Exàmens de PRG - Problemes del Tema 2
Anàlisi d'algorismes. Eficiència. Ordenació

Curs 2018/19

P1 - Curs 18/19: 3 punts

Donada una matriu quadrada `a` de reals, el següent mètode comprova si la suma dels elements que estan per davall de la diagonal principal menys la dels que estan per damunt, coincideix amb la suma dels elements de la diagonal principal.

```
/** Precondició: a és una matriu quadrada. */
public static boolean sumBelowAbove(double[][] a) {
    double sumBelow = 0, sum = 0, sumAbove = 0;
    for (int i = 0; i < a.length; i++) {
        for (int j = 0; j < a.length; j++) {
            if (j < i) { sumBelow += a[i][j]; }
            else if (j == i) { sum += a[i][i]; }
            else { sumAbove += a[i][j]; }
        }
    }
    return sumBelow - sumAbove == sum;
}
```

Es demana:

a) (0.25 punts) Indica quina és la grandària o talla del problema, així com l'expressió que la representa.

b) (0.75 punts) Indica, i justifica, si existeixen diferents instàncies significatives per al cost temporal de l'algorisme i identifica-les si és el cas.

c) (1.50 punts) Tria una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i d'acord amb ella obtingues una expressió matemàtica, el més precisa possible, del cost temporal del mètode, distingint el cost de les instàncies més significatives en cas d'haver-les.

d) (0.50 punts) Expressa el resultat anterior utilitzant notació asimptòtica.

Solució:

a) La talla és el nombre de files (o columnes) de la matriu `a`, $n =$ `a.length`.

b) Per a una talla donada n no existeixen instàncies significatives.

c) Es pot considerar com a instrucció crítica (de cost constant) l'addició de l'element `a[i][j]` de la matriu sobre alguna de les variables `sum`, `sumBelow` o `sumAbove` o l'avaluació de la condició del `if` del bucle més intern. Així, la funció de cost es pot expressar com segueix:

$T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1 = \sum_{i=0}^{n-1} n = n^2$ *i.c.*

O comptant passos de programa: $T(n) = 1 + \sum_{i=0}^{n-1}(\sum_{j=0}^{n-1} 1 + 1) = 1 + \sum_{i=0}^{n-1}(n+1) = n^2 + n + 1$ *p.p.*

d) La funció de cost expressada en notació asimptòtica és $T(n) \in \Theta(n^2)$.

P1 - Curs 18/19: 3 punts

Una *matriu triangular superior* és una matriu quadrada els valors de la qual per davall de la diagonal principal són tots iguals a 0. El mètode recursiu `isUpperTriangular` següent comprova si les files de la matriu `m`, des de `0` fins a `nRow` compleixen aquesta propietat. Així, per a comprovar si certa matriu `mat` és triangular superior faríem la crida `isUpperTriangular(mat, mat.length - 1)`.

```
/** Precondició: m és una matriu quadrada d'enters, -1 <= nRow < m.length */
public static boolean isUpperTriangular(int[][] m, int nRow) {
    boolean res = true;
    if (nRow >= 0) {
        int j = 0;
        while (j < nRow && res) {
            if (m[nRow][j] != 0) { res = false; }
            else { j++; }
        }
        if (res) { res = isUpperTriangular(m, nRow - 1); }
    }
    return res;
}
```

Es demana:

a) (0.25 punts) Indica quina és la grandària o talla del problema, així com l'expressió que la representa.

b) (0.75 punts) Indica, i justifica, si existeixen diferents instàncies significatives per al cost temporal de l'algorisme i identifica-les si és el cas.

c) (1.50 punts) Escriu l'equació de recurrència del cost temporal en funció de la talla per a cadascun dels casos si hi haguera varis, o una única equació si només hi haguera un cas. Resol-la(es) per substitució.

d) (0.50 punts) Expressa la funció de cost utilitzant notació asimptòtica.

Solució:

a) La talla és el nombre de files en consideració de la matriu `m`, $n =$ `nRow + 1`.

b) Per a una talla donada n sí que existeixen instàncies significatives: el cas millor es dóna quan el primer element que s'analitza, `m[nRow][0]` és diferent de `0`; el cas pitjor correspon a una matriu triangular superior, ja que ha de recórrer tots els elements necessaris per a comprovar que efectivament són iguals a `0`.

c) Equacions de recurrència:

- Per al cas millor: en l'única iteració del bucle `while` es compleix que `m[nRow][0]` és diferent de `0`, es posa la variable `res` a `false`, acaba el bucle i no es fa la crida recursiva. Així el cost en el cas millor és constant, i.e. $T^m(n) = 1$.
- Per al cas pitjor: es realitza una sola crida i la grandària del problema es redueix en una unitat però ara el cost de la resta d'operacions (el bucle de cerca) és lineal amb n; per a saber això, es pot considerar com a instrucció crítica la guarda del bucle `j < nRow && res` i veure que el nombre de vegades que es repeteix és n, per als valors de `j` des de `0` fins a `nRow`. Per això,

$$T^p(n) = \begin{cases} T^p(n-1) + n & \text{si } n > 0 \\ 1 & \text{si } n = 0 \end{cases}$$

Resolent per substitució:
$T^p(n) = T^p(n-1) + n = T^p(n-2) + (n-1) + n = T^p(n-3) + (n-2) + (n-1) + n = \ldots = T^p(n-i) + \sum_{j=0}^{i-1}(n-j)$. S'arriba al cas base $T^p(0) = 1$ quan $n - i = 0$, això és, quan $i = n$.

Així, $T^p(n) = 1 + \sum_{j=0}^{n-1}(n-j) = 1 + \sum_{j=0}^{n-1} n - \sum_{j=0}^{n-1} j = 1 + n^2 - \frac{n(n+1)}{2} + n = 1 + \frac{n(n+1)}{2}$

d) Les funcions de cost dels casos millor i pitjor expressades en notació asimptòtica són $T^m(n) \in \Theta(1)$ i $T^p(n) \in \Theta(n^2)$ i, si es pren la funció de cost del cas millor com a cota inferior i la del cas pitjor com a cota superior, la funció de cost expressat en notació asimptòtica és: $T(n) \in \Omega(1), T(n) \in O(n^2)$.

RecP1 - Curs 18/19: 3 punts

Una *matriu triangular superior* és una matriu quadrada, els valors de la qual per baix de la diagonal principal són 0. El mètode iteratiu `isUpperTriangular` comprova si la matriu `m` compleix aquesta propietat.

```
/** Precondició: m és una matriu quadrada d'enters */
public static boolean isUpperTriangular(int[][] m) {
    boolean res = true;
    int i = m.length - 1;
    while (i >= 0 && res) {
        int j = 0;
        while (j < i && m[i][j] == 0) { j++; }
        if (j < i) { res = false; }
        else { i--; }
    }
    return res;
}
```

Es demana:

a) (0.25 punts) Indica la talla del problema, i l'expressió que la representa.

b) (0.75 punts) Indica, i justifica, si existeixen diferents instàncies significatives per al cost temporal de l'algorisme i identifica-les si és el cas.

c) (1.50 punts) Tria una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i d'acord amb ella calcula una expressió matemàtica, el més precisa possible, del cost temporal del mètode, distingint el cost de les instàncies més significatives en cas d'haver-les.

d) (0.50 punts) Expressa el resultat anterior utilitzant notació asimptòtica.

Solució:

a) La talla és l'ordre de la matriu `m`, la seua expressió en Java és `m.length`, a la que d'ara endavant anomenarem n.

b) Per a una talla donada n sí existeixen instàncies significatives: el cas millor es dóna quan el primer element que s'analitza, `m[m.length - 1][0]` és distint de `0`; el cas pitjor correspon a una matriu triangular superior, ja que s'ha de recórrer tots els elements necessaris per a comprovar que efectivament són iguals a `0`.

c) Es pot considerar com instrucció crítica (de cost constant) la condició de cerca del bucle més intern (`m[i][j] == 0`). Així, les funcions de cost es poden expressar:

- Per al cas millor: en l'única iteració del bucle principal es compleix que `m[m.length - 1][0]` és distint de `0`, aleshores el bucle secundari no s'executa, la variable `res` es fica a `false`, acabant el bucle principal. Per tant, el cost en el cas millor és constant, i.e. $T^m(n) = 1$ i.c.
- Per al cas pitjor: la variable `res` sempre val `true` i tots els elements examinats de la matriu contenen un 0, per tant, els bucles de cerca es converteixen en bucles de recorregut.
  $T^p(n) = \sum_{i=n-1}^{0} \sum_{j=0}^{i-1} 1 = \sum_{i=0}^{n-1} i = \frac{n(n-1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$ i.c.

d) Les funcions de cost dels casos millor i pitjor expressades en notació asimptòtica són $T^m(n) \in \Theta(1)$ i $T^p(n) \in \Theta(n^2)$ i, si es pren la funció de cost del cas millor com a quota inferior i la del cas pitjor com a quota superior, la funció de cost expressada en notació asimptòtica és: $T(n) \in \Omega(1), T(n) \in O(n^2)$.

RecP1 - Curs 18/19: 3 punts

El següent mètode calcula recursivament la suma dels elements de la submatriu de grandària `n*n` amb origen en l'element `(0,0)`, d'una matriu quadrada `m`. Noteu que en el cas de voler sumar tots els elements de la matriu, la crida inicial seria `sumar(m, m.length)`.

```
/** Precondició: m és una matriu quadrada d'enters i 0 <= n <= m.length */
public static int sumar(int[][] m, int n) {
    if (n == 0) { return 0; }
    else {
        int s = m[n - 1][n - 1];
        for (int i = 0; i < n - 1; i++) {
            s = s + m[n - 1][i] + m[i][n - 1];
        }
        return s + sumar(m, n - 1);
    }
}
```

Es demana:

a) (0.25 punts) Indica quina és la talla del problema, i l'expressió que la representa.

b) (0.75 punts) Determina si existeixen instàncies significatives. Si n'hi ha, identifica les que representen els casos millor i pitjor de l'algorisme.

c) (1.50 punts) Escriu l'equació de recurrència del cost temporal en funció de la talla per a cadascun dels casos si n'hi hagueren, o una única equació si només hi haguera un cas. Ha de resoldre's per substitució.

d) (0.50 punts) Expressa el resultat anterior mitjançant notació asimptòtica.

Solució:

a) La talla del problema és el valor del paràmetre n.

b) No existeixen instàncies significatives perquè es tracta d'un problema de recorregut.

c) Equació de recurrència:

$$T(n) = \begin{cases} k' & n = 0 \\ T(n-1) + (n-1) * k & n > 0 \end{cases}$$

Resolent per substitució:

$T(n) = T(n-1) + (n-1) * k = T(n-2) + ((n-2) + (n-1)) * k = \ldots = T(n-i) + (\sum_{j=1}^{i}(n-j)) * k$.
S'arriba al cas base $T(0) = k'$ quan $n - i = 0$, això és, quan $i = n$.

Així, $T(n) = k' + (\sum_{j=1}^{n} n - \sum_{j=1}^{n} j) * k = k' + (n^2 - \frac{n * (n+1)}{2}) * k$

d) La funció de cost expressada en notació asimptòtica és $T(n) \in \Theta(n^2)$.

Curs 2017/18

P1 - Curs 17/18: 3 punts

Donada una matriu quadrada `m` d'enters, els components de la qual són tots positius, el següent mètode comprova per a quines files la suma de tots els seus components és menor que `limit`. Per a cadascuna d'aquestes files, escriu el valor de la suma dels seus components.

```
/** Precondició: m és una matriu quadrada, i els seus elements
 *  són tots positius. El valor de limit és > 0. */
public static void sumes(int[][] m, int limit) {
    int n = m.length;
    for (int i = 0; i < n; i++) {
        int suma = m[i][0];
        int j = 1;
        while (j < n && suma < limit) {
            suma += m[i][j];
            j++;
        }
        if (suma < limit) {
            System.out.println("Fila " + i + ": " + suma);
        }
    }
}
```

Es demana:

a) (0.25 punts) Indica quina és la grandària o talla del problema, així com l'expressió que la representa.

b) (0.75 punts) Indica, i justifica, si existeixen diferents instàncies significatives per al cost temporal de l'algorisme i identifica-les si és el cas.

c) (1.50 punts) Tria una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i d'acord amb ella obtín una expressió matemàtica, el més precisa possible, del cost temporal del mètode, distingint el cost de les instàncies més significatives en cas d'haver-les.

d) (0.50 punts) Expressa el resultat anterior utilitzant notació asimptòtica.

Solució:

a) La talla del problema és la dimensió de la matriu `m` i l'expressió que la representa és `m.length`. D'ara endavant, anomenarem a aquest nombre n. Açò és, $n =$ `m.length`.

b) Sí que existeixen diferents instàncies. El cas millor es dóna quan totes les files tenen en el component 0 un valor major o igual que `limit`. El cas pitjor es dóna quan per a totes les files es compleix que la suma de tots els components de la fila no arriba a `limit`.

c) Si triem com a unitat de mesura el pas de programa, es té:

- En el cas millor: $T^m(n) = 1 + \sum_{i=0}^{n-1} 1 = 1 + n$ *p.p.*
- En el cas pitjor: $T^p(n) = 1 + \sum_{i=0}^{n-1}(1 + \sum_{j=1}^{n-1} 1) = 1 + \sum_{i=0}^{n-1} n = 1 + n^2$ *p.p.*

Si triem com a unitat de mesura la instrucció crítica i considerant com a tal, per exemple, l'avaluació de la guarda `j < n && suma < limit` (de cost unitari), es té:

- En el cas millor $T^m(n) = \sum_{i=0}^{n-1} 1 = n$ *i.c.*
- En el cas pitjor: $T^p(n) = \sum_{i=0}^{n-1} \sum_{j=1}^{n} 1 = \sum_{i=0}^{n-1} n = n^2$ *i.c.*

d) En notació asimptòtica: $T^m(n) \in \Theta(n)$ i $T^p(n) \in \Theta(n^2)$. Per tant, $T(n) \in \Omega(n)$ i $T(n) \in O(n^2)$.

P1 - Curs 17/18: 3 punts

Es desitja calcular el cost del següent mètode recursiu:

```
public static double testMethod(double[] v, int left, int right) {
    if (left > right) { return 1.0; }
    else {
        int middle = (left + right) / 2;
        return v[middle]
            * testMethod(v, left, middle - 1)
            * testMethod(v, middle + 1, right);
    }
}
```

Es demana:

a) (0.25 punts) Indica quina és la talla o grandària del problema, així com l'expressió que la representa.

b) (0.75 punts) Indica, i justifica, si existeixen diferents instàncies significatives per al cost temporal de l'algorisme i identifica-les si és el cas.

c) (1.50 punts) Escriu l'equació de recurrència del cost temporal en funció de la talla per a cadascun dels casos si hi ha més d'un, o una única equació si únicament hi haguera un cas. Ha de resoldre's per substitució.

d) (0.50 punts) Expressa el resultat anterior utilitzant notació asimptòtica.

Solució:

a) La talla del problema és el nombre d'elements del subarray `v[right..left]`, donat per l'expressió `right - left + 1` $= n$.

b) Es tracta d'un recorregut i, per tant, no hi ha instàncies significatives.

c) Plantegem l'equació de recurrència considerant les talles correponents al cas base i al cas general. Per simplicitat, aproximem a $n/2$ la talla de les dues crides del cas general.

$$T(n) = \begin{cases} 2 \cdot T(n/2) + 1 & \text{si } n > 0 \\ 1 & \text{si } n = 0 \end{cases}$$

Resolent per substitució:

$T(n) = 2 \cdot T(n/2) + 1 = 2^2 \cdot T(n/2^2) + 2 + 1 = 2^3 \cdot T(n/2^3) + 4 + 2 + 1 = 2^3 \cdot T(n/2^3) + 2^3 - 1 = \ldots = 2^i \cdot T(n/2^i) + 2^i - 1$. Si $1 \leq n/2^i < 2 \rightarrow i = \lfloor log_2 n \rfloor$, amb el que $T(n) = 2^{\lfloor log_2 n \rfloor} \cdot T(n/2^{\lfloor log_2 n \rfloor}) + 2^{\lfloor log_2 n \rfloor} - 1$.

Prenent $2^{\lfloor log_2 n \rfloor} \approx n$, es té que $T(n) = n \cdot T(1) + n - 1 = n \cdot (2 \cdot T(0) + 1) + n - 1$. S'arriba al cas base en el que $T(0) = 1$, amb el que $T(n) = 4n - 1$ *p.p.*

d) En notació asimptòtica, el cost temporal serà $T(n) \in \Theta(n)$.

RecP1 - Curs 17/18: 3 punts

Donada una matriu quadrada `m` de caràcters i un caràcter `c`, el següent mètode escriu les paraules o seqüències de caràcters que apareixen en cada fila, eliminant d'elles cada aparició de `c`.

```
/** Precondició: m és una matriu quadrada. */
public static void escriuSense(char[][] m, char c) {
    int dim = m.length;
    for (int i = 0; i < dim; i++) {
        for (int j = 0; j < dim; j++) {
            if (m[i][j] != c) { System.out.print(m[i][j]); }
        }
        System.out.println();
    }
}
```

Per exemple, si `m` = {{'e','e','l','e'}, {'m','e','m','e'}, {'n','u','l','l'}, {'c','a','s','e'}}, i `c` = 'e', aleshores el mètode escriu:

```
l
mm
null
cas
```

Es demana:

a) (0.25 punts) Indiqueu quina és la grandària o talla del problema, així com l'expressió que la representa.

b) (0.75 punts) Indiqueu, i justifiqueu, si existeixen diferents instàncies significatives per al cost temporal de l'algorisme i identifiqueu-les si és el cas.

c) (1.50 punts) Trieu una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i d'acord amb ella obteniu una expressió matemàtica, el més precisa possible, del cost temporal del mètode, distingint el cost de les instàncies més significatives en cas d'haver-les.

d) (0.50 punts) Expresseu el resultat anterior utilitzant notació asimptòtica.

Solució:

a) La talla del problema és la dimensió de la matriu `m` i l'expressió que la representa és `m.length`. D'ara endavant, anomenarem a aquest nombre n. Açò és, n = `m.length`.

b) No existeixen diferents instàncies. El mètode examina tots els caràcters de totes les files de `m`.

c) Si escollim com unitat de mesura el pas de programa, tenim:

$T(n) = 1 + \sum_{i=0}^{n-1}(1 + \sum_{j=0}^{n-1} 1) = 1 + \sum_{i=0}^{n-1}(1+n) = 1 + n + n^2$ *p.p.*

Si escollim com unitat de mesura la instrucció crítica i considerant com a tal, per exemple, l'avaluació de la condició `m[i][j] != c` (de cost unitari), tenim:

$T(n) = \sum_{i=0}^{n-1}(\sum_{j=0}^{n-1} 1) = \sum_{i=0}^{n-1} n = n^2$ *i.c.*, és a dir, n^2 *p.p.* menyspreant termes d'ordre inferior.

d) En notació asimptòtica $T(n) \in \Theta(n^2)$.

RecP1 - Curs 17/18: 3 punts

Es desitja calcular el cost del següent mètode recursiu, que donats `x` > 1 i `d` $\leq$ `x`, comprova si `x` no té cap divisor propi en el rang [`d`, `x`[:

```
/** Precondició: x > 1 && 1 < d <= x */
public static boolean senseDivisors(int x, int d) {
    if (x == d) { return true; }
    else {
        if (x % d == 0) { return false; }
        else { return senseDivisors(x, d + 1); }
    }
}
```

Es demana:

a) (0.25 punts) Indiqueu quina és la talla o grandària del problema, així com l'expressió que la representa.

b) (0.75 punts) Indiqueu, i justifiqueu, si existeixen diferents instàncies significatives per al cost temporal de l'algorisme i identifiqueu-les si és el cas.

c) (1.50 punts) Escriviu l'equació de recurrència del cost temporal en funció de la talla per a cadascun dels casos si hi ha més d'un, o una única equació si solament hi haguera un cas. Ha de resoldre's per substitució.

d) (0.25 punts) Expresseu el resultat anterior utilitzant notació asimptòtica.

e) (0.25 punts) Quin seria el cost asimptòtic en funció del valor de `x` de la crida `senseDivisors(x, 2)`, és a dir, d'averiguar si `x` és primer?

Solució:

a) La talla del problema és la diferència entre els valors dels arguments `x - d`. Anomenarem n a aquest valor d'ara endavant.

b) Es tracta d'una cerca del primer divisor propi de `x` major o igual que `d` i, per tant, hi ha instàncies significatives. En el cas millor, `d` és divisor de `x`. En el cas pitjor, `x` no té divisors propis majors o iguals que `d`.

c) Considerant el cost expressat en passos de programa, en el cas millor $T^m(n) = 1$ *p.p.* En el cas pitjor, plantegem l'equació de recurrència considerant les talles corresponents al cas base i al cas general:

$$T^p(n) = \begin{cases} 1 + T^p(n-1) & \text{si } n > 0 \quad (\text{quan } \texttt{x} > \texttt{d}) \\ 1 & \text{si } n = 0 \quad (\text{quan } \texttt{x} = \texttt{d}) \end{cases}$$

Resolent per substitució:

$T^p(n) = 1 + T^p(n-1) = 2 + T^p(n-2) = 3 + T^p(n-3) = \ldots = k + T^p(n-k)$ després de k passos de substitució. S'arriba al cas base $T^p(0) = 1$ després de $k = n$ passos de substitució, amb el que $T^p(n) = n + 1$ *p.p.*

d) En notació asimptòtica, el cost temporal és: $T^m(n) \in \Theta(1)$, $T^p(n) \in \Theta(n)$. És a dir, $T(n) \in \Omega(1)$ i $T(n) \in O(n)$.

e) En el cas particular de la crida `senseDivisors(x, 2)`, la talla és $n = \texttt{x} - 2$ i, per tant, el cost en funció de `x` és $\Omega(1), O(\texttt{x})$.

Curs 2016/17

P1 - Curs 16/17: 3 punts

Donat un array de caràcters `a` i un caràcter `c` qualsevol, el següent mètode escriu en l'eixida estàndard, línia a línia, tots els prefixes de la seqüència de caràcters en `a`, de longitud `1` en endavant, que no acaben en el caràcter `c`.

```
public static void prefixes(char[] a, char c) {
    for (int i = 0; i < a.length; i++) {
        if (a[i] != c) {
            for (int j = 0; j <= i; j++) {
                System.out.print(a[j]);
            }
            System.out.println();
        }
    }
}
```

Per exemple, si `a = {'g', 't', 'a', 't', 'c'}`, els prefixes de longituds succesives són `g`, `gt`, `gta`, `gtat` i `gtatc`. Per a `a` i `c = 't'`, el mètode escriu:

```
g
gta
gtatc
```

Es demana:

a) (0.25 punts) Indicar quina és la grandària o talla del problema, així com l'expressió que la representa.

b) (0.75 punts) Indicar si existeixen diferents instàncies significatives per al cost temporal de l'algorisme i identificar-les si és el cas.

c) (1.50 punts) Triar una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i d'acord amb ella obtenir una expressió matemàtica, el més precisa possible, del cost temporal del mètode, distingint el cost de les instàncies més significatives en cas d'haver-les.

d) (0.50 punts) Expressar el resultat anterior utilitzant notació asimptòtica.

Solució:

a) La talla del problema és el nombre d'elements de l'array `a` i l'expressió que la representa és `a.length`. D'ara endavant, anomenarem a aquest número n. Açò és, $n =$ `a.length`.

b) Sí que existeixen diferents instàncies. El cas millor es dóna quan tots els caràcters de l'array `a` són el caràcter `c`. El cas pitjor es dóna quan tots són distints del caràcter `c`.

c) Triant com a unitat de mesura el pas de programa, es té:

- En el cas millor: $T^m(n) = 1 + \sum_{i=0}^{n-1} 1 = n + 1$ *p.p.*
- En el cas pitjor: $T^p(n) = 1 + \sum_{i=0}^{n-1}(1 + \sum_{j=0}^{i} 1) = 1 + \sum_{i=0}^{n-1}(2 + i) = 1 + 2n + \sum_{i=0}^{n-1} i = 1 + n + \frac{n(n+1)}{2} = 1 + \frac{3n}{2} + \frac{n^2}{2}$ *p.p.*

Triant com a unitat de mesura la instrucció crítica i considerant com tal:

- la condició `a[i] != c` de la instrucció `if` (de cost unitari), en el cas millor es té: $T^m(n) = \sum_{i=0}^{n-1} 1 = n$ *i.c.*
- la instrucció del cos del bucle intern `System.out.print(a[j])` (de cost unitari), en el cas pitjor es té: $T^p(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{i} 1 = \sum_{i=0}^{n-1}(1 + i) = n + \sum_{i=0}^{n-1} i = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$ *i.c.*

d) En notació asimptòtica: $T^m(n) \in \Theta(n)$ i $T^p(n) \in \Theta(n^2)$. Per tant, $T(n) \in \Omega(n)$ i $T(n) \in O(n^2)$.

P1 - Curs 16/17: 3 punts

El següent mètode determina si, donat un nombre enter no negatiu `num`, el seu literal pot estar expressat en una base determinada `b` ($2 \leq$ `b` ≤ 10), comprovant que tots els dígits del nombre tenen un valor estrictament menor que la base `b`. Per exemple, el nombre 453123 pot representar un valor en base 6, 7, 8, 9 i 10 ja que tots els seus dígits són estrictament inferiors als valors d'aquestes possibles bases.

```
/** Precondició: 2 <= b <= 10 i num >= 0 */
public static boolean basePossible(int num, int b) {
    if (num == 0) { return true; }
    else {
        int ultDig = num % 10;
        if (ultDig < b) { return basePossible(num / 10, b); }
        else { return false; }
    }
}
```

Es demana:

a) (0.25 punts) Indicar quina és la grandària o talla del problema, així com l'expressió que la representa.

b) (0.75 punts) Indicar si existeixen diferents instàncies significatives per al cost temporal de l'algorisme i identificar-les si és el cas.

c) (1.50 punts) Escriure l'equació de recurrència del cost temporal en funció de la talla per a cada un dels casos si n'hi ha diversos, o una única equació si només hi hagués un cas. Cal resoldre-la per substitució.

d) (0.50 punts) Expressar el resultat anterior utilitzant notació asimptòtica.

Solució:

a) La talla del problema pot ser:

(1) el valor del primer argument del mètode, açò és, `num`; anomenarem a aquest número m.

(2) el nombre de xifres de `num`; anomenarem a aquest número n.

b) Sí que hi ha instàncies significatives, ja que és una cerca. En el millor cas, la xifra de les unitats de `num` és major o igual que la base `b` i en el cas pitjor totes les xifres de `num` són menors que `b`, açò és, `num` es pot representar en base `b`.

c) Plantegem l'equació de recurrència per a cadascuna de les dues instàncies significatives, en passos de programa, considerant cadascuna de les talles possibles.

(1) Per a talla m (valor de `num`) s'obté:

- En el cas millor: $T^m(m) = 1$ *p.p.*
- En el cas pitjor:

$$T^p(m) = \begin{cases} T^p(m/10) + 1 & \text{si } m > 0 \\ 1 & \text{si } m = 0 \end{cases}$$

Resolent-la per substitució:
$T^p(m) = T^p(m/10) + 1 = T^p(m/10^2) + 2 = \ldots = T^p(m/10^i) + i$. Si $1 \leq m/10^i < 10 \rightarrow i = \lfloor log_{10} m \rfloor$, amb el que $T^p(m) = T^p(m/10^{\lfloor log_{10} m \rfloor}) + \lfloor log_{10} m \rfloor = T^p(0) + 1 + \lfloor log_{10} m \rfloor$. S'arriba al cas base en el que $T^p(0) = 1$. Amb el que $T^p(m) = 2 + \lfloor log_{10} m \rfloor$ *p.p.*

(2) Per a talla n (nombre de xifres de `num`) s'obté:

- En el cas millor: $T^m(n) = 1$ *p.p.*
- En el cas pitjor:

$$T^p(n) = \begin{cases} T^p(n-1) + 1 & \text{si } n > 0 \\ 1 & \text{si } n = 0 \end{cases}$$

Resolent-la per substitució:
$T^p(n) = T^p(n-1) + 1 = T^p(n-2) + 2 = \ldots = T^p(n-i) + i$. S'arriba al cas base (talla 0) quan $n - i = 0 \rightarrow i = n$. Amb el que $T^p(n) = 1 + n$ *p.p.*

d) En notació asimptòtica:

(1) Per a talla m (valor de `num`), el cost temporal serà: $T^m(m) \in \Theta(1)$ i $T^p(m) \in \Theta(\log_{10} m)$, és a dir, les fites per al cost són $T(m) \in \Omega(1)$ i $T(m) \in O(\log_{10} m)$.

(2) Per a talla n (nombre de xifres de `num`), el cost temporal serà: $T^m(n) \in \Theta(1)$ i $T^p(n) \in \Theta(n)$, és a dir, les fites per al cost són $T(n) \in \Omega(1)$ i $T(n) \in O(n)$.

Com pots observar el cost temporal del mètode és el mateix només que expressat en funció de talles distintes. Recorda que el nombre de xifres d'un número enter m és $1 + \lfloor log_{10} m \rfloor$, açò és, n.

RecP1 - Curs 16/17: 4 punts

El següent mètode, donat un array `a` ordenat ascendentment, torna el número de vegades que apareix a l'array el primer número repetit (el de valor més menut) o 0 si no hi ha repetits a l'array. Per exemple, si `a = {2, 5, 5, 5, 8, 8, 9}` torna `3` (el primer número que apareix repetit és el `5` i es repeteix `3` vegades); si `a = {2, 5, 8, 9}` torna `0`.

```
/** Precondició: a ordenat ascendentment */
public static int comptarPrimerRepetit(int[] a) {
    int compt = 0, i = 0;
    boolean repe = false;
    while (i < a.length - 1 && !repe) {
        int j = i + 1;
        repe = (a[j] == a[i]);
        while (j < a.length && a[j] == a[i]) { j++; }
        if (repe) { compt = j - i; }
        i++;
    }
    return compt;
}
```

Es demana:

a) (0.25 punts) Indicar quina és la grandària o talla del problema, així com l'expressió que la representa.

b) (0.75 punts) Indicar si existeixen diferents instàncies significatives per al cost temporal de l'algorisme i identificar-les si és el cas.

c) (1.50 punts) Triar una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i d'acord amb ella obtenir una expressió matemàtica, el més precisa possible, del cost temporal del mètode, distingint el cost de les instàncies més significatives en cas d'haver-les.

d) (0.50 punts) Expressar el resultat anterior utilitzant notació asimptòtica.

e) (1 punt) Tenint en compte que l'algorisme d'ordenació per *inserció directa* vist en classe té un cost lineal en el millor cas i quadràtic en el pitjor, quin és el cost d'ordenar `a` per inserció directa i després aplicar el mètode `comptarPrimerRepetit(int[])`?

Solució:

a) La talla del problema és el nombre d'elements de l'array `a` i l'expressió que la representa és `a.length`. D'ara endavant, anomenarem a aquest número n. Açò és, $n =$ `a.length`.

b) Sí que existeixen diferents instàncies ja que es tracta d'una cerca del primer valor repetit que conté un altra cerca per tal de comptar quantes vegades es repeteix aquest valor.

El cas millor es dóna quan `a[0]` està repetit en `a[1]`, és a dir, `a[0] == a[1]`.

El cas pitjor es dóna quan no hi ha elements repetits a l'array i també quan `a[0]` està repetit en tot l'array, és a dir, $\forall$`i:1` $\leq$ `i < a.length: a[0] == a[i]`.

c) Triant com a unitat de mesura el pas de programa, es té:

- Cas millor: $T^m(n) = 3$ *p.p.*
- Cas pitjor: $T^p(n) = 1 + \sum_{i=0}^{n-1} 1 = n + 1$ *p.p.*

Triant com a unitat de mesura la instrucció crítica i considerant com tal la guarda del bucle intern `j < a.length && a[j] == a[i]` (de cost unitari), es té:

- Cas millor: $T^m(n) = 2$ *i.c.*
- Cas pitjor: $T^p(n) = \sum_{i=0}^{n-2} 1 = n - 1$ *i.c.*

d) En notació asimptòtica: $T^m(n) \in \Theta(1)$ i $T^p(n) \in \Theta(n)$. Per tant, $T(n) \in \Omega(1)$ i $T(n) \in O(n)$.

e) La resolució del problema aplicant primer inserció directa i després el mètode `comptarPrimerRepetit` tindrà les següents fites de complexitat:

- En el cas millor, l'array està ordenat ascendentment i `a[0]` està repetit en `a[1]`. Aplicar l'ordenació per inserció directa tindrà un cost $\Theta(n)$ i aplicar el mètode `comptarPrimerRepetit` serà $\Theta(1)$. Per tant, el cost en el cas millor serà lineal amb la talla del problema.
- En el cas pitjor, els elements de l'array estan ordenats descendentment i no hi ha repetits. El cost de l'ordenació per inserció directa serà $\Theta(n^2)$ i aplicar el mètode `comptarPrimerRepetit` serà $\Theta(n)$. Per tant, el cost en el cas pitjor serà quadràtic amb la talla del problema.

RecP1 - Curs 16/17: 3 punts

El següent mètode calcula la suma dels bits de la representació en binari d'un enter no negatiu `num`.

```
/** Precondició: num >= 0 */
public static int sumaBits(int num) {
    if (num <= 1) { return num; }
    else {
        int ultBit = num % 2;
        return sumaBits(num / 2) + ultBit;
    }
}
```

Es demana:

a) (0.25 punts) Indicar quina és la grandària o talla del problema, així com l'expressió que la representa.

b) (0.5 punts) Indicar si existeixen diferents instàncies significatives per al cost temporal de l'algorisme i identificar-les si és el cas.

c) (1.5 punts) Escriure l'equació de recurrència del cost temporal en funció de la talla per a cada un dels casos si n'hi ha diversos, o una única equació si només hi hagués un cas. Cal resoldre-la per substitució.

d) (0.75 punts) Expressar el resultat anterior utilitzant notació asimptòtica.

Solució:

a) La talla del problema és `num`. D'ara endavant, anomenarem a aquest número n. Açò és, $n =$ `num`.

b) No existeixen instàncies significatives; per a una mateixa talla n, el mètode sempre realitza el mateix número d'operacions.

c) L'equació de recurrència es defineix com:

$$T(n) = \begin{cases} T(n/2) + 1 & \text{si } n > 1 \\ 1 & \text{si } n \leq 1 \end{cases}$$

Resolent per substitució:

$T(n) = T(n/2) + 1 = T(n/4) + 2 = T(n/8) + 3 = \ldots = T(n/2^i) + i$.

Si $n/2^i = 1 \rightarrow i = log_2 n$, amb el que $T(n) = T(n/2^{log_2 n}) + log_2 n = T(1) + log_2 n = 1 + log_2 n$ *p.p.*

d) En notació asimptòtica: $T(n) \in \Theta(log_2 n)$

Una solució alternativa consisteix a considerar com talla el número de xifres de la representació en base 2 del valor `num`. En aquest cas:

a) m = número de xifres de la representació en base 2 de `num`.

b) No existeixen instàncies significatives; per a una mateixa talla m, el mètode sempre realitza el mateix número d'operacions.

c) L'equació de recurrència es defineix com:

$$T(m) = \begin{cases} T(m-1) + 1 & \text{si } m > 1 \\ 1 & \text{si } m \leq 1 \end{cases}$$

Resolent per substitució:

$T(m) = T(m-1) + 1 = T(m-2) + 2 = T(m-3) + 3 = \ldots = T(m-i) + i$.

Si $m - i = 1 \rightarrow i = m - 1$, amb el que $T(m) = T(1) + m - 1 = 1 + m - 1 = m$ *p.p.*

d) En notació asimptòtica: $T(m) \in \Theta(m)$

Com es pot observar, el cost temporal del mètode és el mateix, encara que expressat en funció de talles diferents. Recorda que el nombre de xifres de la representació en base 2 d'un número enter n és $1 + \lfloor log_2 n \rfloor$, açò és, m.

Curs 2015/16

P1 - Curs 15/16: 3 punts

El següent mètode comprova si un array d'enters es troba ordenat de manera ascendent entre les posicions `ini` i `fi` inclusivament:

```
/** Torna true si a[ini..fi] està ordenat ascendentment,
 *  false en cas contrari. */
public static boolean ordenat(int[] a, int ini, int fi) {
    if (ini >= fi) { return true; }
    else {
        if (a[ini] > a[ini + 1] || a[fi] < a[fi - 1]) { return false; }
        else { return ordenat(a, ini + 1, fi - 1); }
    }
}
```

Es demana:

a) (0.25 punts) Indiqueu quina és la grandària o talla del problema, així com l'expressió que la representa.

b) (0.5 punts) Indiqueu si existeixen diferents instàncies significatives per al cost temporal de l'algorisme, identificant-les si és el cas.

c) (1.5 punts) Doneu la relació de recurrència per al cost, resolent-la per substitució, distingint el cost de les instàncies més significatives en cas d'haver-les.

d) (0.75 punts) Expresseu el resultat anterior utilitzant notació asimptòtica.

Solució:

a) La talla del problema és el nombre d'elements de l'array `a` en consideració, i l'expressió que la representa és `fi - ini + 1`. D'ara endavant, anomenarem a aquest valor n. És a dir, $n =$ `fi - ini + 1`.

b) Es tracta d'un problema de cerca (ascendent i descendentment, de forma simultània) i, per tant, per a una mateixa talla presenta instàncies diferents. El cas millor és quan la primera o l'última parella d'elements en consideració no estan ordenades. El cas pitjor és quan `a[ini..fi]` està ordenat.

c) Per obtenir el cost del mètode, estudiem cadascuna de les dues instàncies significatives:

- Cas millor: $T^m(n) = 1$ *p.p.*
- Cas pitjor:

$$T^p(n) = \begin{cases} T^p(n-2)+1 & \text{si } n > 1 \\ 1 & \text{si } n \leq 1 \end{cases}$$

expressat en passos de programa (p.p.).
Resolent per substitució: $T^p(n) = T^p(n-2)+1 = T^p(n-4)+2 = \ldots = T^p(n-2\cdot i)+i$. S'arriba al cas base, talla 0 o 1, per a $i = n/2$. Amb el que

$$T^p(n) = 1 + \frac{n}{2} \text{ p.p.}$$

d) En notació asimptòtica: $T^m(n) \in \Theta(1)$ i $T^p(n) \in \Theta(n)$. Per tant, $T(n) \in \Omega(1)$ i $T(n) \in O(n)$, és a dir, el cost temporal depèn, com a molt, linealment de la talla del problema.

P1 - Curs 15/16: 3 punts

El següent mètode transposa una matriu quadrada:

```
/** Canvia la matriu m a la seua transposada.
 *  Precondició: m es una matriu quadrada. */
public static void transposada(int[][] m) {
    for (int i = 0; i < m.length; i++) {
        for (int j = 0; j < i; j++) {
            int aux = m[i][j];
            m[i][j] = m[j][i];
            m[j][i] = aux;
        }
    }
}
```

Es demana:

a) (0.25 punts) Indiqueu quina és la grandària o talla del problema, així com l'expressió que la representa.

b) (0.5 punts) Indiqueu si existeixen diferents instàncies significatives per al cost temporal de l'algorisme, identificant-les si és el cas.

c) (1.5 punts) Escolliu una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i d'acord amb ella obtenir una expressió matemàtica, el més precisa possible, del cost temporal del mètode, distingint el cost de les instàncies més significatives en cas d'haver-les .

d) (0.75 punts) Expresseu el resultat anterior utilitzant notació asimptòtica.

Solució:

a) La talla del problema és la dimensió de la matriu `m`, és a dir, `m.length`. D'ara endavant, anomenarem a aquest valor n. És a dir, $n =$ `m.length`.

b) No existeixen diferents instàncies.

c) Triant com a unitat de mesura el pas de programa, es té:

$$T(n) = 1 + \sum_{i=0}^{n-1}(1 + \sum_{j=0}^{i-1} 1) = 1 + \sum_{i=0}^{n-1}(1+i) = 1 + \frac{(1+n)\cdot n}{2} = 1 + \frac{n}{2} + \frac{n^2}{2} \; p.p.$$

Si es pren com a instrucció crítica l'avaluació de la guarda `j < i`, de cost unitari, i es compta com a mesura del cost el nombre de vegades que s'executa aquesta guarda, s'arriba a l'expressió:

$$T(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{i} 1 = \sum_{i=0}^{n-1}(1+i) = \frac{n}{2} + \frac{n^2}{2} \; i.c.$$

que és com l'expressió anterior, tot i que menyspreant el terme de menor ordre.

d) En notació asimptòtica: $T(n) \in \Theta(n^2)$.

RecP1 - Curs 15/16: 3 punts

Mitjançant el següent mètode es comprova si totes les files de la matriu `m`, quadrada, de `double`, sumen sempre un mateix valor. Se sap que `m` és almenys d'ordre 2 (té un nombre de files i columnes major o igual que 2).

```
/** Precondició: m és quadrada d'ordre major o igual que 2. */
public static boolean sumenIgual(double[][] m) {
    // Es calcula la suma de la primera fila:
    int sum0 = 0;
    for (int i = 0; i < m.length; i++) { sum0 += m[0][i]; }

    boolean sumIgual = true;
    // Es calcula la suma de cada fila posterior:
    for (int i = 1; i < m.length && sumIgual; i++) {
        int sumFil = 0;
        for (int j = 0; j < m.length; j++) { sumFil += m[i][j]; }
        sumIgual = (sum0 == sumFil);
    }
    return sumIgual;
}
```

Es demana:

a) (0.25 punts) Indicar quina és la grandària o talla del problema, així com l'expressió que la representa.

b) (0.75 punts) Indicar si existeixen diferents instàncies significatives per al cost temporal de l'algorisme i identificar-les si és el cas.

c) (1.50 punts) Triar una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i d'acord amb ella obtenir una expressió matemàtica, el més precisa possible, del cost temporal del mètode, distingint el cost de les instàncies més significatives en cas d'haver-les.

d) (0.50 punts) Expressar el resultat anterior utilitzant notació asimptòtica.

Solució:

a) La talla del problema és la dimensió de la matriu `m`, és a dir, `m.length`. D'ara endavant, anomenarem a aquest nombre n. Açò és, $n =$ `m.length`.

b) Si que existeixen diferents instàncies ja que es tracta de la cerca de la primera fila de la matriu `m`, si existeix, que la seua suma siga diferent de la de la primera.

El cas millor és quan la suma dels elements de la primera fila de `m` és diferent de la suma dels de la segona (amb la qual cosa es recorreran solament els elements de la primera i segona files), mentre que el cas pitjor es donarà quan la suma dels elements de totes les files de la matriu `m` val el mateix (i es recorreran per complet totes les files).

c) Triant com a unitat de mesura el pas de programa, es té:

- Cas millor:

$$T^m(n) = 1 + \sum_{i=0}^{n-1} 1 + \sum_{j=0}^{n-1} 1 = 2n + 1 \ p.p.$$

- Cas pitjor:

$$T^p(n) = 1 + \sum_{i=0}^{n-1} 1 + \sum_{i=1}^{n-1}(1 + \sum_{j=0}^{n-1} 1) = 1 + n + \sum_{i=1}^{n-1}(1 + n) = 1 + n + (n-1)(n+1) = n^2 + n \ p.p.$$

Triant com a unitat de mesura la instrucció crítica i considerant com tal la instrucció `sum0 += m[0][i];` del primer bucle i la instrucció `sumFil += m[i][j];` del segon bucle (ambdues de cost unitari), es té:

- Cas millor:

$$T^m(n) = \sum_{i=0}^{n-1} 1 + \sum_{j=0}^{n-1} 1 = 2n \ i.c.$$

- Cas pitjor:

$$T^p(n) = \sum_{i=0}^{n-1} 1 + \sum_{i=1}^{n-1}\sum_{j=0}^{n-1} 1 = n + \sum_{i=1}^{n-1} n = n + n(n-1) = n^2 \ i.c.$$

d) En notació asimptòtica: $T^m(n) \in \Theta(n)$ i $T^p(n) \in \Theta(n^2)$. Per tant, $T(n) \in \Omega(n)$ i $T(n) \in O(n^2)$.

RecP1 - Curs 15/16: 3 punts

Donada la següent implementació d'un algorisme recursiu que resol la multiplicació *a la russa* de dos nombres naturals `a` i `b`:

```
/** Precondició: a >=0 i b >= 0. */
public static int producteRus(int a, int b) {
    if (b == 0) { return 0; }
    else {
        if (b % 2 == 0) { return producteRus(a * 2, b / 2); }
        else { return a + producteRus(a * 2, b / 2); }
    }
}
```

Es demana: analitzar el cost temporal d'aquest algorisme. En concret:

a) (0.25 punts) Indicar quina és la talla del problema, i quina expressió la defineix.

b) (0.5 punts) Identificar, cas que les hi haguera, les instàncies del problema que representen el cas millor i pitjor de l'algorisme.

c) (1.5 punts) Triar una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i d'acord amb ella obtenir una expressió matemàtica, el més precisa possible, de l'equació de recurrència del cost temporal en funció de la talla (una única equació si no hi haguera instàncies significatives; dues equacions, corresponents als casos millor i pitjor, en el cas que el problema tinguera instàncies significatives). Resoldre-la(les) per substitució.

d) (0.75 punts) Expressar el resultat anterior utilitzant notació asimptòtica.

Solució:

a) La talla és el valor del segon argument, l'enter `b`, que anomenarem n d'ara endavant. L'enter `a`, no influeix en el nombre de crides recursives que es generen, ni en el cost temporal de les operacions no recursives, per la qual cosa no té cap efecte en el cost de l'algorisme.

b) Atès que es tracta d'un algorisme que *recorre* la seqüència de valors $b, \frac{b}{2}, \frac{b}{4}, \frac{b}{8} \ldots$ no existeixen instàncies significatives.

c) El cost de les operacions que s'executen en el cas base ($n = 0$) és constant, suposem que igual a $c_0 \in \Re^+$. El cost de les operacions no recursives que s'executen en el cas general ($n > 0$) és també constant, suposem que igual a $c_1 \in \Re^+$. En el cas general es realitza una crida recursiva en la qual la talla del problema (n) es redueix a la meitat. En conseqüència, es pot escriure la següent equació de recurrència:

$$T(n) = \begin{cases} c_0 & \text{si } n = 0 \\ c_1 + T(\frac{n}{2}) & \text{si } n > 0 \end{cases}$$

expressada en passos de programa (*p.p.*).

Resolent per substitució: $T(n) = c_1 + T(\frac{n}{2^1}) = 2c_1 + T(\frac{n}{2^2}) = 3c_1 + T(\frac{n}{2^3}) = \ldots = kc_1 + T(\frac{n}{2^k})$. En el cas base, la talla és 0, i el paràmetre de la funció T hauria de prendre aquest valor, donant lloc a la

igualtat: $\frac{n}{2^k} = 0$. Per a poder trobar el valor de k, canviem aquesta igualtat a $\frac{n}{2^k} = 1$ i obtenim el valor de $k = \log_2 n$. Substituint el valor de k en el terme general $kc_1 + T(\frac{n}{2^k})$, obtenim:

$$T(n) = c_1 \log_2 n + T(1) = c_1 \log_2 n + (c_1 + T(0)) = c_1 \log_2 n + c_1 + c_0 \; p.p.$$

d) En notació asimptòtica: $T(n) \in \Theta(\log n)$, és a dir, el cost temporal de l'algorisme depèn logarítmicament de la talla del problema.

Curs 2014/15

P1 - Curs 14/15: 7 punts

Per tal de determinar quants elements d'un array `a` són menors que un valor donat `x`, es proposen les dues solucions següents en Java, on la primera d'elles suposa que l'array està ordenat ascendentment.

- Solució 1

```
/** Torna el nombre d'elements de l'array a menors que x
 *  Precondició: a està ordenat ascendentment.  */
public static int comptarMenorsX1(int[] a, int x) {
    int i = a.length - 1;
    while (i >= 0 && a[i] >= x) { i--; }
    return i + 1;
}
```

- Solució 2

```
/** Torna el nombre d'elements de l'array a menors que x
 *  Precondició: 0 <= pos <= a.length.
 *  Crida inicial: int res = comptarMenorsX2(a, x, 0); */
public static int comptarMenorsX2(int[] a, int x, int pos) {
    if (pos == a.length) { return 0; }
    else if (a[pos] < x) { return 1 + comptarMenorsX2(a, x, pos + 1); }
    else { return comptarMenorsX2(a, x, pos + 1); }
}
```

Es demana:

i. (3 punts/proposta) Per a cada solució proposada:

a) (*0.25 punts*) Indicar quina és la grandària o talla del problema, així com l'expressió que la representa.

b) (*0.5 punts*) Indicar si existeixen diferents instàncies significatives per al cost temporal de l'algorisme i identificar-les si és el cas.

c) (*1.5 punts*) En el cas del mètode iteratiu, triar una unitat de mesura per a l'estimació del cost (pas de programa, instrucció crítica) i d'acord amb ella, obtindre una expressió matemàtica, el més precisa possible, del cost temporal, a nivell global o en les instàncies més significatives si n'hi ha.
En el cas del mètode recursiu, escriure l'equació de recurrència del cost temporal en funció de la talla per a cada un dels casos si n'hi ha diversos, o una única equació si només hi hagués un cas. Cal resoldre-la per substitució.

d) (*0.75 punts*) Expressar el resultat anterior utilitzant notació asimptòtica.

Solució:

Anàlisi per al mètode de la Solució 1:

a) La grandària o talla del problema és el nombre d'elements de l'array `a` i l'expressió que la representa és `a.length`. D'ara endavant, anomenarem a aquest número n. Açò és, $n =$ `a.length`.

b) Es tracta d'un problema de cerca (iterativa descendent) i, per tant, per a una mateixa talla sí que presenta instàncies distintes. Com l'array `a` està ordenat ascendentment, el cas millor es dóna quan en l'última posició de l'array trobem un valor menor que `x` (`a[a.length - 1]` $<$ `x`), açò és, tots els elements de l'array són menors que `x`. El cas pitjor ocorre quan tots els elements de l'array són majors o iguals que `x` ($\forall$`i`, `0`$\leq$`i`$<$`a.length`, `a[i]` $\geq$ `x`).

c) Triant com unitat de mesura el pas de programa, i considerant una passada del bucle com un pas de programa i la resta d'operacions com un únic pas de programa, la funció de cost temporal en el cas millor serà $T^m(n) = 1$ *p.p.* i en el cas pitjor $T^p(n) = 1 + \sum_{i=0}^{n-1} 1 = n + 1$ *p.p.*

Triant com unitat de mesura la instrucció crítica, l'única instrucció que podem considerar com crítica és la guarda del bucle: `i >= 0 && a[i] >= x` (s'executa sempre una vegada més que qualsevol de les del cos del bucle). En el cas millor només s'executarà una vegada. En el cas pitjor

es repetirà tantes vegades com elements tinga l'array més una (quan es fa falsa). La funció de cost temporal, considerant la instrucció crítica de cost unitari, en el cas millor serà $T^m(n) = 1$ *i.c.* i en el cas pitjor $T^p(n) = \sum_{i=0}^{n} 1 = n + 1$ *i.c.*

d) En notació asimptòtica: $T^m(n) \in \Theta(1)$ i $T^p(n) \in \Theta(n)$. Per tant, $T(n) \in \Omega(1)$ i $T(n) \in O(n)$, és a dir, el cost temporal està fitat inferiorment per una funció constant i superiorment per una funció lineal amb la talla del problema.

Anàlisi per al mètode de la Solució 2:

a) La grandària o talla del problema és el nombre d'elements de l'array `a` en consideració i l'expressió que la representa és `a.length - pos`. D'ara endavant, anomenarem a aquest número n. Açò és, $n =$ `a.length - pos`.

b) Es tracta d'un problema de recorregut (recursiu ascendent) i, per tant, per a una mateixa talla no presenta instàncies distintes.

c) Per obtenir el cost del mètode, plantegem l'equació de recurrència:

$$T(n) = \begin{cases} T(n-1) + k & \text{si } n > 0 \\ k' & \text{si } n = 0 \end{cases}$$ sent k i k' constants positives, en alguna unitat de temps.

Resolent per substitució: $T(n) = T(n-1) + k = T(n-2) + 2k = \ldots = T(n-i) + ik$. En arribar al cas base, per a talla 0, $n - i = 0 \rightarrow n = i$. Amb el que $T(n) = k' + nk$.

d) En notació asimptòtica: $T(n) \in \Theta(n)$, és a dir, el cost temporal depèn linealment de la talla del problema.

ii. (1 punt) Tenint en compte que l'algorisme d'ordenació per *inserció directa* vist en classe té un cost lineal en el millor cas i quadràtic en el pitjor, i que per a la primera solució caldria ordenar l'array prèviament, comparar el cost temporal total de comptar els valors menors que `x` en un array `a` no necessariament ordenat, quan es resol el problema mijançant els següents algorismes:

- Algorisme 1: Primer ordenar `a` per inserció directa i després aplicar el mètode `comptarMenorsX1(int[], int)`.
- Algorisme 2: Aplicar directament a l'array `a` el mètode `comptarMenorsX2(int[], int, int)`.

Solució:

La resolució del problema aplicant l'Algorisme 1 tindrà les següents fites de complexitat:

- En el cas millor, l'array està ordenat ascendentment i tots els seus elements són menors que `x`. Aplicar l'ordenació per inserció directa tindrà un cost $\Theta(n)$ i comptar el nombre d'elements menors que `x` serà $\Theta(1)$. Per tant, el cost en el cas millor serà lineal amb la talla del problema.
- En el cas pitjor, els elements de l'array estan ordenats descendentment i tots ells són majors o iguals que `x`. El cost de l'ordenació per inserció directa serà $\Theta(n^2)$ i comptar el nombre d'elements menors que `x` serà $\Theta(n)$. Per tant, el cost en el cas pitjor serà quadràtic amb la talla del problema.

La resolució del problema aplicant l'Algorisme 2, siga quin siga l'array `a`, tindrà sempre un cost $\Theta(n)$. Per tant, podem concloure que l'Algorisme 2 és més eficient quan el problema no està restringit a arrays ordenats.

Curs 2013/14

P1 - Curs 13/14: 3 punts

Considerar el següent mètode recursiu en Java que comprova si tots els elements del subarray `a[pos..a.length-1]` apareixen formant una progressió aritmètica de diferència `d`:

```
/** Retorna true si per a tota parella a[i], a[i + 1] en a[pos..a.length - 1]
 *  es compleix que a[i + 1] = a[i] + d, i false en cas contrari.
 *  Precondició: a.length >= 1 && 0 <= pos <= a.length - 1. */
public static boolean progAritmetica(int[] a, int d, int pos) {
    if (pos == a.length - 1) { return true; }
    else { return a[pos + 1] == a[pos] + d  && progAritmetica(a, d, pos + 1); }
}
```

Es demana:

a) (0.25 punts) Indicar quina és la talla del problema i quina expressió la defineix.

b) (0.5 punts) Identificar, cas de que les haguera, les instàncies del problema que representen el cas millor i pitjor de l'algorisme.

c) (1.5 punts) Escriure l'equació de recurrència del cost temporal en funció de la talla per a cadascun dels casos si n'hi ha diversos, o una única equació si només hi hagués un cas. Resoldre-la(les) per substitució.

d) (0.75 punts) Expressar el resultat anterior utilitzant notació asimptòtica.

Solució:

a) La talla és el nombre d'elements de l'array en consideració en cada crida. L'expressió que la defineix és `a.length-pos`, que anomenarem n.

b) Existeixen diferents instàncies per què és un problema de cerca en un array.

El *cas millor* es presenta quan la condició no es compleix per al primer parell d'elements considerats, és a dir, quan la diferència entre les dues primeres components del subarray considerat no és `d` (açò és, `a[pos + 1]` $\neq$ `a[pos] + d`), de manera que torna `false` sense fer cap crida recursiva.

El *cas pitjor* ocorre quan la condició es compleix per a tots els parells d'elements, és a dir, quan la diferència entre tots els parells d'elements consecutius del subarray considerat és `d`, de manera que torna `true` quan s'arriba al cas base, després de realitzar el número màxim de crides recursives.

c) En el cas millor, per a qualsevol talla es té: $T^m(n) = k$, sent k una constant positiva, en alguna unitat de temps.

En el cas pitjor, el cost s'expressa recurrentment com:

$$T^p(n) = \begin{cases} k_0 & si\ n = 1 \\ k_1 + T^p(n-1) & si\ n > 1 \end{cases}$$

sent k_0, k_1 constants positives, en alguna unitat de temps. Resolent per substitució:

$$\begin{array}{rl} T^p(n) & = k_1 + T^p(n-1) = \ 2 \cdot k_1 + T^p(n-2) = \ 3 \cdot k_1 + T^p(n-3) = \cdots = \\ & = \ i \cdot k_1 + T^p(n-i) = \cdots = \\ & (cas\ base: \ n - i = 1, \ i = n-1) \\ & = k_1 \cdot (n-1) + T^p(1) = k_1 \cdot n + (k_0 - k_1) \end{array}$$

d) En notació asimptòtica: $T^m(n) \in \theta(1)$, $T^p(n) \in \theta(n)$. Per tant, $T(n) \in \Omega(1)$, $T(n) \in O(n)$.

P1 - Curs 13/14: 4 punts

El següent mètode, `triangle(int)`, determina, escrivint-los, el número de triangles rectangles de costats enters i hipotenusa `h`:

```
/** El mètode compta, escrivint-los, tots els triangles
 *  rectangles de costats enters i hipotenusa h. */
public static int triangle(int h) {
    int cont = 0;
    for (int c1 = 4; c1 < h; c1++) {
        for (int c2 = 3; c2 < c1; c2++) {
            if (c1 * c1 + c2 * c2 == h * h) {
                cont++;
                System.out.println("c1= " + c1 + ", c2= " + c2 + ", h= " + h);
           }
        }
    }
    return cont;
}
```

Es demana:

a) (0.5 punts) Indicar quina és la grandària o talla del problema, així com l'expressió que la representa.

b) (0.5 punts) Indicar si existeixen diferents instàncies significatives per al cost temporal de l'algorisme i identificar-les si és el cas.

c) (2 punts) Triar una unitat de mesura per al cost (passos de programa, instrucció crítica) i d'acord amb ella, obtenir una expressió el més precisa possible del cost temporal del programa, a nivell global o en les instàncies més significatives si és el cas.

d) (1 punt) Expressar el resultat anterior en notació asimptòtica.

Solució:

a) La talla n del problema és el paràmetre `h`, és a dir, la hipotenusa dels triangles buscats.

b) No hi ha diferents instàncies, el número de passades que fan els bucles només depèn de la talla.

c) En passos de programa:

$$T(n) = 1 + \sum_{i=4}^{n-1}(1 + \sum_{j=3}^{i-1} 1) = 1 + \sum_{i=4}^{n-1}(i-2) = 1 + \frac{(n-1)\cdot(n-4)}{2} \ \ p.p.$$

Si triem com instrucció crítica, de cost unitari, l'avaluació de la condició `c1 * c1 + c2 * c2 == h * h`, el que equival a despreciar termes d'ordre inferior:

$$T(n) = \sum_{i=4}^{n-1}\sum_{j=3}^{i-1} 1 = \sum_{i=4}^{n-1}(i-3) = \frac{(n-3)\cdot(n-4)}{2} \ \ i.c.$$

d) En notació asimptòtica: $T(n) \in \theta(n^2)$.

RecP1 - Curs 13/14: 3 punts

El següent mètode recursiu, `inversio(String)`, obté una `String` amb la inversió dels caràcters de la que rep com a argument. Per exemple, la inversió de la cadena `hola` és `aloh`.

```
public static String inversio(String s) {
    if (s.length() <= 1) { return s; }
    else { return inversio(s.substring(1)) + s.charAt(0); }
}
```

Es vol estudiar el seu cost temporal en les dues situacions següents:

1. Suposar que tant l'operació `substring(int)` com l'operació de concatenació (operador `+`) tenen un **cost constant** amb la llargària de la String `s`.
2. Suposar que l'operació `substring(int)` té un **cost lineal** amb la llargària de la String `s`, mentre que l'operació de concatenació (operador `+`) té un **cost constant** amb el nombre total de caràcters que es concatenen.

Per a cadascuna de les dues situacions **es demana**:

(a) (0.25 punts) Indicar quina és la mida o talla del problema, així com l'expressió que la representa.

(b) (0.5 punts) Indicar si existeixen diferents instàncies significatives per al cost temporal de l'algorisme, i identificar-les si és el cas.

(c) (1.5 punts) Escriure les equacions de recurrència del cost temporal en funció de la talla, resolent-les per substitució.

(d) (0.5 punts) Expressar el resultat anterior en notació asimptòtica.

(e) (0.25 punts) Quines de les dues situacions creus que és la més favorable des d'un punt de vista de cost temporal? Justifica la teva resposta.

Solució:

(a) Per a ambdues situacions, la talla del problema és el nombre de caràcters de la `String s`, que canviarà en cada crida recursiva. L'expressió que la defineix és `s.length()`, que anomenarem n.

(b) Per a ambdues situacions, no existeixen instàncies significatives perquè es tracta d'un recorregut sobre la `String s`.

(c) Per al cas de `substring` i operador de concatenació amb costos constants:

$$T(n) = \begin{cases} k' & si\ n \leq 1 \\ T(n-1) + k & si\ n > 1 \end{cases}$$

sent k i k' constants positives, en alguna unitat de temps. Resolent per substitució:

$$\begin{array}{rl} T(n) & = T(n-1) + k = \ T(n-2) + 2k = \ T(n-3) + 3k = \cdots = \\ & = T(n-i) + i \cdot k = \cdots = \\ & (cas\ base: \ n - i = 1, \ i = n - 1) \\ & = T(1) + (n-1)k = kn - k + k' \end{array}$$

Per al cas de `substring` amb cost lineal i operador de concatenació amb cost constant:

$$T(n) = \begin{cases} k' & si\ n \leq 1 \\ T(n-1) + kn & si\ n > 1 \end{cases}$$

sent k i k' constants positives, en alguna unitat de temps. Resolent per substitució i menyspreant termes d'ordre inferior:

$$\begin{array}{rl} T(n) & = T(n-1) + kn = \ T(n-2) + k(n-1) + kn = \ T(n-3) + k(n-2) + k(n-1) + kn = \cdots = \\ & = T(n-i) + \sum_{j=n-(i-1)}^{n} kj = \cdots = \\ & (cas\ base: \ n - i = 1, \ i = n - 1, \ n - (i-1) = 2) \\ & = T(1) + \sum_{j=2}^{n} kj = k' + k\Big(\frac{n(n+1)}{2} - 1\Big) \end{array}$$

(d) En notació asimptòtica:

Per al cas de `substring` i operador de concatenació amb costos constants: $T(n) \in \theta(n)$.

Per al cas de `substring` amb cost lineal: $T(n) \in \theta(n^2)$.

(e) A la vista del cost temporal asimptòtic és més favorable la situació on `substring` presenta cost constant ja que l'algorisme té un cost lineal amb la talla del problema.

RecP1 - Curs 13/14: 4 punts

El següent algorisme iteratiu retorna un enter corresponent a la suma màxima dels valors emmagatzemats en posicions consecutives d'un array donat `a`.

```
/** Precondició: a.length >= 1. */
public static int metode(int[] a) {
    int n = a.length, max = a[0];
    for (int i = 0; i <= n - 1; i++) {
        int suma = 0;
        for (int j = i; j <= n - 1; j++) {
            suma = suma + a[j];
            if (suma > max) { max = suma; }
        }
    }
    return max;
}
```

Es demana:

a) (0.5 punts) Indicar quina és la mida o talla del problema, així com l'expressió que la representa.

b) (0.5 punts) Indicar si existeixen diferents instàncies significatives per al cost temporal de l'algorisme, i identificar-les si és el cas.

c) (2 punts) Escollir una unitat de mesura per al cost (passos de programa, instrucció crítica) i d'acord amb ella, obtenir una expressió el més precisa possible del cost temporal del programa (per al cas millor i el cas pitjor si és el cas).

d) (1 punt) Expressar el resultat anterior en notació asimptòtica.

Solució:

a) La talla és la grandària de l'array, `a.length`, que anomenarem n.

b) No hi ha instàncies significatives, perquè és un problema de recorregut d'un array.

c) En passos de programa:

$$T(n) = 1 + \sum_{i=0}^{n-1}(1 + \sum_{j=i}^{n-1} 1) = 1 + \sum_{i=0}^{n-1}(1 + n - i) = 1 + n + n^2 - \frac{n(n-1)}{2} = \frac{1}{2}n^2 + \frac{3}{2}n + 1 \ p.p.$$

Si prenem com instrucció crítica: `suma = suma + a[j];` i considerant-la de cost unitari:

$$T(n) = \sum_{i=0}^{n-1}\sum_{j=i}^{n-1} 1 = \sum_{i=0}^{n-1}(n-i) = n\sum_{i=0}^{n-1} 1 - \sum_{i=0}^{n-1} i = n^2 - \frac{n(n-1)}{2} = n^2 - \frac{1}{2}(n^2 - n) = \frac{1}{2}n^2 + \frac{1}{2}n \quad i.c.$$

d) El cost és quadràtic amb la talla del problema, $T(n) \in \theta(n^2)$.

Curs 2012/13

P1 - Curs 12/13: 3 punts

Donada certa matriu `m` quadrada i un array `a`, d'enters, els dos amb la mateixa dimensió (això és, `m.length == a.length`), el següent mètode **iteratiu** torna la posició (número de fila) on es troba l'array `a` en `m`, cas de que es trobe, o torna -1 si no fora així:

```
/** Torna la posició en que l'array a es troba com a fila
 *  dins de la matriu quadrada m, o -1 si no es troba.
 *  Precondició: m es quadrada i m.length == a.length. */
public static int cercaPosFila(int[][] m, int[] a) {
    boolean trobat = false;
    int i = 0;
    while (i < m.length && !trobat) {
        trobat = true;
        for (int j = 0; j < m.length && trobat; j++) {
            trobat = (m[i][j] == a[j]);
        }
        if (!trobat) { i++; }
    }
    if (trobat) { return i; }
    else { return -1; }
}
```

Es demana l'estudi del seu cost temporal:

a) Indica quina és la grandària o talla del problema, així com l'expressió que la representa.

b) Identifica, en cas que n'hi hagués, les instàncies del problema que representen el cas millor i pitjor de l'algorisme.

c) Tria una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i d'acord amb ella, obté una expressió matemàtica, el més precisa possible, del cost temporal del programa, a nivell global o en les instàncies més significatives si les hi ha.

d) Expressa el resultat anterior utilitzant notació asimptòtica.

Solució:

a) La talla o grandària del problema és $n =$ `m.length`, és a dir, la dimensió de la matriu.

b) El mètode és un problema de cerca i, per tant, per a una mateixa talla sí que presenta instàncies distintes.

Cas millor: Per una banda, si les n components d'`a` apareixen en la primera fila de la matriu, el bucle exterior sols executa una passada, en la que es comprova en n passades que les `n` successives components d'`a[0..n - 1]` coincideixen amb les corresponents components de `m[0]`.

Per altra banda, es pot donar que el bucle intern siga lo més curt possible per a totes les files si el primer element de cada fila és diferent d'`a[0]`. En eixe cas el bucle exterior completaria les n passades (una per cada fila de la matriu), i totes elles amb el cost de fer una sola comprovació.

Cas pitjor: Ocorre quan per a totes les files es fan el màxim de comprovacions possibles, és a dir, les components d'`a[0..n - 2]` coincideixen amb les corresponents de la fila, però `a` no apareix complet en cap fila de `m` (excepte potser la darrera).

c)
- Si optem per triar com unitat de mesura el pas de programa, s'obté:
  En el cas millor, $T^m(n) = 1 + \sum_{i=0}^{n-1} 1 = n + 1$ *p.p.*
  En el cas pitjor, $T^p(n) = 1 + \sum_{i=0}^{n-1}(1 + \sum_{j=0}^{n-1} 1) = 1 + \sum_{i=0}^{n-1}(1 + n) = 1 + n + n^2$ *p.p.*

- Si optem per triar la instrucció crítica com unitat de mesura per a l'estimació del cost, es pot considerar com a tal la comparació: `(m[i][j] == a[j])`, de cost unitari.

  En el *cas millor* s'executarà n vegades i la funció de cost temporal en aquest cas serà: $T^m(n) = n$ *i.c.*

  En el *cas pitjor* es repetirà el número màxim de vegades possible i la funció de cost serà: $T^p(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1 = n \cdot n = n^2$ *i.c.*

d) En notació asimptòtica: $T^m(n) \in \Theta(n)$, $T^p(n) \in \Theta(n^2)$. Per tant, $T(n) \in \Omega(n)$, $T(n) \in O(n^2)$.

P1 - Curs 12/13: 2 punts

El següent mètode comprova si cert subarray `a[i..f]` d'`int` està o no ordenat ascendentment subdividint-lo prèviament i comprobant que cadascuna de les seues dues parts ho estiga, així com la relació entre elles:

```
/** Torna cert sii a[i..f] està ordenat ascendentment.
 *  Precondició: i <= f. */
public static boolean esOrdenat(int[] a, int i, int f) {
    if (i == f) { return true; }
    else {
        int m = (i + f) / 2;
        return esOrdenat(a, i, m) && esOrdenat(a, m + 1, f) && a[m] <= a[m + 1];
   }
}
```

Es demana l'estudi del seu cost temporal:

a) Indica quina és la talla del problema i quina expressió la defineix.

b) Determina si existeixen instàncies significatives. Si n'hi ha, identifica les que representen els casos millor i pitjor de l'algorisme.

c) Escriu l'equació de recurrència del cost temporal en funció de la talla per a cada un dels casos si n'hi ha diversos, o una única equació si només hi hagués un cas. Cal resoldre-la per substitució.

d) Expressa el resultat anterior fent servir notació asimptòtica.

Solució:

a) La talla, n, és l'amplària del subarray comprès entre les posicions `i` i `f`. Això és, $n =$ `f - i + 1`.

b) El cas millor és aquell en el que sempre s'executa només la primera crida, això és, quan el primer element està fora d'ordre (quan inicialment `a[i]` > `a[i + 1]`). El cas pitjor és aquell en el qual s'han d'executar totes les crides recursives. Aquesta situació es dóna, per exemple, quan inicialmente el subarray `a[i..f]` està ordenat.

c) Plantegem l'equació de recurrència per a cadascuna de les dues instàncies significatives, sent k i k' constants positives, en alguna unitat de temps.

Cas millor:

$$T^m(n) = \begin{cases} k & n = 1 \\ T^m(\frac{n}{2}) + k' & n > 1 \end{cases}$$

$$\begin{array}{lll} 1) & T^m(n) = & T^m(\frac{n}{2}) + k' \\ 2) & & T^m(\frac{n}{2^2}) + 2k' \\ 3) & & T^m(\frac{n}{2^3}) + 3k' \\ .. & & ... \\ i) & & T^m(\frac{n}{2^i}) + ik' \end{array}$$

$\frac{n}{2^i} = 1; \quad n = 2^i; \quad log_2 n = i;$

$T^m(n) = k + k' log_2 n$

Cas pitjor:

$$T^p(n) = \begin{cases} k & n = 1 \\ 2T^p(\frac{n}{2}) + k' & n > 1 \end{cases}$$

$$\begin{array}{lll} 1) & T^p(n) = & 2T^p(\frac{n}{2}) + k' \\ 2) & & 2^2 T^p(\frac{n}{2^2}) + k'(1+2) \\ 3) & & 2^3 T^p(\frac{n}{2^3}) + k'(1+2+2^2) \\ .. & & ... \\ i) & & 2^i T^p(\frac{n}{2^i}) + k' \sum_{j=0}^{i-1} 2^j \end{array}$$

$\frac{n}{2^i} = 1; \quad n = 2^i;$

$T^p(n) = 2^i k + (2^i - 1)k' = nk + (n-1)k' = n(k + k') - k'$

d) En notació asimptòtica: $T^m(n) \in \Theta(log\ n)$, $T^p(n) \in \Theta(n)$. Per tant, $T(n) \in \Omega(log\ n)$, $T(n) \in O(n)$

P1 - Curs 12/13: 1 punt

És possible modificar una única instrucció de l'algorisme anterior per a que el seu cost temporal siga $\Omega(1)$. Quina instrucció s'hauria de modificar i de quina forma?

Solució: Cal modificar l'instrucció amb les crides recursives per a que s'execute primer la comprovació, això és, caldria substituir-la per:

```
return a[m] <= a[m + 1] && esOrdenat(a, i, m) && esOrdenat(a, m + 1, f);
```

RecP1 - Curs 12/13: 3.0 punts

Donada certa matriu `m` quadrada de valors reals, el següent mètode determina si aquesta és triangular inferior (és a dir, si tots els elements superiors a la diagonal principal són iguals a 0):

```
/** Determina si una matriu quadrada és triangular inferior, això és:
  * si tots els elements superiors a la diagonal principal valen 0. * /
public static boolean esInferior(double[][] m) {
    boolean esInf = true;
    for (int i = 0; i < m.length && esInf; i++) {
        for (int j = i + 1; j < m.length && esInf; j++) {
            esInf = m[i][j] == 0;
        }
    }
    return esInf;
}
```

Es demana: estudiar el cost temporal del mètode, per el que has de:

a) Indicar quina és la mida o talla del problema, així com l'expressió que la representa.

b) Identificar, cas que n'hi hagués, les instàncies del problema que representen el cas millor i pitjor de l'algorisme.

c) Triar una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i acord amb ella, obtenir una expressió matemàtica, el més precisa possible, del cost temporal del programa, a nivell global o en les instàncies més significatives si n'hi ha.

d) Expressar el resultat anterior utilitzant notació asimptòtica.

Solució:

a) La talla del problema és $n =$ `m.length`, és a dir, la dimensió de la matriu.

b) El mètode és un problema de cerca i, per tant, per a una mateixa talla sí que presenta instàncies diferents.

Cas millor: Quan `m[0][1] != 0` amb el que ambdós bucles finalitzaran en acabar la seua primera iteració.

Cas pitjor: Quan la matriu és triangular inferior i tots els elements superiors a la diagonal principal valen 0. En aquest cas, s'efectuen totes les iteracions possibles.

c)
- Si optem per escollir com a unitat de mesura el pas de programa, s'obté:
  - *Cas millor:* $T^m(n) = 1$ *p.p.*
  - *Cas pitjor:* $T^p(n) = 1 + \sum_{i=0}^{n-1}(1 + \sum_{j=i+1}^{n-1} 1) = 1 + \sum_{i=0}^{n-1}(n-i) = n^2/2 + n/2 + 1$ *p.p.*

- Si optem per escollir la instrucció crítica com a unitat de mesura per a l'estimació del cost, es pot considerar com a tal `esInf = m[i][j] == 0;`, de cost unitari.
  - *Cas millor:* $T^m(n) = 1$ *i.c.*
  - *Cas pitjor:* $T^p(n) = \sum_{i=0}^{n-1}(\sum_{j=i+1}^{n-1} 1) = \sum_{i=0}^{n-1}(n-i-1) = n^2/2 - n/2$ *i.c.*

d) En notació asimptòtica: $T^m(n) \in \Theta(1)$ i $T^p(n) \in \Theta(n^2)$. Per tant, $T(n) \in \Omega(1)$ i $T(n) \in O(n^2)$.

RecP1 - Curs 12/13: 2.5 punts

El següent mètode recursiu comprova si dos `String` que tenen la mateixa longitud són simètriques. Dues `String` són simètriques quan el primer element de la primera és igual a l'últim de la segona i així successivament.

Per exemple, les `String`: `"HOLA"` i `"ALOH"` són simètriques, mentre que `"HOLA"` i `" ALHA"` no ho són.

```
/** Determina si a i b són o no simètriques
 *  Precondició: a.length() == b.length(). */
public static boolean simetriques(String a, String b) {
    if (a.length() == 0) { return true; }
    else {
        int ult = b.length() - 1;
        return a.charAt(0) == b.charAt(ult)
            && simetriques(a.substring(1), b.substring(0, ult));
    }
}
```

Es demana: estudiar el cost temporal del mètode anterior, sabent que **totes les operacions de la classe** `String` que s'apliquen a alguna `String` en l'algorisme **tenen un cost constant** (això és, el seu cost no depèn de la llargària de la `String` a la qual s'aplique) per a això:

a) Indica quina és la talla del problema i quina expressió la defineix.

b) Determina si hi ha instàncies significatives. Si n'hi ha, identifica les que representen els casos millor i pitjor de l'algorisme.

c) Escriu l'equació de recurrència del cost temporal en funció de la talla per a cada un dels casos si n'hi ha diversos, o una única equació si només hi hagués un cas. Resol-la per substitució.

d) Expressa el resultat anterior usant notació asimptòtica.

Solució:

a) La talla és la longitud n de cada `String` (`a.length()`).

b) Sí que hi ha instàncies significatives. El *cas millor* es dóna quan el primer i últim caràcter d'ambdues `String` no concorden. El *cas pitjor* es dóna quan ambdues `String` són simètriques.

c) Per obtenir el cost del mètode, expressat en passos de programa (p.p.), estudiem cadascuna de les dues instàncies significatives:

- Cas millor: $T^m(n) = 1$ *p.p.*
- Cas pitjor:

$$T^p(n) = \begin{cases} 1 & si\ n = 0 \\ 1 + T^p(n-1) & si\ n > 0 \end{cases}$$

Resolent per substitució: $T^p(n) = T^p(n-1) + 1 = T^p(n-2) + 2 = \ldots = T^p(n-i) + i = \ldots = T^p(0) + n = 1 + n$ *p.p.*

d) En notació asimptòtica: $T^m(n) \in \Theta(1)$ i $T^p(n) \in \Theta(n)$. Per tant, $T(n) \in \Omega(1)$ i $T(n) \in O(n)$.

Curs 2011/12

P1 - Curs 11/12: 2 punts

Donat el següent mètode:

```
/** Precondició: n >= 0. */
public static void binari(int n) {
    if (n > 0) { binari(n / 2); }
    System.out.print(n % 2);
}
```

Es demana:

a) Indica quina és la talla del problema i quina expressió la defineix.

b) Determina si existeixen instàncies significatives. Si n'hi ha, identifica les que representen els casos millor i pitjor de l'algorisme.

c) Escriu l'equació de recurrència del cost temporal en funció de la talla per a cada un dels casos si n'hi ha diversos, o una única equació si només hi hagués un cas. Cal resoldre-la per substitució.

d) Expressa el resultat anterior fent servir notació asimptòtica.

Solució:

a) La talla del problema és el valor de l'argument del mètode, açò és, `n`.

b) No hi ha instàncies significatives.

c) Plantegem l'equació de recurrència: $T(n) = \begin{cases} T(n/2) + k & \text{si } n > 0 \\ k' & \text{si } n = 0 \end{cases}$ sent k i k' constants positives, en alguna unitat de temps.

Resolent per substitució: $T(n) = T(n/2) + k = T(n/2^2) + 2k = \ldots = T(n/2^i) + ik$. En arribar al cas base $i \approx log_2 n$ i es compleix que $T(n/2^i) \to T(0) = k'$. Amb el que $T(n) \approx k' + klog_2 n$.

d) En notació asimptòtica: $T(n) \in \Theta(\log n)$, és a dir, el cost temporal és logarítmic amb la talla del problema.

P1 - Curs 11/12: 3 punts

Siga una matriu quadrada d'enters, `m`, amb tots els seus valors no negatius. Per tal de comprovar si els elements de la seua diagonal principal sumen més que cert valor `val`, no negatiu, es consideren els dos mètodes següents:

```
public class Problema4 {
    /** Per a tot i, j: 0 <= i < m.length, 0 <= j < m.length, m[i][j] >= 0 i val >= 0 */
    public static boolean metode1(int[][] m, int val) {
        int s = 0;
        for (int i = 0; i < m.length; i++) { s += m[i][i]; }
        return s > val;
    }

    /** Per a tot i, j: 0 <= i < m.length, 0 <= j < m.length, m[i][j] >= 0 i val >= 0 */
    public static boolean metode2(int[][] m, int val) {
        int s = 0;
        for (int i = 0; i < m.length && s <= val; i++) {
            for (int j = 0; j <= i && s <= val; j++) {
                if (i == j) { s += m[i][j]; }
            }
        }
        return s > val;
    }
}
```

Es demana:

a) **Per a cada mètode:**
   1. Indica quina és la grandària o talla del problema, així com l'expressió que la representa.
   2. Identifica, en cas que n'hi hagués, les instàncies del problema que representen el cas millor i pitjor de l'algorisme.
   3. Tria una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i d'acord amb ella, obté una expressió matemàtica, el més precisa possible, del cost temporal del programa, a nivell global o en les instàncies més significatives si les hi ha.
   4. Expressa el resultat anterior utilitzant notació asimptòtica.

b) Descriu breument les diferències entre ambdós mètodes.

c) Com modificaries el primer per tal de millorar-lo?

Solució:

a) Per a cada mètode:

   1. En ambdós mètodes, la talla o grandària del problema és el número de files de la matriu, `m.length`, que al mateix temps coincideix amb el de columnes. D'ara endavant, anomenarem a aquest nombre n. És a dir, $n =$ `m.length`.
   2. El primer mètode és un problema de recorregut, per tant, per a una mateixa talla del problema no hi ha instàncies significatives. El segon mètode és un problema de cerca i, per tant, per a una mateixa talla sí que presenta instàncies distintes. El *cas millor* es dóna quan el primer element de la matriu és major que `val`, és a dir, `m[0][0] > val`. El *cas pitjor* ocorre quan la suma de tots els elements de la diagonal principal de la matriu és menor o igual que `val`.
   3. - Si optem per triar com unitat de mesura el pas de programa, s'obté:
      Per al primer mètode, com a funció de cost temporal: $T(n) = 1 + \sum_{i=0}^{n-1} 1 = n + 1$ *p.p.*
      Per al segon mètode, en el seu *cas millor*, quan tan sols s'avalua una única vegada la guarda del bucle més intern, podem dir que $T^m(n) = 1$ *p.p.*, mentre que per al *cas pitjor* es tindria: $T^p(n) = 1 + \sum_{i=0}^{n-1}(1 + \sum_{j=0}^{i} 1) = 1 + \sum_{i=0}^{n-1}(i+2) = (n^2+3n)/2 + 1$ *p.p.*
      - Si optem per triar la instrucció crítica com unitat de mesura per a l'estimació del cost, es tindrà que:
      En el primer mètode, si es selecciona l'assignació `s += m[i][i]`, de cost unitari, obtenim la següent funció de cost temporal: $T(n) = \sum_{i=0}^{n-1} 1 = n$ *i.c.*
      En el segon mètode, considerem com instrucció crítica la comparació: `(i == j)`, de cost unitari. En el *cas millor* només s'executarà una vegada i la funció de cost temporal en aquest cas serà $T^m(n) = 1$ *i.c.* En el *cas pitjor* es repetirà el número màxim de vegades possible i la funció de cost serà $T^p(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i} 1 = \sum_{i=0}^{n-1}(i+1) = (n^2+n)/2$ *i.c.*
   4. Per al primer mètode, en notació asimptòtica, $T(n) \in \Theta(n)$, és a dir, el cost temporal és lineal amb la talla del problema.
      Per al segon mètode, en notació asimptòtica, $T^m(n) \in \Theta(1)$ i $T^p(n) \in \Theta(n^2)$. Per tant, $T(n) \in \Omega(1)$ i $T(n) \in O(n^2)$, és a dir, el cost temporal està fitat inferiorment per una funció constant i superiorment per una funció quadràtica amb la talla del problema.

b) Per tal de determinar si els elements de la diagonal de la matriu quadrada `m` sumen més que `val`, el primer mètode únicament visita aquests elements mentre que el segon mètode visita també tots els elements per baix de la diagonal. El primer mètode, al tractar-se d'un un recorregut, realitza la suma de tots els elements de la diagonal. No obstant, el segon mètode, al tractar-se d'una cerca, quan la suma dels elements de la diagonal visitats en un moment donat és major que `val`, ja no segueix sumant.

c) El primer mètode es pot millorar si es resol com una cerca, només canviant la guarda del bucle per `i < m.length && s <= val`.

P1 - Curs 11/12: 1 punt

La taula següent mostra els temps d'execució, en mil·lisegons, de cert algorisme per als valors de la talla del problema que es mostren en la primera columna.

```
#    Talla    Temps (ms)
#---------------------------------------
      1000       49.78
      2000      202.33
      3000      454.42
      4000      804.03
      5000     1270.28
      6000     1841.47
      7000     2506.30
      8000     3253.62
      9000     4141.05
     10000     5277.99
```

a) Des d'un punt de vista asimptòtic, quina creus que és la funció de cost temporal que millor aproxima els valors de la columna `Temps`?

b) De forma aproximada, quant de temps creus que tardaria l'algorisme anterior en executar-se per a una talla de `20000` elements?

Solució:

a) La funció de cost temporal que més s'aproxima als valors de la taula és una funció quadràtica amb la talla del problema. Si anomenem n a aquesta talla, $T(n) \in \Theta(n^2)$. Es pot comprovar que quan la talla es duplica (de n a $2n$), el temps es multiplica per 4 (de n^2 a 2^2n^2). Per exemple, per a $n = 2000$ el temps és 202.33 ms i per a $n = 4000$ el temps és 804.03 ms, aproximadament 4 vegades el de $n = 2000$.

b) Per a una talla de `20000` elements, el temps d'execució seria aproximadament $4 \times 5277.99 \approx 20000$ mil·lisegons.

RecP1 - Curs 11/12: 2.5 punts

Donat el següent mètode:

```
/** Precondició: n >= 0, 1 <= x <= 9. */
public static boolean cercarX(int n, int x){
    if (n > 0) {
        if (n % 10 == x) { return true; }
        else { return cercarX(n / 10, x); }
    }
    else { return false; }
}
```

Es demana:

a) Indicar quina és la talla del problema i quina expressió la defineix.

b) Determinar si existeixen instàncies significatives. Si n'hi ha, identificar les que representen els casos millor i pitjor de l'algorisme.

c) Escriure l'equació de recurrència del cost temporal en funció de la talla per a cada un dels casos si n'hi ha diversos, o una única equació si només hi hagués un cas. Cal resoldre-la per substitució.

d) Expressar el resultat anterior fent servir notació asimptòtica.

Solució:

a) La talla del problema és el valor de l'argument del mètode, açò és, `n`.

b) Sí que hi ha instàncies significatives, és una cerca. En el millor cas, `x` és la xifra de les unitats de `n`, i en el cas pitjor cap xifra de `n` és `x`.

c) Plantegem l'equació de recurrència per a cadascuna de les dues instàncies significatives, en passos de programa.

En el cas millor: $T^m(n) = 1$ *p.p.*

En el cas pitjor:

$$T^p(n) = \begin{cases} T^p(n/10) + 1 & \text{si } n > 0 \\ 1 & \text{si } n = 0 \end{cases}$$

Resolent-la per substitució:

$T^p(n) = T^p(n/10) + 1 = T^p(n/10^2) + 2 = \ldots = T^p(n/10^i) + i$. Quan $i = 1 + \lfloor log_{10} n \rfloor$ s'arriba al cas base en el que $T^p(0) = 1$.

Amb el que $T^p(n) = 2 + \lfloor log_{10} n \rfloor$ *p.p.*

d) En notació asimptòtica: $T^m(n) \in \Theta(1)$ i $T^p(n) \in \Theta(\log n)$, és a dir, les fites per al cost són $T(n) \in \Omega(1)$ i $T(n) \in O(\log n)$.

RecP1 - Curs 11/12: 3.5 punts

Siga `a` un array $\{a_0, a_1, a_2, \ldots, a_{n-1}\}$ de `double`, que representa els coeficients d'un polinomi $a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$. Per tal de calcular el valor del polinomi per a un x donat, es proposen els següents mètodes:

Mètode 1:

```
/** Precondició: a.length >= 1. */
public static double polinomi1(double[] a, double x) {
    double result = a[0];
    for (int i = 1; i < a.length; i++) {
        double pot = 1;
        for (int k = 1; k <= i; k++) { pot = pot * x; }
        result += a[i] * pot;
    }
    return result;
}
```

Mètode 2:

```
/** Precondició: a.length >= 1. */
public static double polinomi2(double[] a, double x) {
    double result = a[a.length - 1];
    for (int i = a.length - 2; i >= 0; i--) {
        result = result * x + a[i];
    }
    return result;
}
```

Es demana:

a) **Per a cada mètode:**

1. Indicar quina és la grandària o talla del problema, així com l'expressió que la representa.
2. Identificar, en cas que n'hi hagués, les instàncies del problema que representen el cas millor i pitjor de l'algorisme.
3. Triar una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i d'acord amb ella, obtindre una expressió matemàtica, el més precisa possible, del cost temporal del programa, a nivell global o en les instàncies més significatives si les hi ha.
4. Expressar el resultat anterior utilitzant notació asimptòtica.

b) Indicar quin dels dos mètodes és més eficient i perquè.

Solució:

a) **Mètode 1.**

1) La talla $n =$ `a.length`, és a dir, el nombre d'elements d'`a`.
2) No hi ha diferents instàncies, per lo que sols s'estudiarà una funció $T(n)$, vàlida per a qualsevol entrada de talla n.
3) En passos de programa, $T(n) = 1 + \sum_{i=1}^{n-1}(1+i) = 1 + (2 + 3 + \ldots + n) = \frac{(n+1)\cdot n}{2}$ *p.p.*
4) En resum, $T(n) \in \Theta(n^2)$.

Mètode 2.

1) Ídem que per al mètode 1.
2) Ídem que per al mètode 1.
3) En passos de programa, $T(n) = 1 + \sum_{i=1}^{n-1} 1 = n$ *p.p*
4) En resum, $T(n) \in \Theta(n)$.

b) El primer mètode és quadràtic amb la talla del problema i el segon és linial, per lo que es pot concloure que el segon és més eficient.

De l'anàlisi realitzat s'observa que el primer mètode recalcula completament la potència de `x` elevat a `i` en cada passada del bucle, operació que va resultant més costosa a mesura que augmenta `i`. El segon mètode ho evita fent que les potències de `x` acumulades en `result` vagen augmentant el seu grau en 1 en cada passada del bucle.

Curs 2010/11

P1 - Curs 10/11: 3 punts

El següent mètode **iteratiu** calcula el producte d'un vector fila `x` per una matriu quadrada `a`, obtenint com a resultat un altre vector fila. Per tractar-se d'una matriu quadrada tant el vector `x` com el vector resultat tindran la mateixa dimensió, és a dir, el nombre de files de la matriu.

```
/** Precondició: x.length = a.length i a és una matriu quadrada. */
public static double[] vectorPerMatriu(double[] x, double[][] a) {
    double[] r = new double[a.length];
    for (int i = 0; i < r.length; i++) {
        r[i] = 0.0;
        for (int j = 0; j < x.length; j++) {
            r[i] += x[j] * a[j][i];
        }
    }
    return r;
}
```

Per exemple, per `x` $= (2, 1, 3)$, un vector fila de dimensió 1×3, i `a` $= \begin{pmatrix} 4 & 3 & 2 \\ 2 & 5 & 1 \\ 1 & 0 & 3 \end{pmatrix}$, una matriu de dimensió 3×3, el producte `x` $\times$ `a` és `r` $= (13, 11, 14)$, un vector fila de dimensió 1×3.

Es demana:

a) Indicar quina és la grandària o talla del problema, així com l'expressió que la representa.

b) Identificar, en cas que n'hi hagués, les instàncies del problema que representen el cas millor i pitjor de l'algorisme.

c) Triar una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i d'acord amb ella, obtenir una expressió matemàtica, el més precisa possible, del cost temporal del programa, a nivell global o en les instàncies més significatives si les hi ha.

d) Expressar el resultat anterior utilitzant notació asimptòtica.

Solució:

a) La talla o grandària del problema és el nombre de files de la matriu `a.length`, que al coincideix amb el de columnes de la propia matriu i amb el nombre de components del vector `x`.D'ara endavant, anomenarem a aquest nombre n. És a dir, $n =$ `a.length`.

b) Es tracta d'un problema de recorregut, per tant, per a una mateixa talla del problema no hi ha instàncies significatives.

c) Si es pren com a unitat de mesura el pas de programa, es pot expressar el cost de l'algorisme de la manera següent: $T(n) = 1 + \sum_{i=0}^{n-1}(1 + \sum_{j=0}^{n-1} 1) = 1 + \sum_{i=0}^{n-1}(1 + n) = 1 + n + n^2$ *p.p.*

Si es tria com a unitat de mesura per a l'estimació del cost una instrucció crítica, en concret, seleccionant l'assignació més interna `r[i] += x[j] * a[j][i]`, de cost unitari, obtenim la següent funció de cost temporal: $T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1 = n^2$ *i.c.*

d) En notació asimptòtica: $T(n) \in \Theta(n^2)$.

P1 - Curs 10/11: 3.5 punts

Per obtenir la posició de l'últim element senar d'un array `v` d'enters, es proposa el següent mètode **iteratiu**:

```
public static int posUltimSenar(int[] v) {
    int i = v.length - 1;
    while (i >= 0 && v[i] % 2 == 0) { i--; }
    return i;
}
```

Es demana:

a) Indicar quina és la grandària o talla del problema, així com l'expressió que la representa.

b) Identificar, en cas que n'hi hagués, les instàncies del problema que representen el cas millor i pitjor de l'algorisme.

c) Triar una unitat de mesura per a l'estimació del cost (passos de programa, instrucció crítica) i d'acord amb ella, obtenir una expressió matemàtica, el més precisa possible, del cost temporal del programa, a nivell global o en les instàncies més significatives si les hi ha.

d) Expressar el resultat anterior utilitzant notació asimptòtica.

Solució:

a) La grandària o talla del problema és el nombre d'elements de l'array `v` i l'expressió que la representa és `v.length`. D'ara endavant, anomenarem a aquest nombre n. És a dir, $n =$ `v.length`.

b) Es tracta d'un problema de cerca (seqüencial iterativa descendent) i, per tant, per a una mateixa talla si que presenta instàncies diferents. El *cas millor* es dóna quan l'últim element de l'array és senar (`v[v.length - 1] % 2 == 1`). El *cas pitjor* passa quan tots els elements de l'array són parells ($\forall$`i`, `0`$\leq$`i`$<$`v.length`, `v[i] % 2 == 0`).

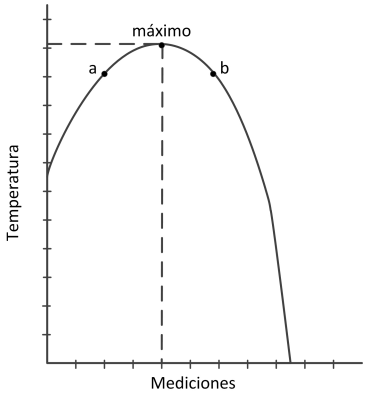
c) Optem per escollir la instrucció crítica com a unitat de mesura per a l'estimació del cost. L'única instrucció que podem considerar com a crítica és la guarda del bucle: `i >= 0 && v[i] % 2 == 0` (s'executa sempre una vegada més que qualsevol de les de dins del bucle). En el *cas millor* només s'executarà una vegada i la funció de cost temporal en aquest cas serà $T^m(n) = 1$ *i.c.* En el *cas pitjor* es repetirà tantes vegades com elements tinga la matriu més una (quan es fa falsa) i la funció de cost serà $T^p(n) = \sum_{i=0}^{n} 1 = n + 1$ *i.c.*

d) En notació asimptòtica: $T^m(n) \in \Theta(1)$ i $T^p(n) \in \Theta(n)$. Per tant, $T(n) \in \Omega(1)$ i $T(n) \in O(n)$, és a dir, el cost temporal està fitat inferiorment per una funció constant i superiorment per una funció lineal amb la talla del problema.

P1 - Curs 10/11: 3.5 punts

Es disposa d'una sèrie de mesures de temperatura que formen part d'una corba parabòlica (positiva) en un array `v` de reals. El següent mètode **recursiu** obté el màxim valor de la corba representada per l'array `v`.

Per a aquest tipus de corbes, donat un valor se sap que si l'anterior i el posterior són menors que ell, ens trobem en el punt màxim mesurat. Si per contra l'anterior és més gran i el posterior és menor, estem a la dreta del màxim, i si l'anterior és menor i el posterior gran, estem a l'esquerra del màxim. Un altra situació és impossible en aquest tipus de corba. A la corba de la figura, es pot observar que els punts `a` i `b` (anterior i posterior al punt màxim, respectivament) són menors que el punt `máximo`.



```
public static double maximaTemp(double[] v, int ini, int fi) {
    if (ini == fi) { return v[ini]; }
    else if (ini == fi - 1) { return Math.max(v[ini], v[fi]); }
    else {
        int meitat = (fi + ini) / 2;
        if (v[meitat] > v[meitat - 1] && v[meitat] > v[meitat + 1]) {
            return v[meitat];
        }
        else if (v[meitat] < v[meitat - 1] && v[meitat] > v[meitat + 1]) {
            return maximaTemp(v, ini, meitat);
        }
        else { return maximaTemp(v, meitat, fi); }
    }
}
```

La crida inicial serà: `double maxim = maximaTemp(v, 0, v.length - 1);`

Es demana:

a) Indicar quina és la talla del problema i quina expressió la defineix.

b) Determinar si hi ha instàncies significatives. Si n'hi ha, identificar les que representen els casos millor i pitjor de l'algorisme.

c) Escriure l'equació de recurrència del cost temporal en funció de la talla per a cada un dels casos si n'hi ha diversos, o una única equació si només hi hagués un cas. Resoldre per substitució.

d) Expressar el resultat anterior fent servir notació asimptòtica.

Solució:

a) La talla del problema és el nombre d'elements de l'array `v` entre les posicions `ini` i `fi`. És a dir, l'expressió de la talla és $n =$ `fi - ini + 1`.

b) Es tracta d'un problema de cerca (binària), de manera que per a una mateixa talla si es distingeixen instàncies. El *cas millor* es dóna quan el punt màxim es troba en `v[(fin + ini) / 2]` sent `ini = 0` i `fi = v.length - 1`. El *cas pitjor* es dóna quan per tal de trobar el punt màxim es fan el major nombre de crides possible, és a dir, quan la talla del problema es redueix fins aplegar a la talla del cas base.

c) En el *cas millor*, el cost és constant, $T^m(n) = c$. Per obtenir el cost en el *cas pitjor*, plantegem l'equació de recurrència: $T^p(n) = \begin{cases} T^p(n/2) + k & \text{si } n > 2 \\ k' & \text{si } n = 1 \text{ o } n = 2 \end{cases}$ sent k i k' constants positives, en alguna unitat de temps.

Resolent per substitució: $T^p(n) = T^p(n/2) + k = T^p(n/2^2) + 2k = \ldots = T^p(n/2^i) + ik$. En arribar al cas base, per talla 1, $n/2^i = 1 \rightarrow i = log_2 n$; i per talla 2, $n/2^i = 2 \rightarrow n = 2^i * 2 \rightarrow i = log_2 n + log_2 2 = log_2 n + 1$. Amb el que $T^p(n) \approx k' + k log_2 n$.

d) En notació asimptòtica: $T^m(n) \in \Theta(1)$ i $T^p(n) \in \Theta(log_2 n)$. Per tant, $T(n) \in \Omega(1)$ i $T(n) \in O(log_2 n)$, és a dir, el cost temporal està fitat inferiorment per una funció constant i superiorment per una funció logarítmica amb la talla del problema.

RecP1 - Curs 10/11: 5 punts

Donat el següent mètode **recursiu** on la seua invocació inicial deu ser: `palindrom(s, 0, s.length() - 1)`:

```
public static boolean palindrom(String s, int ini, int fi) {
    if (ini >= fi) { return true; }
    if (s.charAt(ini) != s.chartAt(fi)) { return false; }
    return palindrom(s, ini + 1, fi - 1);
}
```

Es demana:

a) Indicar quin és el tamany o talla del problema, així com l'expressió que el representa.
b) Identificar, en cas de que les hi haja, les instàncies del problema que representen els casos millor i pitjor de l'algorisme.
c) Triar una unitat de mesura per estimar el cost (pas de programa o instrucció crítica), i d'acord amb la mesura escollida obtenir l'expressió matemàtica, el més precisa possible, del cost temporal del programa. A nivell general si no existeixen instàncies significatives, o per als casos millor i pitjor en cas de que sí en existeixen.
d) Expressar el resultat anterior fet ús de la notació asimptòtica.

Solució:

a) La grandària o talla del problema és el número de caràcters de la cadena `s` i l'expressió que la representa és `fi - ini + 1` = `s.length()`. D'ara endavant, anomenarem a aquest número n. És a dir, $n =$ `fi - ini + 1` = `s.length()`.

b) Per a una mateixa talla sí que presenta instàncies distintes. El *cas millor* es dóna quan el primer caràcter de la cadena és diferent a l'últim caràcter de la cadena. El *cas pitjor* ocorre quan la cadena és un palíndrom (capicua per a les lletres).

c) Resolem el cost per recurrència:

En el *cas millor* només s'executarà una vegada i la funció de cost temporal en aquest cas serà $T^m(n) = 1$.

En el *cas pitjor* tindrem la següent funció $T^p(n) = \begin{cases} T^p(n-2) + k & \text{si } n > 1 \\ k' & \text{si } n = 0 \text{ o } n = 1 \end{cases}$ sent k i k' constants positives, en alguna unitat de temps.

Resolent per substitució: $T^p(n) = T^p(n-2) + k = T^p(n-4) + 2k = \ldots = T^p(n - 2 \cdot ik) + i$. S'arriba al cas base, talla 0 o 1, per a $i = n/2$. Amb el que $T^p(n) = k' + \frac{n}{2}k$.

d) En notació asimptòtica: $T^m(n) \in \Theta(1)$ i $T^p(n) \in \Theta(n)$. Per tant, $T(n) \in \Omega(1)$ i $T(n) \in O(n)$, és a dir, el cost temporal està fitat inferiorment per una funció constant i superiorment per una funció lineal amb la talla del problema.