

Ejercicio 2. [30 puntos] Dado el siguiente programa y un invariante I para su ciclo, decidir si I permite demostrar que el ciclo preserva el invariante. En caso afirmativo, demostrarlo. En caso negativo, corregir I y realizar la demostración.
Asumir que la especificación al problema cuenta con el siguiente requiere: $n == |a| \wedge n \ mod \ 2 == 0$.

```
int copiarAlPrincipio(int[] a, int n) {
    int i = 0;
    // E0
    while (i < n / 2) {
        a[i] = a[i + n / 2];
        i++;
    }
}
```

$I : 0 \leq i \leq n/2 \wedge (\forall j \leftarrow [0..i))a_j == a@E0_{j+n/2}$

Ejercicio 3. [15 puntos] Implementar un programa en *C++* que dados dos textos decida si son anagramas. En este caso, en vez de recibir caracteres, se recibirán dos arreglos con valores númericos que representan los textos a analizar. En particular, se debe cumplir con la siguiente especificación:

```
problema sonAnagramas (a: [ℤ], n: ℤ, b:[ℤ], m:ℤ) = res : Bool {
    requiere |a| == n;
    requiere |b| == m;
    requiere (∀i ← [0..n))0 ≤ a_i ≤ 255;
    requiere (∀i ← [0..m))0 ≤ b_i ≤ 255;
    asegura res == mismos(a, b);
}
```

La signatura de la función a implementar debe ser

```
bool sonAnagramas(int a[], int n, int b[], int m);
```

Ejercicio 4. [25 puntos]

Dado un arreglo ordenado de números enteros mayores o iguales a 0, y un entero C. Se quiere encontrar el mínimo entero T, t
que la cantidad de números del arreglo que se encuentren en el intervalo entre 0 y T, sea mayor o igual a C. Formalmente, se quier
resolver el problema que cumple la siguiente especificación.

```
problema menorEnsaguchador (a: [Z], n: Z, C: Z) = res : Z {
    requiere |a| == n;
    requiere n ≥ 1;
    requiere losValoresEstanAcotados : (∀i ← [0..n))0 ≤ a_i ∧ a_i ≤ n;
    requiere arregloOrdenado : (∀i ← [0..n − 1))a_i ≤ a_{i+1};
    requiere existeResultado : (∃T ← [0..máx(a)])cuentaHastaT(a, T) ≥ C;
    asegura sirveComoResultado : cuentaHastaT(a, res) ≥ C;
    asegura noHayOtroMasChico : ¬(∃otro ← [0..res))cuentaHastaT(a, otro) ≥ C;
    aux cuentaHastaT (a: [Z], T: Z) : Z = |[1 | x ← a, x ≤ T]|;
}
```

Para resolver este problema, se proponen los 3 siguientes programas en lenguaje *C++*. Cada uno de los códigos tiene comentado los pasos utilizados para resolverlo.

```
int menorEnsaguchador1(int a[], int n, int C){
    bool encontre = false;
    int T = 0;
    //El siguiente while, encierra una busqueda lineal para encontrar el T indicado
    while(!encontre){
        int cantidad = 0;
        int i = 0;
        //El siguiente while, encierra una busqueda lineal para buscar cuantos valores
        //del arreglo son menores o iguales a T
        while(i<n){
            if(a[i]<=T){
                cantidad++;
            }
            i++;
        }
        //Aca se chequea que la cantidad de valores sirva con respecto al parametro C
        if(cantidad>=C){
            encontre = true;
        }else{
            T++;
        }
    }
    return T;
}

int menorEnsaguchador2(int a[], int n, int C){
    bool encontre = false;
    int T = 0;
    //El siguiente while, encierra una busqueda lineal para encontrar el T indicado
    while(!encontre){
        int cantidad;

        //Si el ultimo valor es menor o igual a T, entonces todos lo son.
        if(a[n-1]<=T){
            cantidad = n;
        }else{
            //Si el primer valor es mayor a T, entonces ninguno sirve.
            if(a[0]>T){
                cantidad = 0;
            }else{
                //En caso que no se cumpla ninguna de las dos condiciones anteriores,
                //hay que buscar cuantos valores del arreglo son menores o iguales a T.
                //El siguiente while, encierra una busqueda binaria entre la primera
                //y la ultima posicion del arreglo que realiza correctamente el
                //computo deseado para la variable cantidad.
                while(...){
                    ...
                }
                cantidad = ...;
            }
        }
```

```
            //Aca se chequea que la cantidad de valores sirva con respecto al parametro C
            if(cantidad>=C){
                encontre = true;
            }else{
                T++;
            }
        }
        return T;
    }

    int menorEnsaguchador3(int a[], int n, int C){
        //El siguiente while, encierra una busqueda binaria entre -1 y
        //el ultimo valor del arreglo para encontrar el T indicado
        int inf = -1;
        int sup = a[n-1];
        while(inf+1!=sup){
            int cantidad;
            int T = (inf+sup)/2;
            //Si el ultimo valor es menor o igual a T, entonces todos lo son.
            if(a[n-1]<=T){
                cantidad = n;
            }else{
                //Si el primer valor es mayor a T, entonces ninguno sirve.
                if(a[0]>T){
                    cantidad = 0;
                }else{
                    //En caso que no se cumpla ninguna de las dos condiciones anteriores,
                    //hay que buscar cuantos valores del arreglo son menores o iguales a T.
                    //El siguiente while, encierra una busqueda binaria entre la primera
                    //y la ultima posicion del arreglo que realiza correctamente el
                    //computo deseado para la variable cantidad.
                    while(...){
                        ...
                    }
                    cantidad = ...;
                }
            }

            //Aca se chequea que la cantidad de valores sirva con respecto al parametro C
            if(cantidad>=C){
                sup = T;
            }else{
                inf = T;
            }
        }
        return sup;
    }
```

a) Indicar el orden de complejidad temporal en peor caso de los 3 algoritmos presentados y explicar como se llega a dichos órdenes.

b) Si se elimina la restricción de que el arreglo esté ordenado, hay que modificar los algoritmos para que funcionen. Si se cuenta con la función *sort*[1] que sirve para ordenar un arreglo de n enteros con un algoritmo que tiene un orden de complejidad perteneciente a $\mathcal{O}(nlog(n))$. ¿Cómo se modifican los 3 algoritmos propuestos para que sigan funcionando?¿Que complejidad temporal pasan a tener cada uno de los algoritmos?

c) Si además de eliminar la restricción de que el arreglo esté ordenado, se tiene en cuenta el requiere del problema que indica que los números del arreglo se encuentran acotados entre 0 y la longitud del arreglo. ¿Podría usarse esta restricción para modificar la complejidad de las soluciones propuestas en el inciso anterior? En caso afirmativo, indique como cambiaría el orden de complejidad. En caso negativo, explique brevemente el porqué.

[1] La función sort para un arreglo a de tamaño n se utiliza escribiendo sort(a,$a+n$)

NÚMERO DE ORDEN: 50

MOJA Nº

FECHA

① a) P_c : $n>0 \wedge i == n-1$ I_2

I : $\underbrace{0 \leq i \leq n-1}_{I_1} \wedge (\forall j \leftarrow [i \,..\, n-1))\, a[j] < a[j+1]$

Φ_c : $\underbrace{(n>1 \wedge i>0) \Rightarrow a[i-1] \geq a[i]}_{\Phi_{c_1}} \wedge \underbrace{(\forall j \leftarrow [i \,..\, n-1))\, a[j] < a[j+1]}_{\Phi_{c_2}}$

f_v : [redacted] i

cota: 0 ✓

b) ¿$(I \wedge \neg B) \Rightarrow \Phi_c$?

$\neg B$: $i \leq 0 \vee a[i] \leq a[i-1]$

$I \wedge \neg B$: $I \wedge (i \leq 0 \vee a[i] \leq a[i-1]) \Rightarrow \underbrace{(I \wedge i \leq 0)}_{\text{caso 1}} \vee \underbrace{(I \wedge a[i] \leq a[i-1])}_{\text{caso 2}}$

Puesto que mostrando que los dos casos de $I \wedge \neg B$ implican Φ_c:

*Caso 1:

• Por $I_1 \wedge i \leq 0 \Rightarrow i == 0$

∴ $(n>1 \wedge i>0) \Rightarrow a[i-1] \geq a[i]$ vale porque $i>0$ es falso (falso ⇒ verdadero / falso es siempre verdadero)

⇒ Vale Φ_{c_1}

• I_2 implica a Φ_{c_2} trivialmente ⇒ Vale Φ_{c_2} ✓

∴ el caso 1 de $I \wedge \neg B$ implica Φ_c.

*Caso 2:

• Por I_1: $0 \leq i \leq n-1$ pero $a[i-1]$ es un elemento de a ¿Por qué?

⇒ $i-1 \geq 0 \Rightarrow i \geq 1 \Rightarrow i>0$

⇒ $0 < i \leq n-1 \Rightarrow 0 < n-1 \Rightarrow 1 < n$

$(n>0 \wedge i>0) \Rightarrow a[i-1] \geq a[i]$ ∴ Φ_{c_1} vale

(ambos miembros valen verdadero) verdadero por $I \wedge \neg B$ caso 2. (verdadero ⇒ verdadero)

• De nuevo I_2 implica trivialmente a Dc_2 → Vale Dc además como I_2 de $I \wedge \neg B$ también implica Dc.

c) ¿ fv < cota → $\neg B$?

fv < cota : $i < 0$

$\neg B$: $\underbrace{i \leq 0}_{\neg B_1}$ V $\underbrace{a[i] > a[i-1]}_{\neg B_2}$

• $i < 0 \Rightarrow i \leq 0 \rightarrow \neg B_1$

• Como $p \Rightarrow (q \vee r)$ (p, q y r proposiciones) es equivalente a $(p \Rightarrow q) \vee (p \Rightarrow r)$ entonces probar fv < cota → $\neg B$ es equivalente a probar que $(fv < cota \Rightarrow \neg B_1) \vee (fv < cota \Rightarrow \neg B_2)$ y yo mostré que lo primer proposición vale, por lo tanto por lógico de conjunción, todo lo expresión vale.

(2) Me permite demostrar que sigo preservando mi invariante

I_1: $0 \le i \le n/2 \wedge (\forall j \leftarrow [0..i))\, a[j] == a@ES0[j+n/2] \wedge$ $(\forall h \leftarrow [i..n))\, a[h] == a@ES0[h]$ (I_2 / I_3)

```
// estado ES0
while (i < n/2) {
// estado ES1
    a[i] = a[i+n/2];
// estado ES2
    i++;
// estado ES3;
}
```

Sé que $I \wedge B$ valen en ES1, quiero mostrar que I vale en ES3.

* En ES1: $0 \le i \le n/2 \wedge (\forall j \leftarrow [0..i))\, a[j] == a@ES0[j+n/2] \wedge (\forall h \leftarrow [i..n))\, a[h] == a@ES0[h] \wedge i < \frac{n}{2}$
$\Rightarrow 0 \le i < \frac{n}{2}$ ✓

* En ES2: $i == i@ES1 \wedge a[i] == a@ES1[i@ES1 + n/2] \wedge$
(+) $(\forall m \leftarrow [0..n), m \neq i@ES1)\, a[m] == a@ES1[m]$ ✓

* En ES3: $a == a@ES2 \wedge i == i@ES2 + 1$ ✓

* ¿I_1 vale en ES3?

① $i@ES3 == i@ES2 + 1 \Rightarrow i@ES3 == i@ES1 + 1 \Rightarrow i@ES1 == i@ES3 - 1$

② $0 \le i@ES1 < \frac{n}{2}$ ✓

① y ② $\Rightarrow 0 \le i@ES3 - 1 < \frac{n}{2} \Rightarrow 1 \le i@ES3 < \frac{n}{2} + 1 \Rightarrow 0 \le i@ES3 \le \frac{n}{2}$
$\Rightarrow I_1$ ✓

* ¿I_2 vale en ES3?

- $(\forall j \leftarrow [0..i@ES1))\, a[j] == a@ES0[j + n/2]$
- $a@ES3 == a@ES2 \Rightarrow a[i@ES1] == a@ES1[i@ES1 + \frac{n}{2}] \wedge$
$\wedge (\forall m \leftarrow [0..n), m \neq i@ES1)\, a[m] == a@ES1[m]$ ✓
- Sé que $(\forall j \leftarrow [0..i@ES1))$ vale que $a[j] == a@ES0[j+n/2]$
y $a[i@ES1] == a@ES1[i@ES1 + n/2]$ ✓

$\Rightarrow (\forall j \leftarrow [0..i@ES1])\, a[j] == a@ES0[j + n/2]$ $\quad a@ES1 == a@ES0$ pero $i \ge i@ES1$ (por +)

y como $i@\in C1 == i@\in C3 - 1 \Rightarrow (\forall j \leftarrow [0 .. i@\in C3))\, a[j]$
$a[j + n/2] \Rightarrow I_2$

$[0 .. i@\in C3 - 1] == [0 .. i@\in C3)$

*c I_3 vale $m \in C3$?

① $(\forall h \leftarrow [i@\in 1 .. n))\, a[h] == a@\in 0[h]$

② $i@\in C1 == i@\in C3 - 1$

③ $a[i@\in C1] == a[i@\in C1 + n/2]$

④ $(\forall m \leftarrow [0 .. n), m \neq i@\in C1)\, a[m] == a@\in C1[m]$

Por ① ③ y ④ $(\forall h \leftarrow [i@\in C1 \ldots n))\, a[h] == a@\in 0[h]$

y por ② ⇒ $(\forall h [i@\in C1 .. n))\, a[h] == a@\in 0[h] \Rightarrow I_3$

y ③

(3)

```
bool sonAnagramas (int a[], int n, int b[], int m){
    int i=0; int j=0; int z=0;
    bool res = true;
    while (i<255 && res){
        j = cuento(a, n, i);
        z = cuento(b, n, i);
        if (j != z){
            res = false;
        }else{
            i++;
        }
    }
    return res;
}
int cuento (int a[], int n, int i){
    int c=0;
    int j=0;
    while(j<n){
        if(a[j]==i){
            c++;
        }
        j++;
    }
    return c;
}
```

if (n != m){ res = false; }

n == |a|