Pràctica 3 PRG (2019/2020)

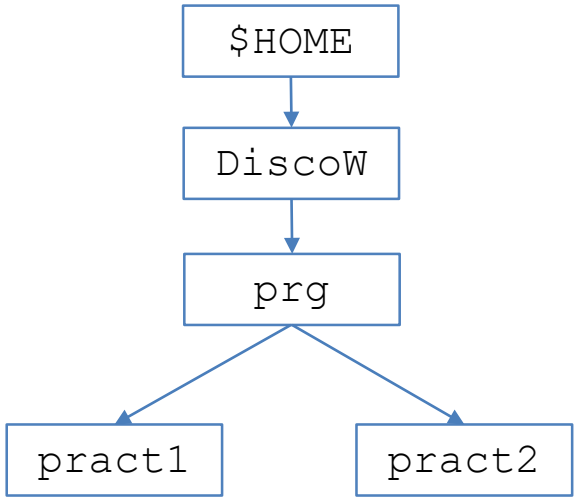
EFICIÈNCIA II

DURACIÓ 2 SESSIONS

IMPORTANT, UTILITZAR LA SESSIÓ PRG/EDA EN LLOC DE LA DE LINUX

Recordatorio:

- Si no has finalitzat la pràctica anterior recorda fer-ho abans del parcial.
- L'estructura de carpetes que hauries de tindre per a començar aquesta pràctica és:



PRÀCTICA 3: Com mesurar el temps?

1. Per a saber el temps que costa fer una activitat,
    1. Mire l'hora i me la apunto,
    2. Faig l'activitat
    3. Torne a mirar l'hora
    4. Calcule la diferència i ja està.

Però no és tan sencill

- *Si el meu rellotge nomes te manella dels segons i jo vull precisió de décimes de segon…*
- *Si vull saber quant costa fer eixa activitat sense necessitat d'establir un tamany inicial (per exemple, quant costa dibuixar un quadrat de costat 2 en la pissarra?.. i un de dos metres?)*
- *Si no sempre costa el mateix PER A UN MATEIX TAMANY (per exemple, si la pissarra es horitzontal, o vertical o en el sostre, o és una superfície llisa, o es irregular…)*
- *Si pot ocórrer que, a vegades i per motius inesperats, l'activitat cost més temps… (per exemple, em criden al telèfon mentre dibuixe)*
- *Si vull preveure quant em costarà per a qualsevol talla i en qualsevol circumstància…*

PRÀCTICA 3: Como mesurar el temps?

Per a saber el temps que costa fer una activitat, primer hauré fet un **anàlisi a priori** del cost

1. Per a una serie de tamanys creixents: Bucle
    1. Construeix experiments, un per a cada instància
    2. Per a cadascuna de las instàncies
        1. Repeteix varies vegades (si és necessari tinc còpies de cada experiment) Bucle
            1. Mire l'hora i me la apunte,
            2. Faig l'activitat
            3. Torne a mirar l'hora
            4. Calcule la diferència i l'acumule
        2. Divideix el temps acumulat pel nombre de repeticions
        3. Guarde el resultat medi d'eixa activitat en eixe tamany
    3. Guarde o mostre tots els resultats per a eixe tamany tabulats
2. Represente la taula resultant en una gràfica i
3. Ajuste els resultats amb la funció obtinguda per l'**anàlisi a priori** per a preveure el comportament per a qualsevol tamany

De debò cal fer-ho?
Sí, de debò

Índex d'Activitats

Pràctica 3. Mesura empírica de la complexitat computacional
(2 sessions)

- Crear paquets i posar en el seu lloc les classes java.
- Mirar com es calculen tempss de cerca
- Representar temps de cerca amb GNUPLOT
- Aproximar funcions amb GNUPLOT

- Implementar mètode per a emplenar l'array valors aleatoris

- Per parts:
  - Mesurar 1 cridada al mètode SelectionSort
  - Mesurar n repeticions de crides amb la mateixa talla
  - Mesurar n repeticions de crides per a talles creixents

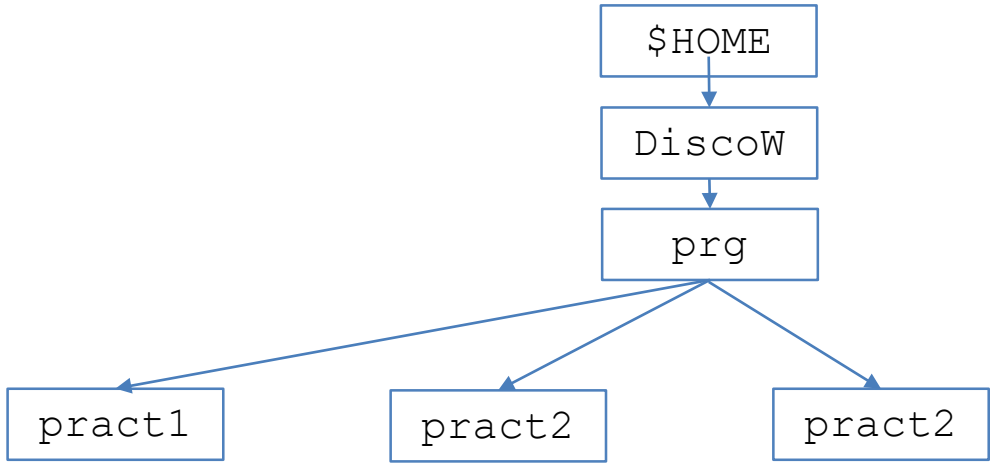
- Representar temps de SelectionSort amb GNUPLOT

- Implementar mètodes per a emplenar arrays ordenats

- Mesurar tempss de InsertionSort per a cada instància.
- Representar temps de InsertionSort per a cada instància

PRÀCTICA 3: Activitat 1

1. Crea dintre del projecte un paquet **pract3** per a la tercera pràctica.
2. Descarrega, de **PoliformaT** les classes de la pràctica 3 i afegeix-las al paquet **pract3**.




```
MeasuringLinearSearch.java
MeasuringSortingAlgorithms.java
MeasurableAlgorithms.java
```

PRÀCTICA 3: Activitats 2, 3, 4...

- Obtenim la taula de resultatos i la bolquem en un fitxer de text... Cóm?
  i. Compilem des de la línia de comandos tots els fitxers
  ii. Executem des de la línia de comandos la classe i bolquem el resultat en un fitxer de text
  iii. Ajustem la taula de dades resultant a una gràfica amb gnuplot (línia de comandos):
    i. Primer dibuixem dades, **per exemple**:

En valencià: Dibuixar el fitxer "linearSearch.out" usant les columnes 1 i 3 amb línies, …

En GnuPlotenc: `plot "linearSearch.out" using 1:3 with lines , …`

    ii. Després, ajustem a la funció de cost obtinguda a priori i podem tornar a dibuixar f(x) = a * x + b

En valencià: Ajustar f(x) el fitxer "linearSearch.out" amb les columnes1 i 3 obtenint a i b

En GnuPlotenc: `fit f(x) "linearSearch.out" using 1:3 via a,b`

    iii. Ahora podemos volver a dibujar juntos la función i los datos

```
plot f(x) with lines, "linearSearch.out" using 1:3 with points
```

PRÀCTICA 3: Anàlisi de Selecció Directa

Activitat 5: Afexeix un mètode a la classe `MeasuringSortingAlgorithms`

```
/** Omple els elements d'un array a d'int amb
 *  valors aleatoris entre 0 i a.length-1.
 *  @param a int[], l'array.
 */
private static void fillArrayRandom(int[] a) {
    // COMPLETAR

}
```

Per a cada posició de l'array a, utilitza el mètode **`random()`** de la classe **`Math`**
Per a operar fins a obtindre un enter des de **`0`** fins a **`a.length-1`** i ho emmagatzema en eixa posició. (Consulta la documentació de la classe **`Math`**)

PRÀCTICA 3: Activitats 6,7, 8

Activitat 6: temps de execució d'una única crida al mètode

Activitat 7: temps de execució per a una única talla donada

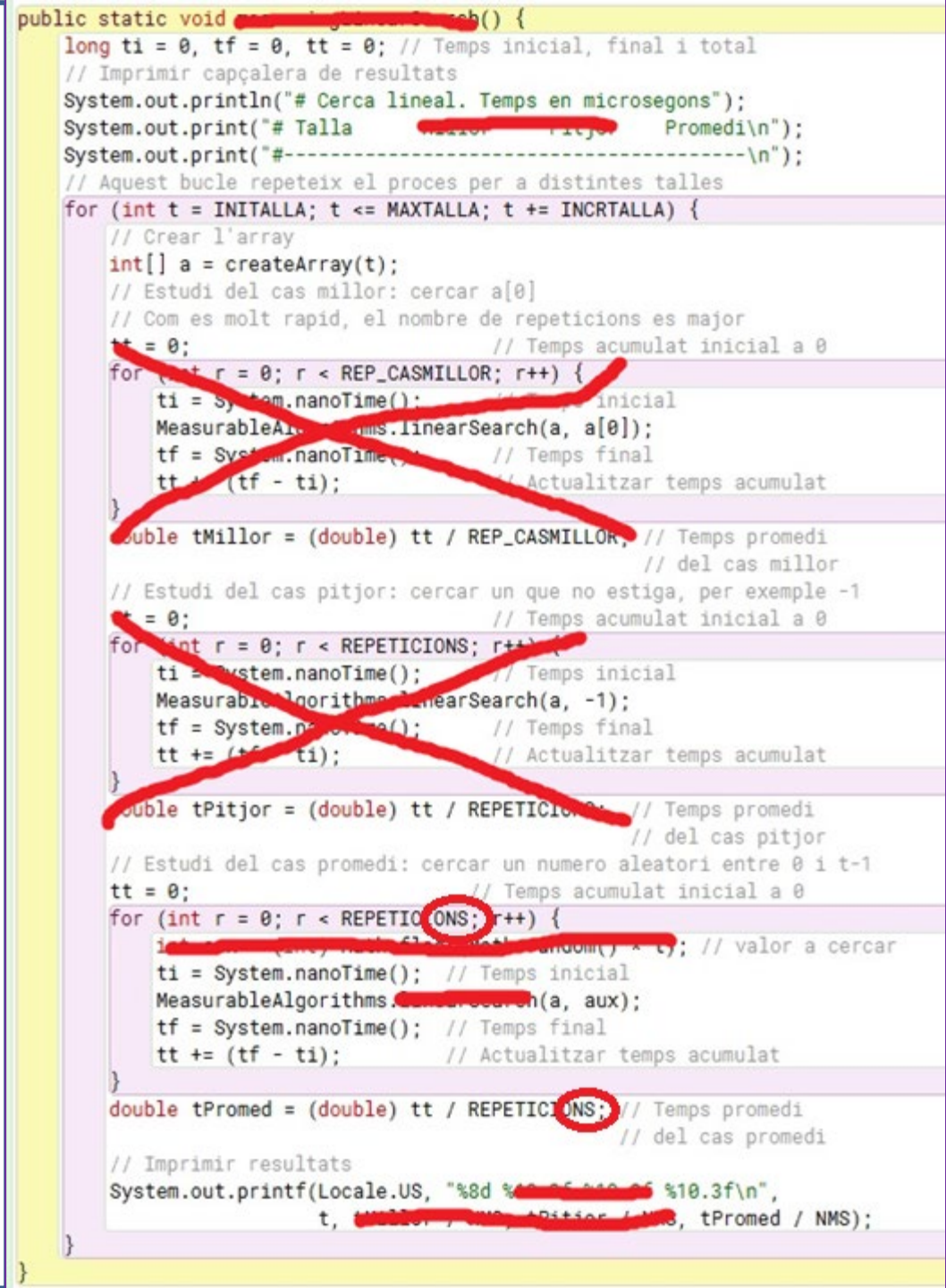
Activitat 8: temps de execució per a diferents talles

```java
public static void measuringLinearSearch() {
    long ti = 0, tf = 0, tt = 0; // Temps inicial, final i total
    // Imprimir capçalera de resultats
    System.out.println("# Cerca lineal. Temps en microsegons");
    System.out.print("# Talla      Millor     Pitjor    Promedi\n");
    System.out.print("#-----------------------------------------\n");
    // Aquest bucle repeteix el proces per a distintes talles
    for (int t = INITALLA; t <= MAXTALLA; t += INCRTALLA) {
        // Crear l'array
        int[] a = createArray(t);
        // Estudi del cas millor: cercar a[0]
        // Com es molt rapid, el nombre de repeticions es major
        tt = 0;                          // Temps acumulat inicial a 0
        for (int r = 0; r < REP_CASMILLOR; r++) {
            ti = System.nanoTime();      // Temps inicial
            MeasurableAlgorithms.linearSearch(a, a[0]);
            tf = System.nanoTime();      // Temps final
            tt += (tf - ti);             // Actualitzar temps acumulat
        }
        double tMillor = (double) tt / REP_CASMILLOR; // Temps promedi
                                                      // del cas millor
        // Estudi del cas pitjor: cercar un que no estiga, per exemple -1
        tt = 0;                          // Temps acumulat inicial a 0
        for (int r = 0; r < REPETICIONS; r++) {
            ti = System.nanoTime();      // Temps inicial
            MeasurableAlgorithms.linearSearch(a, -1);
            tf = System.nanoTime();      // Temps final
            tt += (tf - ti);             // Actualitzar temps acumulat
        }
        double tPitjor = (double) tt / REPETICIONS;  // Temps promedi
                                                     // del cas pitjor
        // Estudi del cas promedi: cercar un numero aleatori entre 0 i t-1
        tt = 0;                         // Temps acumulat inicial a 0
        for (int r = 0; r < REPETICIONS; r++) {
            int aux = (int) Math.floor(Math.random() * t); // valor a cercar
            ti = System.nanoTime();  // Temps inicial
            MeasurableAlgorithms.linearSearch(a, aux);
            tf = System.nanoTime();  // Temps final
            tt += (tf - ti);         // Actualitzar temps acumulat
        }
        double tPromed = (double) tt / REPETICIONS; // Temps promedi
                                                    // del cas promedi
        // Imprimir resultats
        System.out.printf(Locale.US, "%8d %10.3f %10.3f %10.3f\n",
                          t, tMillor / NMS, tPitjor / NMS, tPromed / NMS);
    }
}
```

Què tinc que modificar per a poder fer el mètode:

`measuringSelectionSort`

A més del nom?



Recorda que tens una variable pròpia per a les repeticions de cost quadràtic, **`REPETICIONESQ`**, utilitza-la tant en el bucle com per a dividir el temps total.

Ja tens canviada la capçalera del mètode i la capçalera de la taula per a solo mostrar el temps mitjà

No he de fer els bucles de cas millor i case pitjor

Abans de cada crida, en cada repetició, he de generar un nou array aleatori amb el mètode **`fillRandomArray`** per al fet que l'ordenació siga de debò mitjana, si no ho faig, la primera vegada sàrria sobre un array aleatori però les altres serà sobre un array ordenat

Finalment, només mostre per a cada talla el temps mitjà.

```
public static void [redacted]() {
    long ti = 0, tf = 0, tt = 0; // Temps inicial, final i total
    // Imprimir capçalera de resultats
    System.out.println("# Cerca lineal. Temps en microsegons");
    System.out.print("# Talla [redacted]    Promedi\n");
    System.out.print("#----------------------------------------\n");
    // Aquest bucle repeteix el proces per a distintes talles
    for (int t = INITALLA; t <= MAXTALLA; t += INCRTALLA) {
        // Crear l'array
        int[] a = createArray(t);
        // Estudi del cas millor: cercar a[0]
        // Com es molt rapid, el nombre de repeticions es major
        tt = 0;                          // Temps acumulat inicial a 0
        for (int r = 0; r < REP_CASMILLOR; r++) {
            ti = System.nanoTime();      // Temps inicial
            MeasurableAlgorithms.linearSearch(a, a[0]);
            tf = System.nanoTime();      // Temps final
            tt += (tf - ti);             // Actualitzar temps acumulat
        }
        double tMillor = (double) tt / REP_CASMILLOR; // Temps promedi
                                                      // del cas millor
        // Estudi del cas pitjor: cercar un que no estiga, per exemple -1
        tt = 0;                          // Temps acumulat inicial a 0
        for (int r = 0; r < REPETICIONS; r++) {
            ti = System.nanoTime();      // Temps inicial
            MeasurableAlgorithms.linearSearch(a, -1);
            tf = System.nanoTime();      // Temps final
            tt += (tf - ti);             // Actualitzar temps acumulat
        }
        double tPitjor = (double) tt / REPETICIONS; // Temps promedi
                                                    // del cas pitjor
        // Estudi del cas promedi: cercar un numero aleatori entre 0 i t-1
        tt = 0;                          // Temps acumulat inicial a 0
        for (int r = 0; r < REPETICIONS; r++) {
            int aux = (int) (StdRandom.random() * t); // valor a cercar
            ti = System.nanoTime();  // Temps inicial
            MeasurableAlgorithms.linearSearch(a, aux);
            tf = System.nanoTime();  // Temps final
            tt += (tf - ti);         // Actualitzar temps acumulat
        }
        double tPromed = (double) tt / REPETICIONS; // Temps promedi
                                                    // del cas promedi
        // Imprimir resultats
        System.out.printf(Locale.US, "%8d [redacted] %10.3f\n",
                          t, [redacted], tPromed / NMS);
    }
}
```

PRÀCTICA 3: Anàlisi de Selecció Directa

Activitat 9: Representació Gràfica de Resultats.

No només hi ha que representar-los en una gràfica, però també hi ha que ajustar a la funció que millor represente al cost de Selecció Directa… eixa fórmula ha de ser:

$$f(x) = a\,x^2 + b$$

o lo que es el mateix en "gnuplotenc":

```
f(x) = a * x * x + b
```

PRÀCTICA 3: Anàlisi d'Inserció Directa

Activitat 10: Afegeix dos mètodes a la classe `MeasuringSortingAlgorithms`

```java
/** Omple els elements d'un array a de forma creixent,
 *  amb valors des de 0 fins a.length-1.
 *  @param a int[], l'array.
 */
private static void fillArraySortedInAscendingOrder(int[] a) {
    // COMPLETAR
}

/** Omple els elements d'un array a de forma decreixent,
 *  amb valors des de a.length-1 fins 0.
 *  @param a int[], l'array.
 */
private static void fillArraySortedInDescendingOrder(int[] a) {
    // COMPLETAR
}
```

Omplir l'array A amb valors iguals al seu índex.

Omplir l'array A amb valors iguals a la llargària de l'array menys el seu índex.

PRÀCTICA 3: Anàlisi d'Inserció Directa

Activitat 11: Termina el mètode **`measuringInsertionSort()`**

```
public static void measuringInsertionSort() {
    long ti = 0, tf = 0, tt = 0; // Temps inicial, final i total
    // Imprimir capçalera de resultats
    System.out.println("# Insercio. Temps en microsegons");
    System.out.print("# Talla   Millor     Pitjor     Promedi \n");
    System.out.print("#---------------------------------------\n");

    // COMPLETAR
}
```

Hi ha que fer mesures per al cas millor, cas pitjor i mitjà.

Cadascuna ha de tindre el seu propi bucle de repeticions i el seu propi càlcul de temps.

El cas millor es molt ràpid, per això hi ha que utilitzar REPETICIONESL només en eixe cas.

En la resta utilitzar REPETICIONESQ

No oblidar utilitzar eixes constants per a dividir el temps total.

PRÀCTICA 3: Anàlisi d'Inserció Directa

Activitat 12: Representació Gràfica de Resultats.

No només hi ha que representar-los en una gràfica, però també hi ha que ajustar a la funció que millor represente al cost de Inserció Directa… eixa fórmula ha de ser per al cas pitjor i cas mitja:

$$f(x) = a\,x^2 + b$$

o lo que es el mateix en "gnuplotenc":

```
f(x) = a * x * x + b
```

Y per al cas millor:

$$f(x) = a\,x + b$$

o lo que es el mateix en "gnuplotenc":

```
g(x) = c * x + d
```

PRÀCTICA 3: Anàlisis de mergeSort

Activitat Addicional: Realitza l'anàlisi complet (representació gràfica inclosa) del algorisme `mergeSort`, en aquest cas, només hi ha cas mitja i la funció de cost resultant ha de ser:

$$f(x) = a\ x\ log(x) + b$$

o el que es el mateix en "gnuplotenc":

```
f(x) = a * x * log(x) + b
```