Exàmens de PRG - Problemes del Tema 1
Recursió

Curs 2018/19

P1 - Curs 18/19: 4 punts

Donats un array d'enters `a` i un enter `k`, `k` $\geq$ `a.length`, es diu que `a` és *un capicua de clau* `k` si la diferència en valor absolut entre les components primera i última és menor que `k`, menor que `k` $-$ 1 entre les components segona i penúltima, menor que `k` $-$ 2 entre la tercera i l'antepenúltima, i així successivament. S'entén que l'array buit o amb una sola component són capicues, per a qualsevol clau. Es vol un mètode recursiu que comprove si `a` és un capicua de clau `k`. Per exemple,

- per a l'array $\{20, 15, 14, 32, 10, 7, 22\}$ i la clau 10, el mètode ha de retornar `true` ja que:
  - $|20 - 22| < 10$,
  - $|15 - 7| < 9$,
  - $|14 - 10| < 8$ i
  - trivialment, $|32 - 32| < 7$.
- per a l'array $\{20, 15, 14, 10, 7, 22\}$ i la clau 10, el mètode també ha de retornar `true` ja que:
  - $|20 - 22| < 10$,
  - $|15 - 7| < 9$,
  - $|14 - 10| < 8$.
- per als arrays anteriors i la clau 9 el mètode ha de retornar `false`, atès que:
  - $|20 - 22| < 9$, però
  - $|15 - 7|$ no és menor que 8.

Es demana:

a) (0.75 punts) Perfil del mètode recursiu, amb els paràmetres adequats per a resoldre recursivament el problema, i precondició relativa a aquests paràmetres.

b) (1.25 punts) Cas base i cas general de la recursió.

c) (1.50 punts) Implementació en Java del mètode recursiu.

d) (0.5 punts) Crida inicial al mètode per a comprovar si un cert array `data` és o no capicua de clau `k`.

Solució:

a) Una possible solució consisteix a definir un mètode amb el següent perfil:

```
/** Precondició: k >= a.length, 0 <= ini, fin < a.length */
public static boolean palindromeKey(int[] a, int ini, int fin, int k)
```

tal que comprova si l'array `a[ini..fin]` és capicua de clau `k`, sent `k` $\geq$ `a.length`, 0 $\leq$ `ini` i `fin` $<$ `a.length`.

b)
- Cas base, `ini >= fin`: Subarray de 0 o 1 elements. És capicua de clau `k` i retorna `true`.
- Cas general, `ini < fin`: Subarray de 2 o més elements. Si la diferència en valor absolut entre `a[ini]` i `a[fin]` no és menor que `k`, l'array no és capicua de clau `k` i retorna `false`. En cas contrari (`Math.abs(a[ini] - a[fin]) < k`), el problema es redueix a comprovar si el subarray `a[ini + 1..fin - 1]` és capicua de clau `k - 1`.

c)
```
/** Precondició: k >= a.length, 0 <= ini, fin < a.length */
public static boolean palindromeKey(int[] a, int ini, int fin, int k) {
   if (ini < fin) {
       int diff = Math.abs(a[ini] - a[fin]);
       return diff < k && palindromeKey(a, ini + 1, fin - 1, k - 1);
   }
   else { return true; }
}
```

d) La crida inicial per a comprovar si l'array `data` és capicua de clau `k` seria, sempre que `k` $\geq$ `data.length`: `palindromeKey(data, 0, data.length - 1, k)`.

RecP1 - Curs 18/19: 4 punts

Donat un enter `n` > 0, escriu un mètode recursiu que mostre per l'eixida estàndard les xifres de `n` en sentit invers, seguides de les mateixes xifres en el sentit en que apareixen en `n`. Per exemple, si `n` és 4873, cal escriure 37844873; si `n` és 48, cal escriure 8448; si `n` és 4, cal escriure 44.

Es demana:

a) (0.75 punts) Perfil del mètode amb la seua precondició.
b) (1.25 punts) Cas base i cas general.
c) (2 punts) Implementació en Java.

Solució:

a) Una possible solució consisteix en definir un mètode amb el següent perfil:

```
/** Precondició: n > 0 */
public static void xifres(int n)
```

b)
- Cas base: `n` ≤ 9, només una xifra.
- Cas general: `n` ≥ 10, més d'una xifra.

c)
```
/** Precondició: n > 0 */
public static void xifres(int n) {
    if (n <= 9) { System.out.print(n + "" + n); }
    else {
        int xifra = n % 10;
        System.out.print(xifra);
        xifres(n / 10);
        System.out.print(xifra);
    }
}
```

Curs 2017/18

P1 - Curs 17/18: 4 punts

Donats un array `v` de `String` i un nombre natural `n`, escriu un mètode **recursiu** que retorne quants elements de l'array `v` tenen una longitud `n`.

Per exemple, si l'array `v` continguera {`"barco"`, `"autobus"`, `"tren"`, `"moto"`, `"bici"`}, i `n` fóra 4, llavors el mètode retornaria 3. Si l'array fóra el mateix, i `n` fóra 5, llavors el mètode retornaria 1. Per al mateix array `v`, i `n` igual a 3, el mètode retornaria 0.

Es demana:

a) (0.75 punts) Perfil del mètode, amb els paràmetres adequats per a resoldre recursivament el problema, i precondició relativa a aquests paràmetres.

b) (1.25 punts) Cas base i cas general.

c) (1.50 punts) Implementació en Java.

d) (0.50 punts) Crida inicial perquè es realitze el càlcul sobre tot l'array.

Solució:

a) Una possible solució consisteix a definir un mètode amb el següent perfil:

```
/** Precondició: n >= 0 && 0 <= pos <= v.length */
public static int comptarPar(String[] v, int n, int pos)
```

de manera que retorne quants elements de longitud `n` hi ha a l'array `v[pos..v.length - 1]`, sent $0 \leq$ `pos` $\leq$ `v.length`.

b)
- Cas base, `pos = v.length`: Subarray buit. Retorna 0.
- Cas general, `pos` $<$ `v.length`: Subarray amb un o més elements. Es calcula el nombre d'elements de longitud `n` al subarray `v[pos + 1..v.length - 1]` i a aquest nombre se li suma 1 si la longitud de `v[pos]` és `n`. Es retorna aquest nombre.

c)
```
/** Precondició: n >= 0 && 0 <= pos <= v.length */
public static int comptarPar(String[] v, int n, int pos) {
    if (pos == v.length) { return 0; }
    else {
        if (v[pos].length() == n) { return 1 + comptarPar(v, n, pos + 1); }
        else { return comptarPar(v, n, pos + 1); }
    }
}
```

d) Per a un array `v` i un nombre `n`, la crida `comptarPar(v, n, 0)` resol el problema enunciat.

RecP1 - Curs 17/18: 4 punts

Siga `a` un array de `double`, on les seues components representen valors de la coordenada Y d'una enumeració de punts pertanyents a una recta de pendent positiva, de manera que l'array està ordenat ascendentment.

S'ha d'escriure un mètode recursiu que cerque el punt de tall de la recta amb l'eix X, o que retorne `-1` si el punt de tall no es troba entre els d'`a`.

Per exemple: si l'array és {`-7.4, -1.3, 0.0, 1.8, 2.3, 3.6`}, el mètode ha de retornar `2`. Si l'array és {`-7.4, -1.3, 1.8, 2.6, 3.6`}, el mètode ha de retornar `-1`.

Es demana:

a) (0.75 punts) Perfil del mètode, amb els paràmetres adequats per a resoldre recursivament el problema, i precondició relativa a aquests paràmetres.

b) (1.25 punts) Cas base i cas general.

c) (1.50 punts) Implementació en Java.

d) (0.50 punts) Crida inicial perquè, donat un cert array `a`, es realitze el càlcul sobre tot l'array.

Solució:

1. Una possible solució consisteix a definir un mètode amb el següent perfil, i que segueix una estratègia de cerca dicotòmica:

```
/** Cerca el valor 0 en a[left..right]. Precondició: 0 <= left, right < a.length,
 *  a[left..right] ordenat ascendentment.  */
public static int intercepts(double[] a, int left, int right) {
    if (left > right) { return -1; }
    else {
        int middle = (left + right) / 2;
        if (a[middle] == 0) { return middle; }
        else if (a[middle] > 0) {
            return intercepts(a, left, middle - 1);
        } else {
            return intercepts(a, middle + 1, right); }
    }
}
```

   La crida inicial per a resoldre el problema sobre un cert array `a` hauria de ser `intercepts(a, 0, a.length - 1)`.

2. Una altra possible solució consisteix a definir un mètode amb el següent perfil, i que segueix una estratègia de cerca lineal:

```
/** Cerca el valor 0 en a[ini..a.length - 1].
 *  Precondició: 0 <= ini, a[ini..a.length - 1] ordenat ascendentment. */
public static int intercepts(double[] a, int ini) {
    if (ini >= a.length) { return -1; }
    else if (a[ini] == 0) { return ini; }
    else if (a[ini] > 0) {return -1;}
    else { return intercepts(a, ini + 1); }
}
```

   La crida inicial per a resoldre el problema sobre un cert array `a` hauria de ser `intercepts(a, 0)`.

Ambdós mètodes són $\Omega(1)$, encara que el primer és $O(logn)$, mentre que el segon és $O(n)$, sent $n =$ `a.length`.

Curs 2016/17

P1 - Curs 16/17: 4 punts

Donat un array `a` de `int` i un enter `x`, escriu un mètode **recursiu** que torne quants múltiples de `x` hi ha a l'array `a`.

Es demana:

a) (0.75 punts) Perfil del mètode, amb els paràmetres adequats per a resoldre recursivament el problema.

b) (1.25 punts) Cas base i cas general.

c) (1.50 punts) Implementació en Java.

d) (0.50 punts) Crida inicial perquè es realitze el càlcul sobre tot l'array.

Solució:

a) Una possible solució consisteix en definir un mètode amb el següent perfil:

```
/** Precondició: 0 <= pos */
public static int multiplesX(int[] a, int x, int pos)
```

de manera que torne quants múltiples de `x` hi ha a l'array `a[pos..a.length - 1]`, sent $0 \leq$ `pos`.

b)
- Cas base, `pos` $\geq$ `a.length`: Subarray buit. Torna 0.
- Cas general, `pos` $<$ `a.length`: Subarray d'un o més elements. Si `a[pos] % x == 0`, torna 1 més el número de múltiples de `x` en `a[pos + 1..a.length - 1]`; si no, torna el número de múltiples de `x` en `a[pos + 1..a.length - 1]`.

c)
```
/** Torna el número de múltiples de x en a[pos..a.length - 1].
 *  Precondició: 0 <= pos */
public static int multiplesX(int[] a, int x, int pos) {
    if (pos >= a.length) { return 0; }
    else if (a[pos] % x == 0) { return 1 + multiplesX(a, x, pos + 1); }
    else { return multiplesX(a, x, pos + 1); }
}
```

d) Per un array `a`, la crida `multiplesX(a, x, 0)` resol el problema enunciat.

RecP1 - Curs 16/17: 3 punts

Un número triangular és aquell que pot recomposar-se en la forma d'un triangle equilàter (considerarem que el primer número triangular és el 0). Els primers 10 números triangulars són: 0, 1, 3, 6, 10, 15, 21, 28, 36, 45.

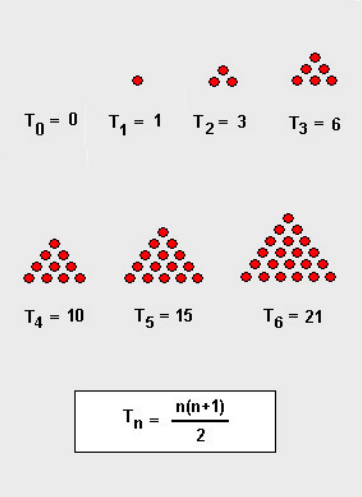

Figura 1: Representació gràfica dels 7 primers números triangulars

Cada número triangular es pot calcular com:

$$t(n) = \begin{cases} 0 & n = 0 \\ n + t(n-1) & n > 0 \end{cases}$$

On $t(n)$ és l'enèsim número triangular i $t(n-1)$ el seu anterior.

Es vol implementar, usant recursió, un algorisme que permeta instanciar un array d'un cert tamany n, i que continga els n primers termes d'aquesta successió dels números triangulars.

Es demana implementar els següents mètodes, per als que a més s'hauran d'especificar les precondicions necessàries:

- Un primer mètode, llançadora, no recursiu, que reba un únic argument, el número n, que haurà de ser un valor enter positiu. Aquest mètode haurà d'instanciar l'array al tamany adequat, invocar al segon mètode (encarregat de reomplir l'array), i retornar l'array.
- Un segon mètode, recursiu, que reba dos arguments: l' array de tamany n i un altre argument, necessari per a implementar el disseny recursiu. En aquest mètode s'hauran de calcular els valors de la successió, i magatzemmar-los en les posicions que corresponga en l'array.

Per exemple, la crida al mètode llançadora amb $n = 5$ haurà de retornar l'array $\{0, 1, 3, 6, 10\}$, mentre que amb $n = 1$ haurà de retornar l'array $\{0\}$.

Solució: Una possible solució consisteix a calcular cada nombre triangular, en el cas general, mitjançant l'equació $t(n) = n + t(n-1)$.

```
/** Precondició: n>0 */
public static int[] creaSuccessioTriangulars(int n) {
    int[] a = new int[n];
    creaSuccessioTriangulars(a, n - 1);
    return a;
}

/** Precondició: 0 <= n < a.length */
private static void creaSuccessioTriangulars(int[] a, int n) {
    if (n == 0) { a[n] = 0; }
    else {
        creaSuccessioTriangulars(a, n - 1);
        a[n] = n + a[n - 1];
    }
}
```

Una altra possible solució consisteix a calcular cada nombre triangular mitjançant l'equació $t(n) = n * (n+1)/2$ mostrada en la Figura 1. En aquest cas es presenta, a més, una solució recursiva basada en descomposició ascendent.

```
/** Precondició: n > 0 */
public static int[] creaSuccessioTriangulars(int n) {
    int[] a = new int[n];
    creaSuccessioTriangulars(a, 0);
    return a;
}

/** Precondició: 0 <= n < a.length */
private static void creaSuccessioTriangulars(int[] a, int n) {
    if (n < a.length) {
        a[n] = n * (n + 1) / 2;
        creaSuccessioTriangulars(a, n + 1);
    }
}
```

Curs 2015/16

P1 - Curs 15/16: 4 punts

Donat un array `a` de `int` ordenat ascendentment i un enter `x`, escriu un mètode **recursiu** que calcule la suma dels elements menors que `x` que hi ha a `a`. Per exemple, per `a` = {2,2,7,8,10,10,12,23,34} i `x` = 10 ha de tornar 19; per al mateix array i `x` = 1 ha de tornar 0. El mètode ha d'evitar fer comparacions supèrflues d'elements de `a` amb `x`.

Es demana:

a) (0.75 punts) Perfil del mètode, amb els paràmetres adequats per a resoldre recursivament el problema.
b) (1.25 punts) Cas base i cas general.
c) (1.5 punts) Implementació en Java.
d) (0.5 punts) Crida inicial perquè es realitze el càlcul sobre tot l'array.

Solució:

a) Una possible solució consisteix en definir un mètode amb el següent perfil:

```
/** Precondició: a ordenat ascendentment i 0 <= pos. */
public static int sumaMenors(int[] a, int pos, int x)
```

de manera que torne la suma dels elements de `a[pos..a.length - 1]` menors que `x`, sent `0`≤`pos`.

b)
- Cas base, `pos` ≥ `a.length`: Subarray buit. Torna 0.
- Cas general, `pos` < `a.length`: Subarray d'un o més elements. Si `a[pos]` es menor que `x`, torna la suma de `a[pos]` més la suma dels elements menors que `x` en `a[pos + 1..a.length - 1]`; però si `a[pos]` ≥ `x`, i en estar l'array ordenat ascendentment, es retorna 0 sense necessitat de fer més comprovacions.

c)
```
/** Torna la suma dels elements de a [pos..a.length-1] menors que x.
 *  Precondició: a ordenat ascendentment i 0 <= pos. */
public static int sumaMenors(int[] a, int pos, int x) {
    if (pos < a.length) {
        if (a[pos] >= x) { return 0; }
        else { return a[pos] + sumaMenors(a, pos + 1, x); }
    } else { return 0; }
}
```

d) Per un array `a`, la crida `sumaMenors(a, 0, x)` resol el problema enunciat.

RecP1 - Curs 15/16: 4 punts

Donat un array `a` de `int` on `a.length` ≥ 1, escriure un mètode **recursiu** que determine si els elements de l'array representen una successió de Fibonacci, és a dir, si el valor d'un element correspon a la suma dels dos elements immediatament anteriors (sent els dos primers elements 0 i 1), tal com es mostra en el següent exemple: {0,1,1,2,3,5,8,13,21,...}.

Es demana:

a) (0.75 punts) Perfil del mètode, amb els paràmetres adequats per a resoldre recursivamente el problema.
b) (1.25 punts) Cas base i cas general.
c) (1.5 punts) Implementació en Java.
d) (0.5 punts) Crida inicial per a que determine si es compleix la successió sobre tot l'array.

Solució:

a) Una possible solució consisteix a definir un mètode amb el següent perfil:

```
/** Precondició: a.length >= 1 i 0 <= pos <= a.length - 1. */
public static boolean esFibo(int[] a, int pos)
```

de manera que determine si els elements de `a[0..pos]` representen una successió de Fibonacci, sent $0 \leq$ `pos` $\leq$ `a.length - 1`.

b)
- Cas base, `pos == 0`: Verifica que `a[0] == 0`.
- Cas base, `pos == 1`: Verifica que `a[1] == 1` i que `a[0] == 0`.
- Cas general, `pos > 1`: Subarray de tres o més elements. Verifica que `a[pos] == a[pos - 1] + a[pos - 2]`.

c)
```
/** Determina si els elements de a[0..pos] representen una successió de Fibonacci.
 *  Precondició: a.length >= 1 i 0 <= pos <= a.length - 1. */
public static boolean esFibo(int[] a, int pos) {
    if (pos == 0) { return a[0] == 0; }
    else if (pos == 1) { return a[1] == 1 && a[0] == 0; }
    else { return a[pos] == a[pos - 1] + a[pos - 2] && esFibo(a, pos - 1); }
}
```

d) Per a un array `a`, la crida `esFibo(a, a.length - 1)` resol el problema de l'enunciat.

Curs 2014/15

P1 - Curs 14/15: 3 punts

Donat un array `a` de `int` amb almenys un element, escriure un mètode **recursiu** que comprove si tots els valors de l'array són parells i estan emmagatzemats en ordre creixent.

Es demana:

a) (0.5 punts) Perfil del mètode, amb els paràmetres adequats per tal de resoldre recursivament el problema.
b) (1.2 punts) Cas base i cas general.
c) (1 punt) Implementació en Java.
d) (0.3 punts) Crida inicial perquè es verifique la propietat sobre tot l'array.

Solució:

a) Una possible solució consisteix en definir un mètode amb el següent perfil:

```
/** Precondició: a.length > 0 i 0 <= pos < a.length */
public static boolean parellsICreixents(int[] a, int pos)
```

de manera que determine si els elements de `a[pos..a.length - 1]` són parells i estan emmagatzemats en ordre creixent, sent $0 \leq$ `pos` $<$ `a.length`.

b)
- Cas base, `pos == a.length - 1`: Subarray d'un element. Retorna `true` si `a[pos]` és parell i `false` en cas contrari.
- Cas general, `pos < a.length - 1`: Subarray de més d'un element. Retorna `true` sii `a[pos]` és parell i `a[pos] <= a[pos + 1]` i els elements del subarray `a[pos + 1..a.length - 1]` són parells i estan ordenats creixentment. En un altre cas, retorna `false`.

c)
```
/** Comprova si tots els elements de l'array a són parells
 *  i estan ordenats creixentment.
 *  Precondició: a.length > 0 i 0 <= pos < a.length. */
public static boolean parellsICreixents(int[] a, int pos) {
    if (pos == a.length - 1) { return a[pos] % 2 == 0; }
    else {
        return a[pos] % 2 == 0 && a[pos] <= a[pos + 1]
            && parellsICreixents(a, pos + 1);
    }
}
```

d) Per a un array `a`, la crida `parellsICreixents(a, 0)` resol el problema de l'enunciat.

Curs 2013/14

P1 - Curs 13/14: 3 punts

Donat un array `a` de `int`, escriure un mètode **recursiu** que sume els números capicua des dels extrems de l'array, definits per l'interval `[ini..fi]` ($0 \leq$ `ini`, `fi` $<$ `a.length`), cap al centre fins sumar-los tots o trobar algun parell de números que no siguen iguals. En cas que l'array tinga una quantitat senar d'elements, l'element central de l'interval només ha de sumar-se una vegada. Si l'array no té cap element capicua en els extrems de l'interval, el resultat és zero.

Alguns exemples:

- per a l'array `a = {1,4,1,2,4,1}`, el resultat de la suma és 10 (1+1+4+4),
- per a l'array `a = {1,4,2,2,4,1}`, el resultat de la suma és 14 (1+1+4+4+2+2),
- per a l'array `a = {1,4,1,4,1}`, el resultat de la suma és 11 (1+1+4+4+1),
- per a l'array `a = {1,4,0,0,1}`, el resultat de la suma és 2 (1+1),
- per a l'array `a = {8,4,0,0,1}`, el resultat de la suma és 0.

Es demana:

a) (0.5 punts) Perfil del mètode, amb els paràmetres adequats per tal de resoldre recursivament el problema.

b) (1.2 punts) Cas base i cas general.

c) (1 punt) Implementació en Java.

d) (0.3 punts) Crida inicial.

Solució:

a) Una possible solució consisteix en definir un mètode amb el següent perfil:

```
/** Precondició: 0 <= ini, fi < a.length. */
public static int sumaCapicua(int[] a, int ini, int fi)
```

de manera que torne la suma dels elements capicua de `a[ini..fi]` des dels extrems cap al centre fins sumar-los tots o trobar algun parell de números que no siguen iguals, sent $0 \leq$ `ini` i `fi` $<$ `a.length`.

b)
- Cas base, `ini > fi`: Subarray buit. Retorna el neutre de la suma 0.
- Cas base, `ini == fi`: Subarray d'un element. Retorna `a[ini]`.
- Cas general, `ini < fi`: Si els extrems no són capicua (`a[ini] != a[fi]`), retorna el neutre de la suma 0. En un altre cas (`a[ini] == a[fi]`), el resultat és la suma dels valors dels extrems `a[ini]` i `a[fi]` més la suma dels valors capicua des dels extrems del subarray `a[ini + 1..fi - 1]`.

c)
```
/** Suma dels valors capicua des dels extrems del subarray a[ini..fi].
 *  Si el subarray està buit o no existeixen capicua en els extrems,
 *  el resultat és zero.
 *  Precondició: 0 <= ini, fi < a.length. */
public static int sumaCapicua(int[] a, int ini, int fi) {
    if (ini > fi) { return 0; }
    else if (ini == fi) { return a[ini]; }
    else if (a[ini] != a[fi]) { return 0; }
    else { return a[ini] + a[fi] + sumaCapicua(a, ini + 1, fi - 1); }
}
```

d) Per a un array `a`, la crida `sumaCapicua(a, 0, a.length - 1)` resol el problema de l'enunciat.

RecP1 - Curs 13/14: 3 punts

Es demana: dissenyar un mètode **recursiu** que determine si cert array d'enters `v` conté els elements inicials d'una successió de Fibonacci. Per a això l'array es proporciona amb almenys 3 elements. Cal recordar que la successió de Fibonacci és la següent: 0,1,1,2,3,5,8 ... És a dir, el terme n-èsim és `n` si $n = 0$ o $n = 1$; en un altre cas, es calcula com la suma dels dos termes anteriors ((`n-1`)-èsim i (`n-2`)-èsim).

(a) (2.5 punts). Escriure el mètode d'acord a les consideracions anteriors.

Solució:

mitjançant recorregut ascendent:

```
/** Precondició: v.length >= 3, 2 <= i <= v.length. */
public static boolean es_fibo(int[] v, int i) {
    if (i == v.length) { return true; }
    if (v[0] != 0 || v[1] != 1) { return false; }
    else { return v[i] == v[i - 1] + v[i - 2] && es_fibo(v, i + 1); }
}
```

mitjançant recorregut descendent:

```
/** Precondició: v.length >= 3, 1 <= i <= v.length - 1. */
public static boolean es_fibo(int[] v, int i) {
    if (i <= 1) { return v[0] == 0 && v[1] == 1; }
    else { return v[i] == v[i - 1] + v[i - 2] && es_fibo(v, i - 1); }
}
```

(b) (0.5 punts). D'acord a la implementació del mètode anterior, indicar la crida inicial perquè es verifique la propietat sobre tot l'array.

Solució: Assumint `v` un array amb almenys 3 elements, la crida inicial, per a la solució recursiva ascendent, seria: `es_fibo(v, 2)`

I per a la solució recursiva descendent: `es_fibo(v, v.length - 1)`

Curs 2012/13

P1 - Curs 12/13: 4 punts

Donat un array `a` d'`int` i un enter `m`, escriure un mètode **recursiu** que comprove si existeix una parella `a[i]` i `a[f]`, $0 \leq$ `i` $\leq$ `f` $<$ `a.length`, de components *simètriques* (la distància d'`i` a 0 és la mateixa que la distància de `f` a `a.length - 1`) que sumen `m`. Si existeix la parella, ha de tornar l'índex en el que es troba (l'índex `i`, el més baix de la parella), -1 en cas contrari.

Per exemple, per a `m` $= 4$ i `a` $= \{1,4,5,9,6,0,-8\}$ el mètode en la crida inicial que s'extenga sobre tot l'array ha de tornar 1, per a `m` $= 18$ i `a` $= \{1,4,5,9,6,0,2\}$ ha de tornar 3, per a `m` $= 25$ i `a` $= \{1,3,2,5,4,6\}$ ha de tornar -1.

Es demana:

a) Perfil del mètode, amb els paràmetres adequats per tal de resoldre recursivament el problema.
b) Cas base i cas general.
c) Implementació en Java.
d) Crida inicial.

Solució:

a) Una possible solució consisteix a definir el mètode com

```
/** Precondició: 0 <= ini, fi < a.length. */
public static int parellaSim(int[] a, int ini, int fi, int m)
```

de manera que éssent `0 <= ini, fi < a.length`, es delimita a cercar una parella d'elements simètrics en el subarray `a[ini..fi]`.

b)
- Cas base `ini > fi`: No es troba la parella buscada, s'ha de retornar -1.
- Cas general, `ini <= fi`. Si `a[ini] + a[fi]` val `m`, s'ha de retornar `ini`, sinó la cerca es redueix a `a[ini + 1..fi - 1]`.

c)
```
/** Busca l'índex de la primera parella d'elements simètrics en a[ini..fi],
 *  que sumats dóna m. Si no existeix, retorna -1.
 *  Precondició: 0 <= ini, fi < a.length. */
public static int parellaSim(int[] a, int ini, int fi, int m) {
    if (ini > fi) { return -1; }
    else if (a[ini] + a[fi] == m) { return ini; }
    else { return parellaSim(a, ini + 1, fi - 1, m); }
}
```

d) Per a un array `a`, i un enter `m`, la crida `parellaSim(a, 0, a.length - 1, m)` resol el problema de l'enunciat.

RecP1 - Curs 12/13: 2.5 punts

Per a determinar si cert nombre enter no negatiu `n` pot estar expressat en una base determinada `b` ($2 \leq$ `b` ≤ 10), és suficient que tots els dígits del nombre tinguen 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.

Es demana: implementar en Java un mètode **recursiu** que, donats `n` i `b`, resolga el problema plantejat, especificant els casos base i general de la recursió.

Solució:

```
/** Torna si és o no possible que n estiga en base b.
 *  Precondició: 2 <= b <= 10 i n >= 0. */
public static boolean basePossible(int n, int b) {
    if (n == 0) { return true; }          // cas base
    else {                                // cas general
        int ultDig = n % 10;
        if (ultDig < b) { return basePossible(n / 10, b); }
        else { return false; }
    }
}
```

RecP1 - Curs 12/13: 2.0 punts

Donat un array `a` de valors reals i cert valor real `x`, es vol determinar, **recursivament**, el nombre d'elements d'`a`, des de la posició `pos` inclosa fins a la final (`a[pos..a.length - 1]`), que tinguen valor més gran que `x`. Per a això, partint del perfil:

```
public static int numMajors(double[] a, double x, int pos)
```

Es demana:

- Implementar el mètode **recursiu** demanat, especificant els casos base i general de la recursió.
- Escriure la crida inicial per obtenir el nombre d'elements més grans que `x` de tot un array.

Solució:

```
/** Calcula el nombre d'elements amb valor major que un x donat
 *  des de la posició pos.
 *  Precondició: 0 <= pos. */
public static int numMajors(double[] a, double x, int pos) {
    if (pos >= a.length) { return 0; }                              // cas base
    else if (a[pos] > x) { return 1 + numMajors(a, x, pos + 1); }   // cas general
    else { return numMajors(a, x, pos + 1); }                       // cas general
}
```

Crida inicial: `int num = numMajors(v, x, 0);` complint `v` i `x` les condicions donades per la precondició del mètode.

Curs 2011/12

P1 - Curs 11/12: 2 punts

Escriu un mètode **recursiu** que reba com argument un valor enter no negatiu i escriga en l'eixida estàndard les seqüències descendent i ascendent de valors enters des del número inicial fins el zero. Cada seqüència en una línia diferent.

Per exemple, si el mètode sol·licitat s'executa amb el valor 14, aleshores escriurà per l'eixida estàndard el següent:

```
14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

Nota que els números de cada seqüència apareixen tots seguits en una mateixa línia.

Solució:

```
/** Precondició: n >= 0. */
public static void sequencies(int n) {
    if (n > 0) {
        System.out.print(n + " ");
        sequencies(n - 1);
        System.out.print(" " + n);
    }
    else { System.out.print("0\n0"); }
}
```

P1 - Curs 11/12: 2 punts

Escriu un mètode **recursiu** que, donats un array d'enters `v`, una posició inicial `ini >= 0`, una posició final `fi < v.length` i un valor enter `x`, determine si la suma dels elements simètrics és `x`, és a dir, els extrems dels subarrays formats per les posicions en l'interval `[ini + i, fi - i]` per a `0 <= i <= (ini + fi) / 2` compleixen que `v[ini + i] + v[fi - i] == x`.

Per exemple:

Si `x = 10, ini = 1, fi = v.length - 2` i `v = {1,2,3,4,5,6,7,8,9}` retorna `true`.

Si `x = 10, ini = 1, fi = v.length - 2` i `v = {1,2,4,4,5,6,7,8,9}` retorna `false`.

Si `x = 6, ini = 0, fi = v.length - 1` i `v = {1,2,3,3,4,5}` retorna `true`.

Solució:

```
/** Precondició: ini >= 0, fi < v.length. */
public static boolean metode(int[] v, int ini, int fi, int x) {
    if (ini > fi) { return true; }
    else if (ini == fi) { return (2 * v[ini]) == x; }
    else if (v[ini] + v[fi] != x) { return false; }
    else { return metode(v, ini + 1, fi - 1, x); }
}
```

RecP1 - Curs 11/12: 2 punts

Siga `a` un array d'`int` i `x` un `int`. **Es demana** un mètode **recursiu** que indique si els elements d'`a` formen una progressió geomètrica de raó `x`, és a dir, si cada component `a[i + 1]` de l'array val `a[i] * x`. Per exemple, per a `a = {3,6,12,24,48}` i `x` = 2 el mètode ha de retornar `true`, per a `a = {3,6,12,33,48}` i `x` = 2 el mètode ha de retornar `false`. Si sols hi ha un element, s'entén que és una progressió geomètrica siga quin siga `x`.

Indicar quina haurà de ser la primera crida.

Solució:

```
/** Comprova si les components de l'array a[i..a.length - 1]
 *  formen una progressió geomètrica de raó x.
 *  Precondició: 0 <= i < a.length. */
public static boolean geometrica(int[] a, int i, int x) {
    if (i == a.length - 1) { return true; }
    else if (a[i + 1] == a[i] * x) { return geometrica(a, i + 1, x); }
    else { return false; }
}
```

La primera crida hauria de ser `geometrica(a, 0, x)`.

RecP1 - Curs 11/12: 2 punts

Escriure un mètode **recursiu** que donat un enter $\texttt{n} \geq 0$, escriga en la sortida estàndard, i en la mateixa línia, els valors `-n -(n - 1) ... -2 -1 0 1 2 ... (n - 1) n`. Per exemple, per a $\texttt{n} = 3$, en la sortida s'ha d'escriure:

```
-3 -2 -1 0 1 2 3
```

Solució:

```
/** Precondició: n >= 0. */
public static void escriu(int n) {
    if (n == 0) { System.out.print(0); }
    else {
        System.out.print(-n + " ");
        escriu(n - 1);
        System.out.print(" " + n);
    }
}
```