- **EXERCICIS: Anàlisi d'algorismes iteratius**
  1. *Indicar quina és la grandària o talla del problema, així com l'expressió que la representa.*
  2. *Indicar si existeixen diferents instàncies significatives per al cost temporal de l'algorisme i identificar-les si és el cas.*
  3. *Triar una unitat de mesura per a l'estimació del cost (pas de programa, instrucció crítica) i, d'acord amb ella, obtenir una expressió matemática, el més precisa possible, del cost temporal de l'algorisme, a nivell global o en les instàncies més significatives si n'hi ha.*
  4. *Expressar el resultat anterior utilitzant notació asimptòtica.*

a) *Càlcul de la mitjana dels elements d'un array de reals (double).*

```
public static double mitjana(double[] a) {
    double suma = 0.0;
    for (int i = 0; i < a.length; i++) { suma += a[i]; }
    return suma / a.length;
}
```

a) Talla
b) Instàncies
c) Unitat Mesura
d) Expressió de Cost

b) *Obtenir el màxim d'un array d'enters (int).*

a) Talla
b) Instàncies
c) Unitat Mesura
d) Expressió de Cost

```java
public static int maxim(int[] v) {
    int maxim = v[0];
    for (int i = 1; i < v.length; i++) {
        if (v[i] > maxim) { maxim = v[i]; }
    }
    return maxim;
}
```

c) Producte escalar de dos arrays d'enters (de la mateixa grandària).

a) Talla
b) Instàncies
c) Unitat Mesura
d) Expressió de Cost

```
/** a.length = b.length */
public static int producteEscalar(int[] a, int[] b) {
    int producte = 0;
    for (int i = 0; i < a.length; i++) { producte += a[i] * b[i]; }
    return producte;
}
```

d) *Anàlisi de l'algorisme de cerca seqüencial d'un valor en un array d'enters*

```java
public static int cerca(int[] v, int e) {
    int i = 0;
    while (i < v.length && v[i] != e) { i++; }
    if (i < v.length) { return i; }
    else { return -1; }
}
```

a) Talla
b) Instàncies
c) Unitat Mesura
d) Expressió de Cost

e) *Donat un array d'enters, comprova que el valor de totes les posicions a partir de la 3ª posició (índex 2 de l'array) és igual a la suma dels valors de les dues posicions precedents:*

a) Talla
b) Instàncies
c) Unitat Mesura
d) Expressió de Cost

```
public static boolean compleixCondicio(int[] v) {
    int i = 2; boolean compleix = true;
    while (i < v.length && compleix) {
        compleix = v[i] == v[i - 1] + v[i - 2];
        i++;
    }
    return compleix;
}
```

Problema 7.

a) Talla
b) Instàncies
c) Unitat Mesura
d) Expressió de Cost

```
Algorisme 1:
for (int i = 0; i < n; i++) {
    w[i] = 0;
    for (int j = 0; j < i + 1; j++) {
        w[i] += v[j];
    }
}
```

```
Algorisme 2:
w[0] = v[0];
for (int i = 1; i < n; i++) {
    w[i] = w[i - 1] + v[i];
}
```

Els dos algorismes resolen el mateix problema:

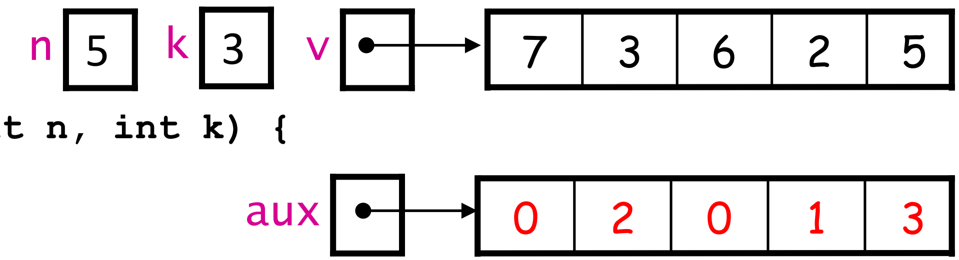
$\forall$ `i,0≤i<v.length,w[i]=` $\sum_{j=0}^{i}$ `v[j]`

f) *L'algorisme d'ordenació que segueix és una versió del de la bambolla que només ordena els **m** elements més menuts (1 ≤ **m** ≤ v.length) d'un array d'enters.*

- a) Talla
- b) Instàncies
- c) Unitat Mesura
- d) Expressió de Cost

```
public static void bambollaM(int[] v, int m) {
    for (int i = 0; i < m; i++) {
        for (int j = v.length - 1; j > i; j--) {
            if (v[j - 1] > v[j]) {
                int x = v[j - 1];
                v[j - 1] = v[j];
                v[j] = x;
            }
        }
    }
}
```

Problema 10. *El mètode següent troba el k-èssim element més menut d'un array de n enters.*




```java
public static int k-minim(int[] v, int n, int k) {
    int[] aux = new int[n];
    int cont = 1, min, posmin;
    while (cont <= k) {
        min = Integer.MAX_VALUE;  posmin = n;
        for (int i = 0; i < n; i++) {
            if (aux[i] == 0 && v[i] < min) {
                min = v[i]; posmin = i;
            }
        }
        aux[posmin] = cont;
        cont++;
    }
    return min;
}
```

a) Talla
b) Instàncies
c) Unitat Mesura
d) Expressió de Cost

Problema 9. *El següent mètode obté, a partir de cert valor enter m, la seqüència dels dígits que el composen (array v) i torna el nombre de xifres que té el número (i).*

a) Talla
b) Instàncies
c) Unitat Mesura
d) Expressió de Cost

```
public static int xifres(int[] v, int m) {
    int i = 0, q = m;
    while (q > 0) {
        i++;
        v[i] = q % 10;
        q = q / 10;
    }
    return i;
}
```

Problema 5. (i)

```
public static int prodEsc(int[] a, int[] b, int n) {
    int prod = 0;
    for (int i = 0; i < n; i++) {
        prod += a[i] * b[i];
    }
    return prod;
}
```

a) Talla
b) Instàncies
c) Unitat Mesura
d) Expressió de Cost

```
public static int[][] sumaMat(int[][] a, int[][] b, int n) {
    int[][] c = new int[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
    return c;
}
```

Problema 5. (ii)

a) Talla
b) Instàncies
c) Unitat Mesura
d) Expressió de Cost

```
public static int[][] prodMat(int[][] a, int[][] b, int n) {
    int[][] c = new int[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            c[i][j] = 0;
            for (int k = 0; k < n; k++) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    return c;
}
```

Problema 6. *El següent algorisme calcula la suma de les components d'un array de n naturals ($n \geq 1$).*

a) Talla
b) Instàncies
c) Unitat Mesura
d) Expressió de Cost

```
public static int suma(int[] v, int n) {
    int s = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < v[i]; j++) {
            s++;
        }
    }
    return s;
}
```

```
public static int trobarX(int[] a, int x) {
    int trobat = -1, i = 0, j = a.length-1;
    while (i <= j && trobat ==- 1) {
        if (a[i] == x) { trobat = i; }
        else if (a[j] == x) { trobat = j; }
        i++; j--;
    }
    return trobat;
}
```

a) Talla
b) Instàncies
c) Unitat Mesura
d) Expressió de Cost

- **EXERCICIS: Anàlisi d'algorismes recursius**
  1. *Indicar quina és la grandària o talla del problema, així com l'expressió que la representa.*
  2. *Indicar si existeixen diferents instàncies significatives per al cost temporal de l'algorisme i identificar-les si és el cas.*
  3. *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.*
  4. *Expressar el resultat anterior utilitzant notació asimptòtica.*

Problema 12.

a) Talla
b) Instàncies
c) Relacions Recurrència
d) Expressió de Cost

```java
/** v.length > 0, 0 <= i < v.length */
public static int maxim(int[] v, int i) {
    if (i == 0) { return v[i]; }
    else {
        int m = maxim(v, i - 1);
        if (m > v[i]) { return m; }
        else { return v[i]; }
    }
}
 /** Crida inicial: int max = maxim(v, v.length - 1); */
```

a) Talla
b) Instàncies
c) Relacions Recurrència
d) Expressió de Cost

```java
/** 0 <= ini <= fi <= v.length - 1 o ini > fi */

public static boolean capicua(String[] v, int ini, int fi) {
    if (ini >= fi) { return true; }
    else if (v[ini].equals(v[fi])) {
        return capicua(v, ini + 1, fi - 1);
    }
    else { return false; }
}

    /** Crida inicial: boolean b = capicua(v, 0, v.length - 1); */
```

Problema 13.

a) Talla
b) Instàncies
c) Relacions Recurrència
d) Expressió de Cost

```java
/** 0 <= i <= j <= v.length - 1 */

public static int suma(int[] v, int i, int j) {
    if (i == j) { return v[i]; }
    else {
        int m = (i + j) / 2;
        return suma(v, i, m) + suma(v, m + 1, j);
    }
}
   /** Crida inicial: int s = suma(v, 0, v.length - 1); */
```

Exercici. *Quan tots els elements d'un array apareixen en el seu ordre original al començament d'un altre array, es diu que el primer array és un prefixe del segon. En l'exemple següent, l'array a és un prefixe de l'array b: en el primer cas i no ho es en els següents*

$a = \{1,3,0,2,4\}$ $b = \{1,3,0,2,4,6,8,1\}$

$a = \{1,3,0,2,4\}$ $b = \{1,3,0,3,5,6,8,1\}$

$a = \{1,3,0,2,4\}$ $b = \{1,3,0,2\}$

Analitza el cost temporal de les versions iterativa i recursiva d'un mètode que, donats dos arrays d'enters a i b, torne com resultat el fet que l'array a siga o no prefixe de l'array b.

```java
public static boolean esPrefixeIt(int[] a, int[] b) {
    if (a.length > b.length) { return false; }
    else {
        int pos;
        for (pos = 0; pos < a.length && a[pos] == b[pos]; pos++);
        return pos == a.length;
    }
}

/** 0 <= pos <= a.length i a.length <= b.length */
private static boolean esPrefixeRec(int[] a, int[] b, int pos) {
    if (pos >= a.length) { return true; }
    else { return a[pos] == b[pos] && esPrefixeRec(a, b, pos + 1); }
}
public static boolean esPrefixe(int[] a, int[] b) {
    if (a.length > b.length) { return false; }
    else { return esPrefixeRec(a, b, 0); }
}
```

Problema 14.

a) Talla
b) Instàncies
c) Relacions Recurrència
d) Expressió de Cost

```java
/** 0 < a i 0 <= b */
public static int potencia(int a, int b) {
    if (b == 0) { return 1; }
    else if (b == 1) { return a; }
    else if (b % 2 == 0) { return potencia(a, b/2) * potencia(a, b/2); }
    else { return potencia(a, b/2) * potencia(a, b/2) * a; }
}
```

```java
/** 0 < a i 0 <= b */
public static int potencia(int a, int b) {
    if (b == 0) { return 1; }
    else if (b == 1) { return a; }
    else {
        int p = potencia(a, b / 2);
        if (b % 2 == 0) { return p * p; }
        else { return p * p * a; }
    }
}
```

Exemple 12.8: Torres d'Hanoi

```
public static void hanoi(int n, String orig, String dest, String aux) {
    if (n == 1) { moureDisc(orig, dest); }
    else {
        // Moure n-1 discos de "origen" a "auxiliar"
        // torre auxiliar és "destí"
        hanoi(n - 1, orig, aux, dest);
        // Moure l'últim disc de "origen" a "destí"
        moureDisc(orig, dest);
        // Moure n-1 discos de "auxiliar" a "destí"
        // torre auxiliar és "origen"
        hanoi(n - 1, aux, dest, orig);
    }
}
```

a) Talla
b) Instàncies
c) Relacions Recurrència
d) Expressió de Cost

a) Talla
b) Instàncies
c) Relacions Recurrència
d) Expressió de Cost

```java
/** m > 0 */
public static int suma(int m) {
    if (m < 10) { return m; }
    else { return m % 10 + suma(m / 10); }
}
```

a) Talla
b) Instàncies
c) Relacions Recurrència
d) Expressió de Cost

```java
/** n >= 0 */
public static int fibonacciRec(int n) {
    if (n > 1) {
        return fibonacciRec(n - 1) + fibonacciRec(n - 2);
    }
    else { return n; } // fibonacciRec(0)=0 i fibonacciRec(1)=1
}

public static int fibonacciIte (int n) {
    if (n <= 1) { return n; }
    int fib = 1;   int prevFib = 1;
    for (int i = 2; i < n; i++) {
        int temp = fib;  fib += prevFib;  prevFib = temp;
    }
    return fib;
}
```

```java
/** 0 <= pos <= v.length - 1 */
public static void selDirectaRec(int[] v, int pos) {
    if (pos < v.length - 1) {
        int min = v[pos + 1], pMin = pos + 1;
        for (int j = pos + 1; j < v.length; j++) {
            if (v[j] < min) { min = v[j];  pMin = j; }
        }
        if (v[pos] > v[pMin]) {
            min = v[pMin]; v[pMin] = v[pos]; v[pos] = min;
        }
        selDirectaRec(v, pos + 1);
    }
}
    /** Crida inicial:  selDirectaRec(v, 0); */
```

a) Talla
b) Instàncies
c) Relacions Recurrència
d) Expressió de Cost