Nom i cognoms:

Algorísmia i Programació 1, GCED, 16 de gener de 2020

L'examen dura tres hores. Es valorarà la concisió, claredat i brevetat a més de la completesa i l'exactitud de les respostes. Contesteu a l'espai marcat amb bona lletra. Poseu el vostre nom a cada full. No podeu consultar cap material addicional.

1 Un xic de Python (2'5 punts)

Digueu què escriuen aquests programes en Python 3.

```
fila = [0, 0, 0]
matriu = [fila, fila, fila, fila]
print(matriu[0][0], matriu[0][1], matriu[1][0])
matriu[0][0] = 2
print(matriu[0][0], matriu[0][1], matriu[1][0])
```

```
def foo(k): k = ["banana"]

def bar(c): c.append("rainbow")

q = ["unicorn"]
foo(q)
bar(q)
print(q)
```

```
def dob (z): return z + z

def lol (f, x, y): return [[f(x), f(f(x))], [f(y), f(f(y))]]

[[i, j], k] = lol (dob, 1, [2, 3])
print(i)
print(j)
print(k)
```

Especifiqueu què fa la funció *norf* del programa següent:

```
def run(s, i, z, u):
    if i == len(s):
        print(s)
    else:
        if z > 0:
            s[i] = 0
            run(s, i + 1, z - 1, u)
        if u > 0:
            s[i] = 1
            run(s, i + 1, z, u - 1)

def norf(n):
    assert n % 2 == 0
    s = [None] * n
    run(s, 0, n // 2, n // 2)
```

Especifiqueu què fa la funció *fog* del programa següent:

```
def go(s, i, z, u):
    if i == len(s):
        print(s)
    else:
        if z < 3:
            s[i] = 0
            go(s, i + 1, z + 1, 0)
        if u < 2:
            s[i] = 1
            go(s, i + 1, 0, u + 1)

def fog(n):
    s = [None] * n
    go(s, 0, 0, 0)
```

Nom i cognoms:

2 Especificació i invariant (2'5 punts)

Considereu el codi següent:

```
def f (v, low, high):
    p = v[high]
    i = low − 1
    for j in range(low, high):
        if v[j] ≤ p:
            i += 1
            v[i], v[j] = v[j], v[i]
    v[i + 1], v[high] = v[high], v[i + 1]
    return i + 1
```

Descriviu un invariant pel bucle. L'invariant ha d'indicar les relacions que hi ha entre les variables *i*, *j*, *low* i *high* i ha de descriure què hi ha als diferents segments del vector.

Doneu una especificació de la funció *f*(), es a dir, expliqueu què fa i què retorna:

3 Omplir matrius (2'5 punts)

Considereu el fragment de codi següent:

```
using Matrix = vector<vector<int>>;

void fill(Matrix& a) {
    int n = a.size();            // nombre de files
    assert(n > 0);
    int m = a[0].size();         // nombre de columnes
    int i = 0, j = 0;            // indexos per visitar la matriu
    int v = 1;                   // valor que posarem als elements de la matriu
    bool work = true;            // per detectar quan cal o no continuar treballant
    while (work) {
        a[i][j] = v++;
        work = next(n, m, i, j);
    }
}
```

i la funció següent:

```
bool next(int n, int m, int& f, int& c) {
    ...
}
```

El codi anterior hauria d'omplir una matriu segons mostren aquests exemples:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{bmatrix}$$

Escriviu una especificació de la funció *next*:

Nom i cognoms:

Implementeu la funció *next* per generar el patró mostrat anteriorment (només cal escriure la part a completar):

Implementeu la funció *next* per tal d'omplir la matriu de la manera següent:

$$\begin{bmatrix} 1 & 2 \\ 4 & 3 \\ 5 & 6 \\ 8 & 7 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 10 & 9 & 8 & 7 & 6 \\ 11 & 12 & 13 & 14 & 15 \end{bmatrix}$$

I ara implementeu la funció *next* per tal d'omplir la matriu de la manera següent:

$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 5 & 7 \\ 6 & 8 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 4 & 9 & 10 \\ 2 & 5 & 8 & 11 & 14 \\ 6 & 7 & 12 & 13 & 15 \end{bmatrix}$$

Nom i cognoms:

4 Eficiència (2'5 punts)

La nostra empresa necessita resoldre un problema algorísmic per un client important que treballa en el mercat de *big data*. Hem demanat a quatre especialistes en algorísmia, l'Ada, la Bruna, la Clara i la Dafne, que dissenyin una funció *f* (**int** *n*) per resoldre el problema. Les quatre ens proposen solucions correctes, però molt diferents, que tenen els esquemes següents:

```
// Ada

void f (int n) {
    for (int i = 1; i ≤ n; ++i) g(i);
}
// g(n) es O(n)
```

```
void f (int n) { // Clara
    if (n > 0) {
        f(n/3); h(n); f(n/3);
    }
}
// h(n) es O(n)
```

```
void f (int n) { // Bruna
    if (n > 0) {
        f(n−1); s(); f(n−1);
    }
}
// s() es O(1)
```

```
void f (int n) { // Dafne
    for (int i = 1; i ≤ n; ++i) {
        for (int j=1; j ≤ n; j *= 2) r();
    }
}
// r() es O(1)
```

El nostre equip tècnic (la majoria no han estudiat a la UPC) ha de triar la solució més eficient. Molts es decanten per la solució de l'Ada, la més senzilla, tot i que acceptarien la de la Dafne si fos més eficient. Dubten sobre les solucions recursives (de fet, no gosen confessar que a la seva Universitat no els van ensenyar ni recursivitat ni eficiència).

Si calgués triar una solució recursiva, preferirien la de la Bruna que, com a mínim, sembla la més senzilla. La majoria queden estupefactes quan veuen la solució de la Clara. No l'acaben d'entendre.

Davant de tanta confusió, tots se't queden mirant. Ets l'única persona que ha estudiant a la UPC i que encara no ha parlat. Mentre la resta discutia sense massa rigor tècnic, tu anaves escrivint unes recurrències sobre un tovalló de paper i analitzant l'eficiència de cada proposta. El teu somriure delatava que ja havies arribat a una conclusió. La pots explicar?

Omple la taula següent indicant l'eficiència de cada algorisme utilitzant la notació O. Quin algorisme és el més eficient?

Ada	
Bruna	
Clara	
Dafne	

Utilitza l'espai que segueix per descriure els càlculs que has fet sobre el tovalló de paper per analitzar els quatre algorismes. N'hi ha prou si expliques com comptes el nombre de vegades que s'executa cada funció o descrius la sèrie que cal sumar per calcular el temps en funció d'n.