

Tema 1. Recursió

Programació (PRG)

Curs 2019/20

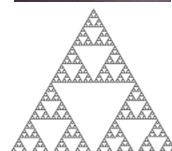
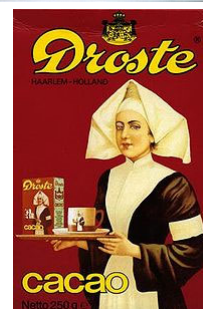
Departament de Sistemes Informàtics i Computació



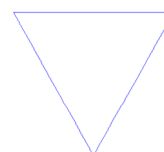
Continguts

Duració: 6 sessions

1. Introducció
 2. Disseny d'un mètode recursiu
 3. Tipus de recursió
 4. Recursivitat i pila de crides
 5. Alguns exemples
 6. Recursió amb arrays: recorregut i cerca
 - Esquemes recursius de recorregut
 - Esquemes recursius de cerca
 - Cerca binària iterativa i recursiva
 7. Recursió versus iteració
- Pràctiques relacionades:
 - PL 1. Resolució recursiva del dibuix d'una figura.
 - PL 2. Resolució d'alguns problemes amb recursió.



- **Descarrega** (del Tema 1 de PoliformaT) el fitxer **exercicisT1.jar** en una carpeta **PRG/Tema 1** dins del teu **disc W**
- Des de l'opció **Projecte** de **BlueJ**, usa l'opció **Open ZIP/JAR...** per tal d'obrir-lo com un projecte **BlueJ** i prepara't per usar-lo



Introducció

- S'anomena **recursiu** a qualsevol ent (definició, procés, estructura, etc.) que **es defineix en funció de si mateix**.
- Una funció és recursiva si la seua resolució requereix la solució prèvia de la funció per a un cas més senzill. Per exemple, la funció factorial:

$$n! = \begin{cases} 1 & n = 0 \\ \prod_{i=1}^n i & n > 0 \end{cases}$$

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

- Un **algorisme** és **recursiu** si obté la solució d'un problema sobre la base dels resultats que ell mateix proporciona per a casos més senzills del mateix problema, és a dir, si **s'invoca a si mateix** ...
- Així, per resoldre un problema complex mitjançant un algorisme recursiu es descompon el problema en una sèrie de problemes més simples que es resolen **usant el mateix algorisme**, per després compondre la solució del problema complex sobre la base de les solucions dels problemes més simples.

28/01/2020



PRG - Curs 2019/20



3

blocs(4) = 30

Una piràmide escalonada d'altura n , $n \geq 1$, és una construcció de n nivells de costats decreixents $n, n-1, \dots, 2, 1$. En la piràmide, el nivell i està format per una plataforma quadrada de $i * i$ blocs. Donat un enter $n \geq 1$, per quants blocs està formada una piràmide escalonada d'altura n ?

blocs(3) = 14

blocs(2) = 5

blocs(1) = 1

blocs(n)?

si n és 1,
blocs(n) és 1
sino
blocs(n) és $n*n + \text{blocs}(n-1)$

$4 * 4$

$3 * 3$

$2 * 2$

28/01/2020



PRG - Curs 2019/20



4

Introducció

- Hi ha dues situacions diferents a l'hora de resoldre el problema plantejat:
 - **Cas base**: el problema és lo suficientment simple per a ser resolt de manera trivial.
 - **Cas general**: el problema requereix l'ús de la resolució d'una instància més simple per trobar la seua solució.
- Una vegada plantejat un algorisme recursiu és necessari verificar:
 1. La **terminació** de l'algorisme, és a dir, que en algun moment s'arribarà al cas base a partir de qualsevol cas general plantejat inicialment.
 2. La **correcció** de l'algorisme, és a dir, que per a qualsevol subproblema que es done, la solució de l'algorisme és correcta (sol requerir una demostració matemàtica per inducció sobre algun paràmetre de l'algorisme).

Disseny d'un mètode recursiu

- Els algorismes recursius s'implementen fàcilment usant **mètodes recursius**.

```

/** PRECONDICIÓ: n >= 0 */
public static int factorial(int n) {
    if (n == 0) { // Condició del cas base
        return 1; // Instruccions del cas base
    }
    else { /* n > 0 */ // Condició del cas general
        return n * factorial(n-1); // Instruccions del cas general
    }
}

```

- L'estructura fonamental consisteix en tenir una instrucció **condicional** que permeta distingir entre cas base i general.
- Les crides recursives s'han de fer cada vegada per a casos estrictament més simples.

Disseny d'un mètode recursiu - Etapes

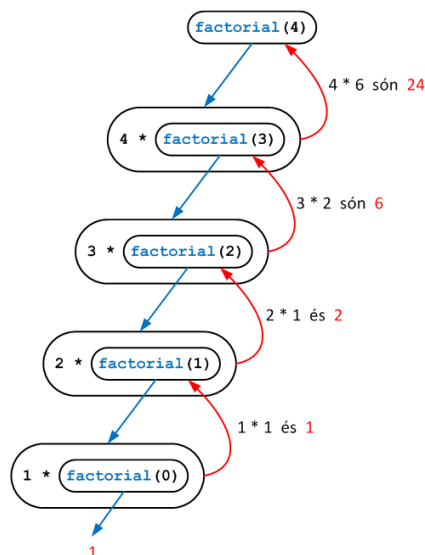
1. **Enunciat del problema.** Consisteix en **declarar la capçalera del mètode** que es construirà, i establir tant les **condicions d'entrada (precondició)** per a les quals s'haurà d'executar l'algorisme, com el **resultat esperat** d'aquesta execució.
2. **Anàlisi de casos.** Consisteix en fer explícit el **cas base** i el **cas general** de la recursió (són **complementaris i excloents**), establint per a cada cas les instruccions pertinents per resoldre el problema. Cal comprovar que es cobreix qualsevol cas possible, és a dir, que davant qualsevol possible entrada del problema, s'efectua el cas base o el general.
3. **Transcripció de l'algorisme a un mètode en Java.**
4. **Validació del disseny.** Determinar que en **cada nova crida recursiva** es compleix que:
 - El nou problema que s'ha de resoldre és **estrictament més proper al cas base** que el problema original, i
 - les dades d'entrada de la nova crida compleixen les **condicions d'entrada** establertes per executar l'algorisme.

Tipus de recursió

- **Recursió Lineal:** hi ha com a molt **una sola crida recursiva** en cada execució de l'algorisme, generant una **seqüència de crides**.
 - **Final:** el resultat de la crida recursiva és el propi resultat de la crida en curs, és a dir, la solució del cas més simple és, al seu torn, la del cas més complex, de manera que no hi ha combinació de resultats.
 - **No final:** el resultat de la crida recursiva s'utilitza per calcular el resultat de la crida actual, llançant un resultat possiblement diferent al de la crida recursiva.
- **Recursió Múltiple:** hi ha **més d'una crida recursiva** en cada execució de l'algorisme, i els seus resultats s'han de combinar per obtenir la solució del cas actual. Es genera un **arbre de crides**.

Tipus de recursió

- Per exemple, el mètode `factorial` és un mètode **lineal** (es fa una única crida recursiva a `factorial(n-1)` en cada execució) **no final** (el resultat de la crida es multiplica per n per retornar-se).
- Per a `factorial(4)` es genera la **seqüència de crides** de la figura.



Recursivitat i pila de crides

- Cada crida recursiva a un mateix mètode està associada a un **registre d'activació** propi que s'empila a la pila de crides. És a dir, hi ha tants registres d'activació com crides pendents. De tots ells, només està **actiu** el que està en el **cim** de la pila.
- Quan una execució d'un mètode finalitza, deixa d'existir el seu registre d'activació (es desempila). En el cas recursiu, l'execució pot reprendre's en una execució immediatament anterior del mateix mètode, que haurà deixat d'estar pendent.
- En la recursió es fa un ús intensiu de la **pila de crides**.
- Pot arribar a provocar seriosos problemes per esgotament de la memòria, produint-se un desbordament de la pila (**stack overflow**).
- La causa habitual del desbordament de la pila és la recursió infinita, provocant l'excepció **StackOverflowError**.



```

/** PRECONDICIÓ: n >= 0 */
public static int factorial(int n) {
    int r;
    if (n == 0) { r = 1; }
    ➔ else { r = n * factorial(n - 1); } *
    return r;
}

public static void main(String[] args) {
    int f = factorial(3); ●
}

```

Prova.main		Prova.factorial	
args {}	AR f	VR n 0	AR * r
args {}	AR f	VR n 1	AR * r
args {}	AR f	VR n 2	AR * r
args {}	AR f	VR n 3	AR ● r

28/01/2020  UNIVERSITAT POLITÈCNICA DE VALÈNCIA PRG - Curs 2019/20  etsinf 11



```

/** PRECONDICIÓ: n >= 0 */
public static int factorial(int n) {
    int r;
    if (n == 0) { r = 1; }
    else { r = n * factorial(n - 1); } *
    ➔ return r;
}

public static void main(String[] args) {
    int f = factorial(3); ●
}

```

Prova.main		Prova.factorial	
args {}	AR f	VR n 0	AR * r 1
args {}	AR f	VR n 1	AR * r
args {}	AR f	VR n 2	AR * r
args {}	AR f	VR n 3	AR ● r

28/01/2020  UNIVERSITAT POLITÈCNICA DE VALÈNCIA PRG - Curs 2019/20  etsinf 12

Recursivitat i pila de crides

- Si es compara l'ús de la pila que provoca la crida `factorial(n)` en la versió iterativa i recursiva del mètode, es pot concloure que aquest ús és més gran en el cas recursiu que en l'iteratiu.
- En la **versió iterativa** de `factorial` coexisteixen simultàniament en memòria, com a molt, el registre d'activació del mètode `factorial` i el registre d'activació del mètode `main`.
- En la pila de crides de la **versió recursiva** de `factorial` poden arribar a coexistir simultàniament $n+1$ registres d'activació de les distintes crides a `factorial` més el registre d'activació del mètode `main`.
- És a dir, el consum de memòria del mètode `factorial` **iteratiu** és sempre el mateix mentre que el del `factorial` **recursiu** depèn del valor de n .

Alguns exemples – Potència n-èsima

1. **Enunciat del problema.** Donat un número natural $n \geq 0$ i un número real $a \neq 0$, calcular la potència a^n .

```
/** PRECONDICIÓ: n >= 0 i a != 0 */
public static double potencia(double a, int n)
```

2. **Anàlisi de casos.**

- **Cas base:** Si $n=0$, aleshores $a^n = 1$.
- **Cas general:** Si $n>0$, aleshores $a^n = a^{n-1} * a$.

3. **Transcripció de l'algorisme a un mètode en Java.**

```
/** PRECONDICIÓ: n >= 0 i a != 0 */
public static double potencia(double a, int n) {
    if (n == 0) { return 1; }
    else { return potencia(a, n - 1) * a; }
}
```

recursió lineal no final

4. **Validació del disseny.**

- En el cas general, en cada crida el valor del segon paràmetre decreix en 1, així, en algun moment arribarà a ser 0, aconseguint el cas base i finalitzant l'algorisme.
- A més, sempre es compleix que el valor de $n \geq 0$ i $a \neq 0$.

Alguns exemples – Residu de la divisió entera

1. **Enunciat del problema.** Donats dos números naturals $a \geq 0$ i $b > 0$, calcular el residu de la seua divisió entera a/b .

```
/** PRECONDICIÓ: a >= 0 i b > 0 */
public static int residu(int a, int b)
```

2. **Anàlisi de casos.**

- **Cas base:** Si $a < b$, aleshores el residu de a/b és a .
- **Cas general:** En un altre cas, el residu de a/b és el residu de $(a-b)/b$.

3. **Transcripció de l'algorisme a un mètode en Java.**

```
/** PRECONDICIÓ: a >= 0 i b > 0 */
public static int residu(int a, int b) {
    if (a < b) { return a; }
    else { return residu(a - b, b); }
}
```

recursió lineal final

4. **Validació del disseny.**

- En el cas general, en cada crida el valor del primer paràmetre va decreixent, en algun moment serà inferior al segon paràmetre, aconseguint el cas base i finalitzant l'algorisme.
- A més, sempre es compleix que $a \geq 0$ i $b > 0$.

Alguns exemples – Successió de Fibonacci

1. **Enunciat del problema.** Calcular el terme n -èsim de la successió de Fibonacci (assumint que el terme 0-èsim és el primer de la successió).

```
/** PRECONDICIÓ: n >= 0 */
public static int fibonacci(int n)
```

2. **Anàlisi de casos.**

- **Cas base:** Si $n \leq 1$, el valor del terme n -èsim és n .
- **Cas general:** En un altre cas, és la suma dels termes $(n-1)$ -èsim i $(n-2)$ -èsim.

3. **Transcripció de l'algorisme a un mètode en Java.**

```
/** PRECONDICIÓ: n >= 0 */
public static int fibonacci(int n) {
    if (n <= 1) { return n; }
    else { return fibonacci(n - 1) + fibonacci(n - 2); }
}
```

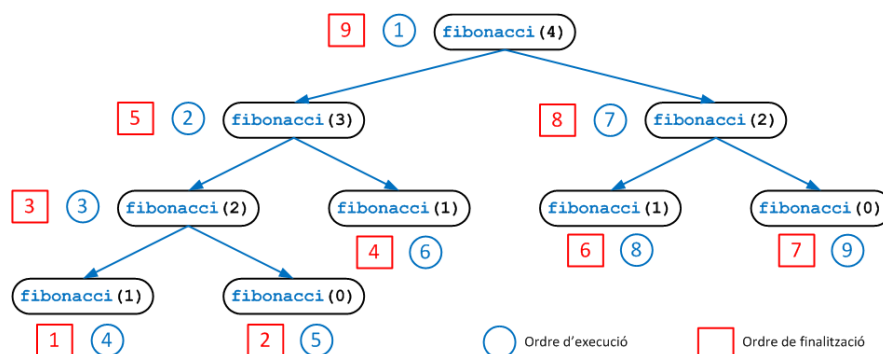
recursió múltiple

4. **Validació del disseny.**

- En el cas general, en cada crida es va tendint a termes inferiors de la successió, en algun moment es complirà la condició del cas base, finalitzant l'algorisme.
- A més, sempre es compleix que $n \geq 0$.

Alguns exemples – Successió de Fibonacci

- El mètode `fibonacci` és un mètode recursiu **múltiple** (en el cas general es realitzen dues crides recursives i el resultat es construeix a partir de la suma dels resultats de les dues crides).
- Per a `fibonacci(4)` es genera l'**arbre de crides** de la figura.



28/01/2020



PRG - Curs 2019/20



17

Alguns exemples – Algorisme d'Euclides I

- Enunciat del problema.** Donats dos números naturals $a > 0$ i $b > 0$, calcular el seu màxim comú divisor (m.c.d.) seguint l'algorisme d'Euclides.

```

/** PRECONDICIÓ: a > 0 i b > 0 */
public static int mcd(int a, int b)

```

- Anàlisi de casos.**

- Cas base:** Si $a=b$, el m.c.d. és b .
- Cas general:** Si $a > b$, el m.c.d. és el m.c.d. de $a-b$ i b .
Si $a < b$, el m.c.d. és el m.c.d. de a i $b-a$.

- Transcripció de l'algorisme a un mètode en Java.**

```

/** PRECONDICIÓ: a > 0 i b > 0 */
public static int mcd(int a, int b) {
    if (a == b) { return b; }
    else if (a > b) { return mcd(a - b, b); }
    else { return mcd(a, b - a); }
}

```

recursió lineal final

- Validació del disseny.**

- En el cas general, en cada crida el valor del primer o segon paràmetre va decreixent, en algun moment arribaran a ser iguals i, en aquest cas, s'arribarà al cas base, finalitzant l'algorisme.
- A més, sempre es compleix que $a > 0$ i $b > 0$.

28/01/2020



PRG - Curs 2019/20



18

Alguns exemples – Algorisme d'Euclides II

1. **Enunciat del problema.** Donats dos números naturals $a > 0$ i $b > 0$, calcular el seu màxim comú divisor (m.c.d.) seguint l'algorisme d'Euclides.

```
/** PRECONDICIÓ: a > 0 i b > 0 */
public static int euclides(int a, int b)
```

2. **Anàlisi de casos.**

- **Cas base:** Si el residu de a/b és 0, el m.c.d. és b .
- **Cas general:** En un altre cas, el m.c.d. és el m.c.d. de b i el residu de a/b .

3. **Transcripció de l'algorisme a un mètode en Java.**

```
/** PRECONDICIÓ: a > 0 i b > 0 */
public static int euclides(int a, int b) {
    if (a % b == 0) { return b; }
    else { return euclides(b, a % b); }
}
```

recursió lineal final

4. **Validació del disseny.**

- En el cas general, en cada crida el valor del segon paràmetre va decreixent; en algun moment arribarà a ser el m.c.d. dels valors originals i, en aquest cas, s'arribarà al cas base, finalitzant l'algorisme.
- A més, sempre es compleix que $a > 0$ i $b > 0$.

Recursió amb arrays

- Donada la declaració d'un array **a** de **num** elements de tipus **tipusBase**:

```
tipusBase[] a = new tipusBase[num];
```

es pot considerar l'array **a** com una seqüència:

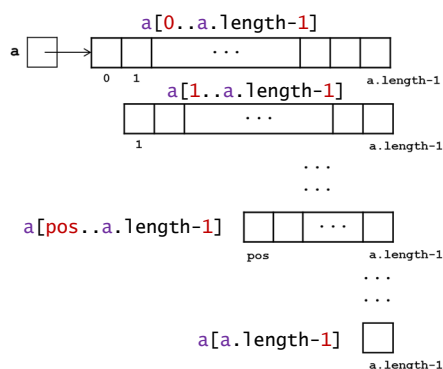
```
a[0], a[1], a[2], ..., a[a.length-2], a[a.length-1]
```

denotada com **a[0..a.length-1]**.

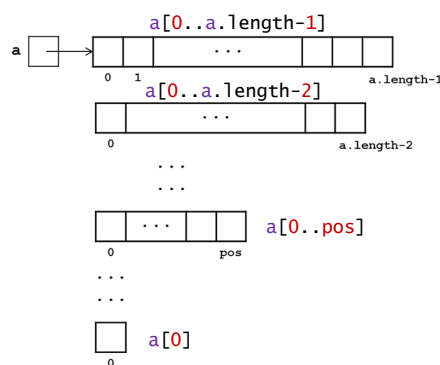
- Aquesta seqüència es pot definir **recursivament** com la seqüència formada per **a[0]** i la subseqüència (subarray) definida per la resta de components de **a**, és a dir, **a[1..a.length-1]**.
- Al seu torn, el subarray **a[1..a.length-1]** pot definir-se recursivament de la mateixa manera i així, successivament, fins que el subarray corresponent estiga buit, sense components. És el que s'anomena **descomposició recursiva ascendent**.
- Anàlogament, l'array **a** es pot definir de forma recursiva considerant el subarray **a[0..a.length-2]** i la seua última component **a[a.length-1]**, el que s'anomena **descomposició recursiva descendent**.

Recursió amb arrays

descomposició recursiva ascendent



descomposició recursiva descendent



28/01/2020



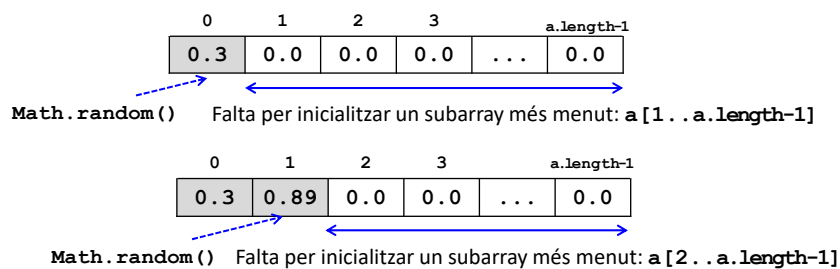
PRG - Curs 2019/20



21

Recursió amb arrays

- Els algoritmes recursius amb arrays es poden formular reduint el problema a subproblemes que treballen amb subarrays o parts de l'array més menudes, açò és, amb **menys components**; formalment, subproblemes d'una **talla** més menuda.
- S'anomena **talla** o **grandària** de un problema al valor o conjunt de valors associats a les dades d'entrada que representen una mesura de la dificultat per a la seua resolució.
- En el cas de problemes amb arrays, la **talla** és el número de components del (sub)array $a[\text{ini} \dots \text{fi}]$ en consideració, és a dir, $t = \text{fi} - \text{ini} + 1$.
- Exemple: implementar un mètode per tal d'inicialitzar un array a de double ($a.\text{length} > 0$) amb valors aleatoris en $[0.0, 1.0[$



28/01/2020

etc...



PRG - Curs 2019/20



22

Recursió amb arrays

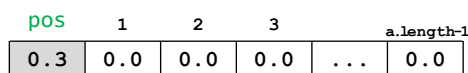
1. **Enunciat del problema.** Donat un array `a` de `double` (`a.length > 0`), inicialitzar els seus elements amb valors aleatoris en `[0.0, 1.0[`.

El mètode necessita un paràmetre addicional que indique on comença (acaba) el subarray a tractar, si és una descomposició recursiva ascendent (descendent):

```
/** Inicialitza amb valors aleatoris en [0.0,1.0[
 * els elements de a[pos..a.length-1]
 * PRECONDICIÓ: 0 <= pos < a.length, a.length > 0 */
public static void iniciar(double[] a, int pos)
```

2. **Anàlisi de casos.**

- **Cas general:** Si `pos < a.length-1`,



s'inicialitza `a[pos]` i falta per inicialitzar el subarray `a[pos+1..a.length-1]`

- **Cas base:** Si `pos == a.length-1`, aleshores només hi ha que inicialitzar `a[pos]`.
- En el cas general, `pos < a.length-1`, es resol el subproblema d'inicialitzar els elements del subarray `a[pos+1..a.length-1]`. És a dir, un array amb menys elements o d'una talla més menuda.

28/01/2020



PRG - Curs 2019/20



23

Recursió amb arrays

3. **Transcripció de l'algorisme a un mètode en Java.**

```
/** Inicialitza amb valors aleatoris en [0.0,1.0[
 * els elements de a[pos..a.length-1]
 * PRECONDICIÓ: 0 <= pos < a.length, a.length > 0 */
public static void iniciar(double[] a, int pos) {
    if (pos == a.length - 1) { a[pos] = Math.random(); }
    else {
        a[pos] = Math.random();
        iniciar(a, pos + 1);
    }
}
```

- Si es vol inicialitzar tot l'array, la crida inicial ha de ser: `iniciar(a, 0)`;

4. **Validació del disseny.**

- En el cas general, en cada crida la talla del problema decreix en 1 perquè `pos` s'incrementa en 1; així, en algun moment `pos` arribarà a ser `a.length-1`, aconseguint el cas base i finalitzant l'algorisme.
- En qualsevol dels dos casos, el valor de `pos` sempre compleix la precondició.

28/01/2020



PRG - Curs 2019/20



24

Recursió amb arrays: recorregut

- En un **recorregut** d'un array **a** s'han de tractar tots els elements de **a[ini..fi]**, amb **ini** i **fi** dins d'un determinat rang de **a**. Per tant, la seua resolució recursiva sol contemplar la següent anàlisi de casos **complementaris** i **excloents**:

- **Cas base:** **a[ini..fi]** és un array suficientment menut com per a que la solució siga directa. Depenent del problema, sol ser **ini == fi** (subarray amb un únic element), o **ini > fi** (subarray buit), o **ini >= fi** (subarray amb, com a molt, un element), ...
- **Cas general:** **a[ini..fi]** és més gran; cal tractar un element i fer un *subrecorregut* sobre una part de l'array més menuda, amb menys components:
 - tractar(**a[ini]**) i recórrer **a[ini+1..fi]** (recorregut **ascendent**),
 - tractar(**a[fi]**) i recórrer **a[ini..fi-1]** (recorregut **descendent**),
 - tractar(**a[ini]**), tractar(**a[fi]**) i recórrer **a[ini+1..fi-1]** (recorregut **combinat**),
 - i **altres** variants específiques de cada problema.

On tractar(**a[ini]**) (tractar(**a[fi]**)) és

l'operació a realitzar amb l'element que ocupa la posició **ini** (**fi**) de l'array.

Els índex sempre han de complir la precondició (estar dins del rang corresponent).

Els **índex que canvien** amb les crides hauran de ser **paràmetres** del mètode

28/01/2020



PRG - Curs 2019/20



25

Recursió amb arrays: recorregut

- Es sol definir un mètode públic homònim, anomenat **guia** o **llançadora**, que realitza la crida inicial, per tal d'ocultar l'estructura recursiva de l'array **a** que mostren els paràmetres **ini** i **fi** de la capçalera del mètode recursiu que ara es defineix privat.
- Per exemple, en el cas d'un **recorregut recursiu ascendent** de tots els elements d'un array **a**

```
/** PRECONDICIÓ: 0 <= ini <= a.length */
private static void recorrer(tipusBase[] a, int ini) {
    if (ini >= a.length) { tractarBuit(); }
    else {
        tractar(a[ini]);
        recorrer(a, ini + 1);
    }
}
```

```
public static void recorrer(tipusBase[] a) {
    recorrer(a, 0);
}
```

- Crida inicial:** **recorrer(a)**

28/01/2020



PRG - Curs 2019/20



26

Alguns exemples - Recorregut

1. **Enunciat del problema.** Determinar la suma de tots els elements d'un array d'enters $a[0..a.length-1]$ ($a.length \geq 0$).

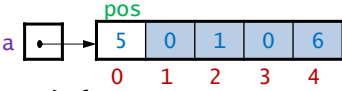
Recorregut recursiu ascendent

```
/** PRECONDICIÓ:  $0 \leq pos \leq a.length$  */
public static int sumaRecAsc(int[] a, int pos)
```

2. **Anàlisi de casos.**

- **Cas base:** Si $pos == a.length$, aleshores la suma de 0 elements és 0.
- **Cas general:** En un altre cas, la suma serà la suma de $a[pos]$ i la de $a[pos+1..a.length-1]$.

3. **Transcripció de l'algorisme a un mètode en Java.**



```
/** PRECONDICIÓ:  $0 \leq pos \leq a.length$  */
public static int sumaRecAsc(int[] a, int pos) {
    if (pos == a.length) { return 0; }
    else { return a[pos] + sumaRecAsc(a, pos + 1); }
}
```

- La crida inicial ha de ser: `int suma = sumaRecAsc(a, 0);`

4. **Validació del disseny.**

- En el cas general, en cada crida la talla del problema decreix en 1 perquè pos s'incrementa en 1; així, en algun moment pos arribarà a ser $a.length$, aconseguint el cas base i finalitzant l'algorisme.
- En qualsevol dels dos casos, el valor de pos sempre compleix la precondició.

28/01/2020



PRG - Curs 2019/20



27

Alguns exemples - Recorregut

1. **Enunciat del problema.** Determinar la suma de tots els elements d'un array d'enters $a[0..a.length-1]$ ($a.length \geq 0$).

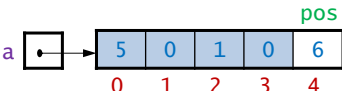
Recorregut recursiu descendent

```
/** PRECONDICIÓ:  $-1 \leq pos < a.length$  */
public static int sumaRecDesc(int[] a, int pos)
```

2. **Anàlisi de casos.**

- **Cas base:** Si $pos == -1$, aleshores la suma de 0 elements és 0.
- **Cas general:** En un altre cas, la suma serà la suma de $a[pos]$ i la de $a[0..pos-1]$.

3. **Transcripció de l'algorisme a un mètode en Java.**



```
/** PRECONDICIÓ:  $-1 \leq pos < a.length$  */
public static int sumaRecDesc(int[] a, int pos) {
    if (pos == -1) { return 0; }
    else { return a[pos] + sumaRecDesc(a, pos - 1); }
}
```

- La crida inicial ha de ser: `int suma = sumaRecDesc(a, a.length - 1);`

4. **Validació del disseny.**

- En el cas general, en cada crida la talla del problema decreix en 1 perquè pos es decrementa en 1; així, en algun moment pos arribarà a ser -1, aconseguint el cas base i finalitzant l'algorisme.
- En qualsevol dels dos casos, el valor de pos sempre compleix la precondició.

28/01/2020



PRG - Curs 2019/20



28

Alguns exemples - Recorregut

1. **Enunciat del problema.** Comptar les aparicions d'un enter x en un array d'enters $a[0..a.length-1]$ ($a.length \geq 0$).

Recorregut recursiu ascendent

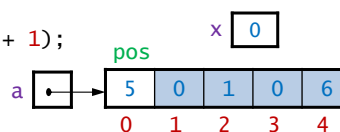
```
/** PRECONDICIÓ:  $0 \leq pos \leq a.length$  */
public static int comptarRecAsc(int[] a, int x, int pos)
```

2. **Anàlisi de casos.**

- **Cas base:** Si $pos == a.length$, aleshores no hi ha elements i torna 0.
- **Cas general:** En un altre cas, hi haurà que comptar les aparicions de x en $a[pos+1..a.length-1]$ i sumar 1 si $a[pos]$ és x .

3. **Transcripció de l'algorisme a un mètode en Java.**

```
/** PRECONDICIÓ:  $0 \leq pos \leq a.length$  */
public static int comptarRecAsc(int[] a, int x, int pos) {
    if (pos == a.length) { return 0; }
    else {
        int cont=comptarRecAsc(a, x, pos + 1);
        if (a[pos] != x) { return cont; }
        else { return 1 + cont; }
    }
}
```



- La crida inicial ha de ser: `int num = comptarRecAsc(a, x, 0);`

4. **Validació del disseny.** Similar a la validació del recorregut ascendent anterior.

28/01/2020



PRG - Curs 2019/20



29

Alguns exemples - Recorregut

1. **Enunciat del problema.** Comptar les aparicions d'un enter x en un array d'enters $a[0..a.length-1]$ ($a.length \geq 0$).

Recorregut recursiu descendent

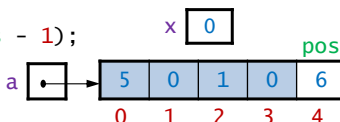
```
/** PRECONDICIÓ:  $-1 \leq pos < a.length$  */
public static int comptarRecDesc(int[] a, int x, int pos)
```

2. **Anàlisi de casos.**

- **Cas base:** Si $pos == -1$, aleshores no hi ha elements i torna 0.
- **Cas general:** En un altre cas, hi haurà que comptar les aparicions de x en $a[0..pos-1]$ i sumar 1 si $a[pos]$ és x .

3. **Transcripció de l'algorisme a un mètode en Java.**

```
/** PRECONDICIÓ:  $-1 \leq pos < a.length$  */
public static int comptarRecDesc(int[] a, int x, int pos) {
    if (pos == -1) { return 0; }
    else {
        int cont=comptarRecDesc(a, x, pos - 1);
        if (a[pos] != x) { return cont; }
        else { return 1 + cont; }
    }
}
```



- La crida inicial ha de ser: `int num = comptarRecDesc(a, x, a.length-1);`

4. **Validació del disseny.** Similar a la validació del recorregut descendent anterior.

28/01/2020



PRG - Curs 2019/20



30

Recursió amb arrays: cerca lineal o seqüencial

- En una **cerca lineal recursiva** per determinar si es compleix certa propietat en un array `a[ini..fi]`, amb `ini` i `fi` dins d'un determinat rang de `a`, igual que en un recorregut recursiu, la zona de l'array on fer la cerca es va reduint a subarrays més menuts.
- L'anàlisi de casos serà la següent:
 - Cas base:** subrray reduït al més menut possible on la cerca no pot tindre èxit.
 - Cas general:** queden elements suficients per comprovar si es compleix o no la propietat a cercar, de manera que:
 - cerca **ascendent**,
 - si `propietat(a[ini])` no té èxit, cercar en `a[ini+1..fi]`,
 - si sí té èxit, i independentment dels elements que queden per revisar, la cerca pot acabar.
 - cerca **descendent**,
 - si `propietat(a[fi])` no té èxit, cercar en `a[ini..fi-1]`,
 - si sí té èxit, i independentment dels elements que queden per revisar, la cerca pot acabar.
 - i **altres** variants específiques de cada problema.

Els **índex que canvien** amb les crides hauran de ser **paràmetres** del mètode

On `propietat(a[ini])` (`propietat(a[fi])`) comprova si l'element que ocupa la posició `ini` (`fi`) de l'array compleix la propietat enunciada.

28/01/2020



PRG - Curs 2019/20



31

Recursió amb arrays: cerca lineal o seqüencial

- Igual que en els recorreguts, es sol definir un mètode públic homònim, anomenat **guia o llançadora**, que realitza la crida inicial, per tal d'ocultar l'estructura recursiva de l'array `a` que mostren els paràmetres `ini` i `fi` de la capçalera del mètode recursiu que ara es defineix privat.
- Per exemple, en el cas d'una **cerca recursiva ascendent** que determina si algun element d'un array `a` compleix certa propietat:

```
/** PRECONDICIÓ: 0 <= ini <= a.length */
private static boolean cercar(tipusBase[] a, int ini) {
    if (ini < a.length) {
        if (propietat(a[ini])) { return true; }
        else { return cercar(a, ini + 1); }
    }
    else { return false; }
}
```

```
public static boolean cercar(tipusBase[] a) {
    return cercar(a, 0);
}
```

- Crida inicial:** `boolean res = cercar(a);`

28/01/2020



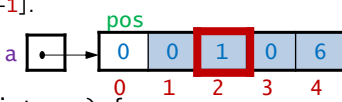
PRG - Curs 2019/20



32

Alguns exemples - Cerca

- Enunciat del problema.** Obtenir la posició del **primer** element **distint de zero** d'un array d'enters $a[0..a.length-1]$ ($a.length \geq 0$). Cerca lineal recursiva ascendent

```
/** PRECONDICIÓ: 0 <= pos <= a.length */
public static int trobarRecAsc(int[] a, int pos)
```
- Anàlisi de casos.**
 - **Cas base:** Si $pos == a.length$, aleshores no es troba i torna -1.
 - **Cas general:** En un altre cas, bé $a[pos]$ és el primer distint de zero i torna pos o bé no ho és i la cerca continua en $a[pos+1..a.length-1]$.
- Transcripció de l'algorisme a un mètode en Java.**


```
/** PRECONDICIÓ: 0 <= pos <= a.length */
public static int trobarRecAsc(int[] a, int pos) {
    if (pos == a.length) { return -1; }
    else if (a[pos] != 0) { return pos; }
    else { return trobarRecAsc(a, pos + 1); }
}

```

 - La **crida inicial** ha de ser: `int noZero = trobarRecAsc(a, 0);`
- Validació del disseny.**
 - En el cas general, en cada crida la grandària del problema decreix en 1 perquè pos s'incrementa en 1; així, en algun moment pos arribarà a ser $a.length$, aconseguint el cas base i finalitzant l'algorisme.
 - En qualsevol dels dos casos, el valor de pos sempre compleix la precondició.

28/01/2020



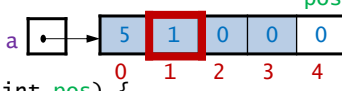
PRG - Curs 2019/20



33

Alguns exemples - Cerca

- Enunciat del problema.** Obtenir la posició del **darrer** element **distint de zero** d'un array d'enters $a[0..a.length-1]$ ($a.length \geq 0$). Cerca lineal recursiva descendent

```
/** PRECONDICIÓ: -1 <= pos < a.length */
public static int trobarRecDesc(int[] a, int pos)
```
- Anàlisi de casos.**
 - **Cas base:** Si $pos == -1$, aleshores no es troba i torna -1.
 - **Cas general:** En un altre cas, bé $a[pos]$ és el darrer distint de zero i torna pos o bé no ho és i la cerca continua en $a[0..pos-1]$.
- Transcripció de l'algorisme a un mètode en Java.**


```
/** PRECONDICIÓ: -1 <= pos < a.length */
public static int trobarRecDesc(int[] a, int pos) {
    if (pos == -1) { return -1; }
    else if (a[pos] != 0) { return pos; }
    else { return trobarRecDesc(a, pos - 1); }
}

```

 - La **crida inicial** ha de ser: `int noZero = trobarRecDesc(a, a.length - 1);`
- Validació del disseny.**
 - En el cas general, en cada crida la grandària del problema decreix en 1 perquè pos es decrementa en 1; així, en algun moment pos arribarà a ser -1, aconseguint el cas base i finalitzant l'algorisme.
 - En qualsevol dels dos casos, el valor de pos sempre compleix la precondició.

28/01/2020



PRG - Curs 2019/20



34

Cerca binària: estratègia

- S'estableixen dues estratègies principals a l'hora de fer una cerca en un array:
 - Cerca lineal o seqüencial:** es va reduint l'espai de cerca (quantitat d'informació sobre la que buscar) element a element. S'ha de realitzar una **cerca exhaustiva** (és a dir, començar des d'un extrem cap a l'altre, fins que es trobe o s'arribe al final sense trobar-lo). Es pot aplicar sempre, independentment de si les dades de l'array estan ordenades o no.
 - Cerca binària o dicotòmica:** es va reduint l'espai de cerca, eliminant cada vegada la meitat d'elements. Les dades de l'array han d'estar **ordenades** d'una forma coneguda (per exemple, de menor a major), però és més eficient que la cerca lineal.
- Enunciat del problema.** Obtenir la posició d'un enter x en un array d'enters $a[\text{ini}..\text{fi}]$ ordenat ascendentment, sent $0 \leq \text{ini} \leq \text{fi} < a.\text{length}$.
- Estratègia de l'algorisme:** examinar la posició central, $\text{meitat} = (\text{ini} + \text{fi}) / 2$, i decidir segons els tres casos possibles:
 - $a[\text{meitat}] == x$, aleshores la cerca acaba amb èxit i torna meitat .
 - $a[\text{meitat}] > x$, aleshores la cerca continua en $a[\text{ini}..\text{meitat}-1]$.
 - $a[\text{meitat}] < x$, aleshores la cerca continua en $a[\text{meitat}+1..\text{fi}]$.

Si x no es troba, torna -1 .

Cerca binària iterativa

- Enunciat del problema.** Obtenir la posició d'un enter x en un array d'enters $a[0..a.\text{length}-1]$ ordenat ascendentment.

```

/** PRECONDICIÓN: 0 ≤ ini ≤ a.length i -1 ≤ fi < a.length
 *                i a[ini..fi] ordenat ascendentment */
public static int cercaBinIter(int[] a, int x, int ini, int fi) {
    int meitat = 0;
    boolean trobat = false;
    while (ini ≤ fi && !trobat) {
        meitat = (ini + fi) / 2;
        if (x == a[meitat]) { trobat = true; }
        else if (x < a[meitat]) { fi = meitat - 1; }
        else { ini = meitat + 1; }
    }
    if (trobat) { return meitat; }
    else { return -1; }
}

```

- Fica un punt de ruptura en la línia del while i observa l'estat de les variables al llarg de l'execució del codi per a l'array $a = \{0, 3, 4, 6, 7, 8, 10, 17\}$, $\text{ini} = 0$, $\text{fi} = 7$ i $x = 20$. Quantes iteracions es fan?

Cerca binària recursiva

1. **Enunciat del problema.** Obtenir la posició d'un enter x en un array d'enters $a[0..a.length-1]$ ordenat ascendentment.

```
/** PRECONDICIÓ: 0 <= ini <= a.length i -1 <= fi < a.length
 *              i a[ini..fi] ordenat ascendentment */
public static int cercaBinRec(int[] a, int x, int ini, int fi)
```

2. **Anàlisi de casos.**

- **Cas base:** Si $ini > fi$, aleshores no es troba i torna -1 .
- **Cas general:** Si $ini \leq fi$, per al subarray $a[ini..fi]$,
 - Determinar la posició de l'element central $meitat = (ini + fi) / 2$.
 - Accedir a l'element central de $a[ini..fi]$, $a[meitat]$, comprovant que si:
 - $a[meitat] == x$, aleshores la cerca acaba amb èxit i torna $meitat$.
 - $a[meitat] > x$, aleshores la cerca continua en $a[ini..meitat-1]$.
 - $a[meitat] < x$, aleshores la cerca continua en $a[meitat+1..fi]$.

28/01/2020



PRG - Curs 2019/20



37

Cerca binària recursiva

3. **Transcripció de l'algorisme a un mètode en Java.**

```
/** PRECONDICIÓ: 0 <= ini <= a.length i -1 <= fi < a.length
 *              i a[ini..fi] ordenat ascendentment */
public static int cercaBinRec(int[] a, int x, int ini, int fi) {
    if (ini > fi) { return -1; }
    else {
        int meitat = (ini + fi) / 2;
        if (a[meitat] == x) { return meitat; }
        else if (a[meitat] > x) {
            return cercaBinRec(a, x, ini, meitat - 1);
        }
        else { return cercaBinRec(a, x, meitat + 1, fi); }
    }
}
```

- La crida inicial ha de ser: `int pos = cercaBinRec(a, x, 0, a.length - 1);`

4. **Validació del disseny.**

- En el cas general, en cada crida la grandària del problema es divideix per 2 (divisió entera); així, en algun moment $ini > fi$, aconseguint el cas base i finalitzant l'algorisme.
- En qualsevol dels dos casos, els valors de ini i fi sempre compleixen la precondició.
- Fica un punt de ruptura en la línia del cas base i en la primera línia del cas general i observa l'estat de les variables al llarg de l'execució del codi per a l'array $a = \{0, 3, 4, 6, 7, 8, 10, 17\}$, $ini = 0$, $fi = 7$ i $x = 20$. Quantes crides es fan en total?

Recursió versus iteració

- Tant recursió com iteració fan ús d'estructures de control: la **recursió** fa servir com instrucció principal una **instrucció de selecció** (condicional) i la **iteració** fa servir com instrucció principal una **instrucció de repetició** (bucle).
- Ambdues requereixen una **condició de terminació**: la **condició del cas base** en la **recursió** i la **condició de la guarda del bucle** en la **iteració**.
- Ambdues s'aproximen gradualment a la terminació: la **iteració** a mesura que s'apropa al compliment d'una condició i la **recursió** conforme es divideix el problema en altres més menuts, apropant-se també al compliment d'una condició, la del cas base.
- Ambdues poden tenir (per error) una **execució potencialment infinita**.
- Es pot demostrar que la solució algorísmica de qualsevol problema algorímicament resoluble, es pot expressar recursivament i també iterativament.
- En aquest sentit, es diu que **recursió** i **iteració** són **equivalents**, i per això, **alternatius**.

Recursió versus iteració

- En general, no és possible afirmar què és el més convenient o senzill.
- És freqüent trobar problemes per als quals la solució **iterativa** és **més senzilla d'estructurar** que la **recursiva**.
- A més, la **recursió** suposa, en general, **més càrrega computacional** (espai en memòria) que la **iteració**.
- En altres casos, la versió **recursiva** reflecteix de manera **més natural, concisa i elegant** la solució al problema que la versió **iterativa**, el que fa que siga més fàcil de depurar i entendre.
- Es pot concloure que **recursió** i **iteració**, a més d'**alternatius**, són **complementaris**.