PRG (ETS d'Enginyeria Informàtica) - Curs 2019-2020

Pràctica 3. Mesura empírica de la complexitat computacional

(2 sessions)

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València

Índex

1 Context i treball previ

En el context acadèmic, esta pràctica correspon al *"Tema 2. Anàlisi d'algorismes. Eficiència. Ordenació."*. Els objectius són els següents:

- Introduir l'anàlisi d'algorismes al laboratori, fent servir un entorn real de programació: anàlisi empírica.
- Representar gràficament l'evolució de les mesures del temps d'execució de distints algorismes per a contrastar amb els resultats teòrics.
- Inferir funcions aproximades per definir el comportament temporal d'un algorisme.
- Usar els resultats empírics per fer comparacions i prediccions.

Abans de la sessió de laboratori, l'alumne/a ha de llegir el butlletí de pràctiques i tractar de resoldre (tant com siga possible) els problemes proposats. La pràctica es realitzarà durant 2 sessions.

2 Mesura del cost de la *cerca lineal*

En aquesta secció es presenta un problema complet d'anàlisi empírica. El problema és la cerca lineal, és a dir, la cerca d'un element en un array unidimensional que pot ser no estiga ordenat.

2.1 Definició del problema

El problema de la cerca lineal consisteix en, donat un array `a` d'elements d'un cert conjunt (per exemple, els números enters) i donat un element arbitrari `e` d'eixe conjunt (un número enter concret), tornar la primera posició de l'array que continga l'element `e`. Si `e` no es trobara en `a`, es tornaria un índex invàlid (per exemple, -1) com a resultat, indicant que l'element no s'ha trobat. El següent mètode resol aquest problema sobre un `array` d'`int`:

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

2.2 Anàlisi de casos

Quan s'enfrontem al problema d'analitzar un algorisme, el primer paràmetre que s'ha de definir és la *talla* del problema. En este cas, la talla del problema és clarament la grandària de l'array, ja que determina el nombre d'iteracions del seu bucle (per la condició `i < a.length`).

A més d'això, hem d'estudiar si l'algorisme presenta *instàncies significatives* o no. La cerca lineal presenta instàncies significatives, és a dir, *casos millor i pitjor*. Estos casos es defineixen per la segona condició del bucle (`a[i] != e`):

- **Cas millor**: quan l'element `e` es troba en la primera posició d'`a` (és a dir, `a[0] == e`), ja que en eixe cas el bucle no executaria cap de les seues iteracions previstes.
- **Cas pitjor**: quan l'element `e` no es troba en `a`, ja que per a verificar eixe fet s'ha d'explorar completament tot l'array.

Amb esta anàlisi prèvia, podem concloure que el cost en el cas millor és constant i que en el cas pitjor és lineal. Per tant, prenent com a talla del problema n = `a.length`, les fites asimptòtiques de l'algorisme són $\Omega(1)$ com a fita inferior i $O(n)$ com a fita superior, aleshores, direm que la funció de cost temporal pertany al conjunt resultant de la intersecció $T(n) \in \Omega(1) \cap O(n)$, donat que $\Omega(1)$ és el conjunt de funcions temporals amb límit inferior constant, i $O(n)$ és el conjunt de funcions temporals amb límit superior lineal.

El cas promedi és difícil de calcular. Es poden fer algunes assumpcions per simplificar els càlculs per este cas. Per exemple, podem assumir que l'element que es cerca sempre és dins de l'array, i que la probabilitat de trobar l'element en qualsevol posició és la mateixa. En este cas, la fita final del cas promedi ens diu que $T^{\mu}(n) \in \Theta(n)$.

2.3 Estructura d'un experiment de mesura

L'anàlisi empírica s'ha de fer després de l'anàlisi teòrica. Al dissenyar una anàlisi empírica, s'han de prendre en consideració els següents punts:

- **La mesura de temps ha de fer-se per diverses talles**: l'objectiu és obtindre una funció de cost que té com paràmetre la talla del problema; per tant, han d'emprar-se diverses talles per obtindre el perfil de la funció.
- **Les instàncies significatives han de mesurar-se separadament**: els casos millor, pitjor i promedi presenten habitualment diferents taxes de creixement i, per tant, diferents funcions de cost; aleshores, han de mesurar-se a diferents parts del codi.
- **Per traure resultats significatius s'han de prendre diverses mesures**: una única mesura per cada talla no és significativa, ja que pot veure's afectada per condicions de l'entorn (per exemple, l'execució d'altres processos a l'ordinador); per tant, per garantir resultats correctes s'han de prendre diverses mesures, de les que s'haurà d'obtindre el seu valor mitjà; aquest valor en terme mitjà es podrà considerar com un resultat significatiu de la mesura.

La mesura de temps d'un algorisme pot presentar-se com el següent procés:

1. Llegir el valor del temps actual del sistema i emmagatzemar-lo en t_I (temps inicial).
2. Executar l'algorisme (mètode).
3. Llegir el valor del temps actual del sistema i emmagatzemar-lo en t_F (temps final).
4. La diferència entre t_F i t_I és el temps que ha emprat l'algorisme en resoldre el problema.

Este procés es pot fer usant un rellotge extern, però és més precís usar el rellotge intern. Java aporta el mètode `static long nanoTime()`, en `java.lang.System`, que retorna el valor actual del temporitzador més precís del sistema en nanosegons (encara que la resolució pot ser menor, però com a mínim és de mil·lisegons). Per tant, el codi Java per mesurar temps és semblant a:

```
long ti, tf, tt;
ti = System.nanoTime();
// Crida al mètode que es vol temporitzar
tf = System.nanoTime();
tt = tf - ti;
```

on a la variable `tt` s'enregistrarà el temps que el mètode ha invertit en resoldre el problema. Esta mesura s'ha de fer moltes voltes i es calcula el temps promedi. Tanmateix, per casos extremadament ràpids (per exemple, el cas millor de la cerca lineal), és habitual incloure el bucle de repeticions dins de la mesura de temps, considerant menyspreable la sobrecàrrega del bucle. O també es pot repetir la mesura un nombre de vegades bastant gran (per exemple, en el cas millor de la cerca lineal, un nombre de vegades molt més gran que per als casos pitjor i promedi).

Finalment, les mesures de temps s'han de representar apropiadament. La forma usual és utilitzar una taula que mostre en cada fila la talla del problema i els temps mesurats per a cadascuna de les instàncies. Aquests temps han de venir expressats en alguna unitat de mesura (microsegons, mil·lisegons, etc.) que facilite la lectura dels valors, és aconsellable que la unitat aparega com un comentari a la taula. Cal notar, a més, que en tractar-se de valors mitjans, és raonable que vinguen expressats amb decimals. La forma típica d'aquesta taula és semblant a la següent:

```
# Cerca lineal. Temps en microsegons
# Talla      Millor     Pitjor    Promedi
#---------------------------------------
   10000      1.793      4.957      3.187
   20000      1.790      8.306      4.964
   30000      1.793     11.589      6.662
   40000      1.793     15.002      8.353
   50000      1.793     18.371     10.131
   60000      1.793     21.803     11.856
   .....
```

Activitat 1: creació del paquet `pract3` en el projecte *BlueJ* `prg`

Obre *BlueJ* en el projecte de treball de l'assignatura (`prg`) i crea un nou paquet `pract3` amb les classes `MeasurableAlgorithms.java` que conté, entre altres, el mètode `linearSearch(int[], int)` i `MeasuringLinearSearch.java` que implementa el codi que realitza l'anàlisi per a les distintes instàncies significatives d'aquest algorisme. Tens disponibles aquestes classes en `Recursos/Laboratori/Pràctica 3`, dins de `PoliformaT` de `PRG`.

Activitat 2: obtenció de temps de cerca

Executa la classe programa `MeasuringLinearSearch` per tal d'obtenir la taula de resultats de temps i guarda-la en un fitxer de nom `linearSearch.out`. Es recomana, per evitar sobrecàrrega, executar des de la línea de comandaments. Per exemple, per salvar els resultats al fitxer `linearSearch.out`, pots executar des de la línia de comandaments al directori `$HOME/DiscoW/prg`:

```
java -cp .  pract3.MeasuringLinearSearch > pract3/linearSearch.out
```

Pot ser que la JVM es queixe per que la versió del compilador és diferent de la del *BlueJ*. Aleshores, hauràs de recompilar el codi; per això, executa des de la línia de comandaments dins del directori `$HOME/DiscoW/prg` les següents ordres:

```
javac pract3/MeasurableAlgorithms.java
javac pract3/MeasuringLinearSearch.java
```

3 Representació gràfica: *gnuplot*

Els resultats numèrics usualment s'interpreten millor amb la seua representació gràfica. En esta secció mostrem com emprar l'eina *gnuplot* per obtindre representacions gràfiques dels resultats i per obtindre funcions de cost que s'aproximen als resultats empírics, les quals poden usar-se per comparar adequadament els algorismes i per obtindre prediccions. Per tal d'usar aquesta eina escriurem *gnuplot* en la línia d'ordres (terminal). *Gnuplot* accepta ordres amb modificadors; les ordres més importants són les següents:

- **plot**: dibuixa dades d'un fitxer o funcions predefinides; alguns modificadors són:
  - *fitxer*: s'especifica entre cometes dobles i diu on és el fitxer amb les dades a dibuixar; les línies del fitxer que comencen amb el símbol `#` s'ignoren.
  - **title** *text*: especifica el títol que ha de donar-se als punts (llegenda).
  - **using** *i:j*: especifica les columnes del fitxer de dades que s'empraran (i per l'eix X, j per l'eix Y).
  - **with** *format*: especifica el format de dibuix (els usuals són `lines`, `points` i `linespoints`).

- **replot**: amb el mateix significat i modificadors que *plot*, però sense esborrar la finestra gràfica, i així permet veure diverses gràfiques alhora; *replot* només redibuixa la finestra gràfica.

- **set xrange** *[inici:fi]*, **set yrange** *[inici:fi]*: fixa el rang de valors de l'eix X (respectivament Y) entre *inici* i *fi*.

- **set xtics** *interval*, **set ytics** *interval*: fixa l'interval entre marques a l'eix X (respectivament Y).

- **set xlabel** *text*, **set ylabel** *text*: fixa l'etiqueta de l'eix X (respectivament Y).

- **load** *fitxer*: carrega un fitxer de text amb ordres de *gnuplot* que són executades per *gnuplot*.

- **fit** *funció fitxer* **using** $i:j$ **via** *paràmetres*: permet ajustar una funció predefinida amb alguns paràmetres lliures a un conjunt de dades d'un fitxer. On:
  - *funció*: indica el nom de la funció a ajustar.
  - *fitxer*: indica el nom del fitxer amb les dades a ajustar (s'especifica entre cometes dobles).
  - **using** $i:j$: especifica les columnes del fitxer de dades que s'usaran (i per a l'eix X i j per a l'eix Y).
  - **via** *paràmetres*: especifica els paràmetres (separats per comes) de la funció a ajustar.

- **print** $f(x)$: mostra el valor d'una funció definida per a un valor de x concret. La funció $f(x)$ és la definida per poder ajustar els seus coeficients mitjançant el comandament fit descrit just abans.

Activitat 3: representació i anàlisi dels resultats empírics

Per tal de representar els temps de la taula emmagatzemada al fitxer `linearSearch.out`, cal que prepares el fitxer `linearSearch1.plot` amb el següent contingut:

`linearSearch1.plot`

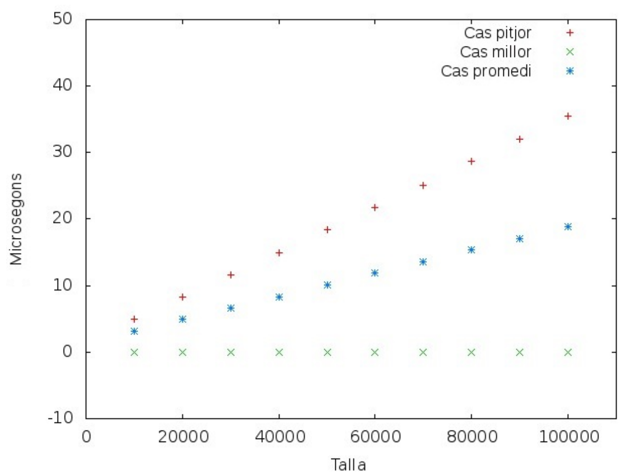
```
set xrange [0:110000]
set yrange [-10:]
set xtics 20000
set ytics 10
set xlabel "Size"
set ylabel "Microseconds"
set key left
set grid

plot "linearSearch.out" using 1:2 title "Best case" with points, \
     "linearSearch.out" using 1:3 title "Worst case" with points, \
     "linearSearch.out" using 1:4 title "Average case" with points
```

i aleshores arranca *gnuplot* i executa el fitxer mitjançant el comandament `load`:

```
gnuplot> load "linearSearch1.plot"
```

La imatge mostrada ha de ser semblant a la presentada en la següent figura:



Per salvar l'eixida gràfica a un fitxer .pdf has d'executar el fitxer `linearSearch2.plot` molt semblant a l'anterior però amb dos ordres noves per tal de canviar l'eixida i en lloc de mostrar el resultat en pantalla emmagatzemar-lo en un fitxer. Les ordres noves són `set term` i `set output`:

`linearSearch2.plot`

```
set xrange [0:110000]
set yrange [-10:]
set xtics 20000
set ytics 10
```

```
set xlabel "Size"
set ylabel "Microseconds"
set key left
set grid

set term pdf colour enhanced solid
set output "linearSearch.pdf"

plot "linearSearch.out" using 1:2 title "Best case" with points, \
     "linearSearch.out" using 1:3 title "Worst case" with points, \
     "linearSearch.out" using 1:4 title "Average case" with points
```

El fitxer `linearSearch.pdf` es desarà al directori actual amb la vista gràfica. Per redirigir l'eixida una altra vegada al terminal s'ha d'executar:

```
gnuplot> set term x11
gnuplot> set output
```

O simplement eixir i tornar a entrar a *gnuplot*. També es pot optar per executar el fitxer de comandaments de *gnuplot* de la següent manera des de la línia de comandaments:

```
gnuplot linearSearch2.plot
```

Activitat 4: aproximació de funcions als resultats empírics

gnuplot permet ajustar els valors de les columnes d'un fitxer de dades a una funció definida prèviament. Per exemple, si es sospita que determinats valors segueixen una distribució quadràtica es pot, mitjançant *gnuplot*, ajustar aquests valors a una paràbola. El resultat del procés d'ajust, seran els coeficients de la paràbola que millor s'aproxime a les dades de l'arxiu.

Per fer aquest tipus d'ajust s'utilitza l'ordre `fit` que, com s'ha dit, permet obtenir funcions aproximades que mostren el comportament d'un algorisme. Per exemple, com el cas pitjor i promedi de la cerca lineal presenten un cost lineal, és possible conèixer la diferència entre tots dos, ajustant prèviament cadascuna de les columnes de dades corresponents a una funció d'aquest tipus (lineal) per, després, comparar les funcions obtingudes.

Executa la següent seqüència de comandaments *gnuplot* per realitzar l'ajust de les dades del cas pitjor mitjançant una funció lineal:

```
gnuplot> f(x)=a*x+b
gnuplot> fit f(x) "linearSearch.out" using 1:3 via a,b
```

El resultat serà similar al que segueix:

```
...
Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 0.000339198      +/- 1.051e-06    (0.3098%)
b               = 1.47013          +/- 0.0652       (4.435%)
...
```

Ara fes el mateix però ajustant les dades del cas promedi mitjançant una altra funció lineal:

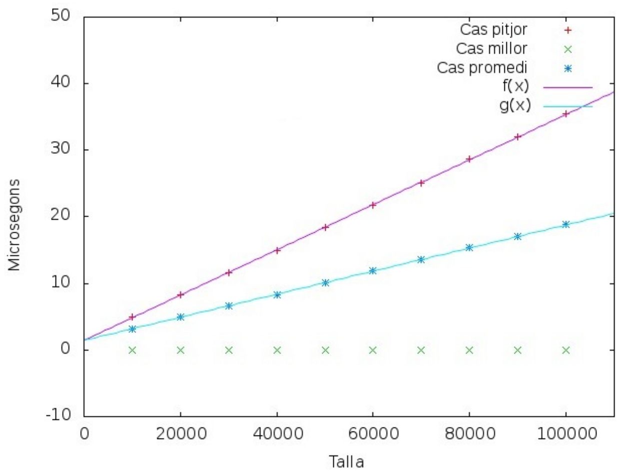
```
gnuplot> g(x)=c*x+d
gnuplot> fit g(x) "linearSearch.out" using 1:4 via c,d
...
Final set of parameters            Asymptotic Standard Error
=======================            ==========================
c                = 0.000173287      +/- 3.002e-07     (0.1733%)
d                = 1.4602           +/- 0.01863       (1.276%)
...
```

per tant, s'hi pot veure que la relació de creixement entre els casos pitjor i promedi és la relació entre les pendents de les funcions lineals corresponents, és a dir, $0.000339198/0.000173287 \approx 2$ per a les dades de la nostra taula. Per tant, es pot inferir que el cas pitjor és aproximadament el doble de lent que el cas promedi.

Les funcions d'ajust recentment estimades es poden representar gràficament junt als anteriors resultats:

```
gnuplot> replot f(x),g(x)
```

apreciant així que les dades experimentals segueixen amb gran fidelitat les funcions calculades:



L'estimació d'aquestes funcions aproximades es pot utilitzar per a fer prediccions. Per exemple, en el cas promedi, per a un array de mil milions d'enters (10^9), el temps requerit per a la seua execució segons aquestos càlculs s'estimaria de la següent manera:

$$g(x) = c * x + d = 0.000173287 * x + 1.4602$$

i substituint x pel valor de la talla del qual volem fer la predicció, 10^9,

$$g(10^9) = 0.000173287 * 10^9 + 1.4602 \approx 173288$$

microsegons, es a dir, prop d'un terç de segon, aproximadament.

Pots executar el fitxer `linearSearchFit.plot` per obtindre-ho tot directament:

linearSearchFit.plot

```
set xrange [0:110000]
set yrange [-10:]
set xtics 20000
set ytics 10
set xlabel "Size"
set ylabel "Microseconds"
set key left
set grid

#set term pdf colour enhanced solid
#set output "linearSearch.pdf"

f(x) = a*x+b
g(x) = c*x+d
fit f(x) "linearSearch.out" using 1:3 via a,b
fit g(x) "linearSearch.out" using 1:4 via c,d

plot "linearSearch.out" using 1:2 title "Best case" with points, \
     "linearSearch.out" using 1:3 title "Worst case" with points, \
     "linearSearch.out" using 1:4 title "Average case" with points, \
     f(x) with lines, g(x) with lines


print "f(", 10**9, ") = ", f(10**9)
print "g(", 10**9, ") = ", g(10**9)
```

4 Anàlisi del cost de l'ordenació per *selecció directa*

L'estratègia Selecció Directa ordena un array amb una complexitat temporal $\Theta(n^2)$, sent n el nombre d'elements de l'array, i no presenta instàncies significatives per al cost. Per això, n'hi ha prou en realitzar l'estudi del cost independentment del contingut de l'array, considerant aleshores que les dades del mateix es generen aleatòriament.

Activitat 5: mètode per a generar un array amb valors aleatoris

Afegeix al teu paquet `pract3` la classe `MeasuringSortingAlgorithms.java` disponible en la `PoliformaT` de `PRG`. Abans de començar amb l'anàlisi del cost del mètode `selectionSort(int[])` que implementa aquesta estratègia d'ordenació, definit en la classe `MeasurableAlgorithms`, has d'escriure en la classe `MeasuringSortingAlgorithms` un mètode per a omplir un array donat amb enters generats de forma aleatòria d'acord al següent perfil:

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

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

Completa el mètode `measuringSelectionSort()` de la classe `MeasuringSortingAlgorithms` amb les instruccions necessàries per tal de:

1. Crear un array `a` de talla 100, utilitzant el mètode `createArray(int)`.
2. Omplir l'array amb dades aleatòries utilitzant el mètode `fillArrayRandom(int[])`.
3. Cridar al mètode `System.nanoTime()` per a obtenir en una variable `ti` (de tipus `long`) el valor del rellotge (en nanosegons) abans de cridar al mètode que volem temporitzar.
4. Cridar al mètode `selectionSort(int[])` de la classe `MeasurableAlgorithms` per a ordenar l'array `a`.
5. Tornar a cridar al mètode `System.nanoTime()` per a obtenir en la variable `tf` el valor del rellotge una vegada acabada l'ordenació.
6. Calcular la diferència de temps `(tf - ti)` per a saber el temps que ha requerit el mètode `selectionSort(int[])` per a ordenar l'array `a`.
7. Mostrar per pantalla una filera de dades amb la talla de l'array i el temps en microsegons.

Activitat 7: temps d'execució per a una única talla donada

Com s'ha comentat anteriorment, prendre una única mesura per estimar el cost d'un mètode per a una determinada talla no és un procediment adequat, ja que aquesta mesura pot veure's afectada per condicions de l'entorn. Així, per garantir resultats correctes s'ha de repetir la mesura i després calcular la mitjana aritmètica sobre el nombre de mesures que s'hagen pres.

Defineix la constant `REPETICIONSQ` (amb valor 200) a la classe `MeasuringSortingAlgorithms` i completa el mètode `measuringSelectionSort()` d'aquesta classe amb un bucle per a repetir aquest nombre de vegades el bloc d'instruccions que ja tenim escrit de l'activitat anterior i calcula el temps promedi. Adona't que per a minimitzar la dependència del nostre experiment respecte d'una instància particular de l'array omplert de forma aleatòria, resulta convenient que **en cada repetició s'omple de nou l'array amb valors aleatoris diferents**[1]. De no ser així, fixa't que a més a més l'array a ordenar ja estaria ordenat a priori a partir de la primera repetició, fet que no invalidaria l'experiment formalment però estadísticament no seria significatiu. Calcula el temps promedi per repetició (en μs) i mostra'l per pantalla junt a la talla de l'array.

Activitat 8: temps d'execució per a diferents talles

Defineix en la classe `MeasuringSortingAlgorithms` les constants `INITALLA`, `MAXTALLA` i `INCRTALLA` per a representar respectivament el valor de la talla més menut a considerar (1000), el valor més gran (10000) i l'increment de talla (1000). Completa, en aquesta classe, el mètode `measuringSelectionSort()`, afegint un bucle per a repetir el càlcul del temps d'execució per a talles des de `INITALLA` fins `MAXTALLA` amb increments de `INCRTALLA`; és a dir, per a talles 1000, 2000, 3000, ..., 10000. El mètode ha de mostrar per pantalla una taula com la que segueix en la que el temps es done en microsegons:

[1] Així, l'array es crea una única vegada però s'omple de nou en cada repetició.

```
# Selecció directa. Temps en microsegons
#  Talla    Promedi
# -----------------
    1000    403.346
    2000   1348.255
    3000   3061.948
   ...
```

Activitat 9: representació gràfica dels resultats

- Executa la classe programa `MeasuringSortingAlgorithms`, seleccionant la opció *Seleccio*, guarda la taula resultat en un fitxer i, utilitzant *gnuplot*, mostra els resultats en una gràfica en la que l'eix X represente la talla i l'eix Y el temps d'ordenació en microsegons. Tingues en compte que les característiques del gràfic a representar han de tindre una correspondència amb les dades de la taula.
- Ajusta els resultats obtinguts a una funció quadràtica (`f(x)=a*x*x+b*x+c`), observant els valors dels paràmetres d'ajust.
- Torna a construir la gràfica mostrant, a més dels punts experimentals, la funció d'ajust. Etiqueta adequadament els eixos, els punts, i la funció d'ajust, afegeix un títol a la gràfica i guarda-la en un fitxer .pdf.
- Utilitza la funció d'ajust per a predir quin seria el temps necessari per a ordenar amb el mètode `selectionSort(int[])` un array de 800000 enters.

5 Anàlisi del cost de l'ordenació per *inserció directa*

L'estratègia Inserció Directa ordena un array amb una complexitat temporal $\Omega(n)$ i $O(n^2)$, sent n el nombre d'elements de l'array, presentant instàncies significatives per al cost: el cas millor quan l'array ja està ordenat (de forma creixent), i el cas pitjor quan l'array també està ordenat, però a l'inrevés, es a dir, de manera decreixent. Per això, és necessari realitzar l'estudi del comportament del mètode en el cas millor (amb arrays ja ordenats), en el cas pitjor (amb arrays ordenats de forma decreixent) i en el cas promedi (amb arrays generats aleatòriament).

Activitat 10: creació d'arrays ordenats

En la classe `MeasurableAlgorithms` està definit el mètode `insertionSort(int[])` que implementa aquesta estratègia d'ordenació. Per a poder analitzar aquest mètode s'han d'escriure en la classe `MeasuringSortingAlgorithms` dos mètodes per a omplir arrays d'enters, de manera que el seu contingut estiga ordenat, respectivament, de forma creixent (amb valors des de `0` fins `a.length-1`) i decreixent (amb valors des de `a.length-1` fins `0`); els seus perfils serien:

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

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

Activitat 11: anàlisi empírica del cost del mètode `insertionSort`

Completa el mètode `measuringInsertionSort()` de la classe `MeasuringSortingAlgorithms` per a estudiar el comportament del mètode `insertionSort(int[])` per als casos millor, pitjor i promedi, talles des de `INITALLA` fins `MAXTALLA` amb increments de `INCRTALLA`; és a dir, per a talles 1000, 2000, 3000, ..., 10000. Per als casos pitjor i promedi, la mesura es repetirà `REPETICIONSQ` vegades (200). En el cas millor (igual que en la cerca lineal), la mesura s'ha de repetir un nombre major de vegades: `REPETICIONSL` (amb valor 20000). El mètode ha de mostrar per pantalla una taula com la que segueix:

```
# Inserció directa. Temps en microsegons
#  Talla   Millor     Pitjor     Promedi
# --------------------------------------
    1000    0.025     422.532     134.647
    2000    0.029     848.167     405.849
    3000    0.040    1904.827     919.622
   ...
```

Fixa't que per a l'algorisme d'inserció directa, la recomanació feta a l'activitat 7, sobre que **en cada repetició s'havia de regenerar un nou array aleatori diferent**, en aquest cas resulta absolutament obligatòria. Si no, els resultats en els casos pitjor i promedi no serien vàlids ja que l'array ja estaria ordenat (sent el cas millor) en les restants repeticions. Recorda crear l'array una única vegada per a cada talla (no per a cada repetició!!!).

Activitat 12: representació gràfica dels resultats

- Executa la classe programa `MeasuringSortingAlgorithms`, seleccionant la opció *Insercio*, guarda la taula resultat en un fitxer i, utilitzant *gnuplot*, mostra els resultats en una gràfica en la que l'eix X represente la talla i l'eix Y el temps d'ordenació en microsegons. Has de dibuixar els punts experimentals obtinguts per als tres casos. Tingues en compte que les característiques del gràfic a representar han de tindre una correspondència amb les dades de la taula.
- Ajusta els resultats obtinguts a les funcions previstes de l'anàlisi teòrica (cas millor a funció lineal, casos pitjor i promedi a funcions quadràtiques) observant els valors dels paràmetres d'ajust.
- Torna a construir la gràfica mostrant, a més dels punts experimentals, les tres funcions d'ajust. Etiqueta adequadament els eixos, els punts, i les funcions d'ajust, afegeix un títol a la gràfica i guarda-la en un fitxer .pdf.

- Utilitza les funcions d'ajust per a predir quin seria el temps necessari per a ordenar un array de 800000 enters mitjançant el mètode `insertionSort(int[])` si: a) l'array ja està ordenat en ordre ascendent; b) si l'array també està ordenat, però en sentit decreixent; i c) per a un array amb valors aleatoris.

6 Anàlisi del cost de l'ordenació per *mescla o MergeSort* (activitat addicional)

Completa el mètode `measuringMergeSort()` de la classe `MeasuringSortingAlgorithms` per tal d'estudiar el comportament del mètode `mergeSort(int[], int, int)` definit en la classe `MeasuringSortingAlgorithms`. En aquest exercici, et suggerim que faces servir talles que siguen potències de 2. Per a això, pots definir en la classe `MeasuringSortingAlgorithms` les constants `INITALLA_MERGE` i `MAXTALLA_MERGE` per representar respectivament el valor de la talla més menuda a considerar (2^{10}) i el valor més gran (2^{19}), i que l'increment de talla siga: `talla *= 2`. El nombre de crides al mètode d'ordenació per obtenir valors significatius, pot ser el mateix que en l'anàlisi dels altres dos algoritmes d'ordenació (200).

7 Avaluació

Aquesta pràctica forma part del primer bloc de pràctiques que serà avaluat en el primer parcial de PRG. El valor d'eixe bloc és d'un 50% respecte al total de les pràctiques. El valor percentual de les pràctiques en l'assignatura és d'un 25% de la nota final.