Esiee-Paris - unité d'algorithmique - Premier projet - version 3 : 26/01/2021

Différence avec version 2 : l'invariant de l'exercice 1 est donné.

Janvier 2021 – R. Natowicz, I. Alame, A. Çela, D. Courivaud, D. Garrigue

1. QuickSelect.

Soit $T[0:n]$ un tableau d'entiers. Si T est croissant, alors sa p-ème valeur, $0 \leq p < n$, est $T[p]$.

Exemple : avec $n = 4$ et $T[0:n] = [5, 8, 3, 5]$ les $p = 0, 1, 2$ et 3-èmes valeurs sont respectivement 3, 5, 5, et 8. Nous avons donc un premier algorithme de calcul de la p-ème valeur de T : " trier T puis retourner $T[p]$. " Sa complexité est en $\Theta(n \log_2 n)$. Peut-on faire mieux ? Oui !

Nous devons au très grand C.A.R. Hoare l'algorithme QuickSelect qui retourne la p-ème valeur du sous-tableau $T[i:j]$ sans trier ce sous-tableau. En pratique son temps de calcul est en $\Theta(n)$. Son pire cas, de probabilité presque nulle, est en $\Theta(n^2)$. Cet algorithme repose sur la décomposition récursive ci-dessous, dans laquelle $m = j - i$ est la taille du sous-tableau $T[i:j]$.

- Si $m = 1$, p est nécessairement égal à 0. La p-ème valeur de $T[i:j]$ est $T[i]$;
- si $m > 1$, p est nécessairement dans l'intervalle $[0:m]$.
  Calculons une permutation de $T[i:j]$ vérifiant $T[i:k] \leq T[k:k+1] < T[k+1:j]$ (voir la procédure `segmenter` du cours.) Alors :
  - si $p + i \in [i:k]$, quelle est la p-ème valeur de $T[i:j]$?
  - même question si $p + i = k$;
  - même question si $p + i \in [k+1:j]$.

– Écrire la fonction récursive `int qsel(int p, int[] T, int i, int j)`, $0 \leq p < j - i$, qui retourne la p-ème valeur de $T[i:j]$.

– Écrire une version itérative. En cours d'exécution de l'algorithme, la p-ème valeur de $T[0:n]$ se trouve dans un certain sous-tableau $T[i:j]$, et cette p-ème valeur est la p'-ème valeur de ce sous-tableau $T[i:j]$. Nous avons l'invariant $I(p', i, j)$: "la p-ème valeur de $T[0:n]$ est la p'-ème valeur de $T[i:j]$."

1) Initialisation : pour quelles valeurs p', i, et j êtes-vous sûr que la propriété $I(p', i, j)$ est vraie ?

2) Condition d'arrêt : la propriété $I(p', i, j)$ étant vraie, pour quelles valeurs p', i, et j connaissez-vous la p-ème valeur de $T[0:n]$ (et quelle est cette valeur ?)

3) Progression : la propriété $I(p', i, j)$ étant vraie et la condition d'arrêt fausse, segmentons le sous-tableau $T[i:j]$. Alors, nous avons : $I(p', i, j)$ et arrêt et $T[i:k] \leq T[k:k+1] < T[k+1:j]$. Trois cas peuvent se présenter :

1. $p' + i \in [i:k]$.
2. $p' + i \in [k:k+1]$.
3. $p' + i \in [k+1:j]$

Dans chacun de ces trois cas : pour quelles valeurs p', i' et j' pouvez-vous dire que la propriété $I(p', i', j')$ est vraie ? En déduire le programme ci-dessous.

```
static int qselIteratif(int p, int[] T){ int n = T.length; // 0 <= p < n
int pprime = ...,  i =...,  j = ... ; // I(p', i, j)
  while (!(....)) { // I(p',i,j) et non arrêt
     int k = segmenter(T, i, j);
     int pppi = pprime + i ;
     if ( i < = pppi && pppi < k ) // I(..., ..., ...)
       { ..... } // I(p', i, j)
     else
     if ( k <= pppi && pppi < k+1 ) // I(..., ..., ...)
       { ........ } // I(p', i, j)
     else //  k+1 <= pppi && pppi < j ) // I(..., ..., ...)
       { ........} // I(p', i, j)
  // I(p', i, j) et arrêt, donc la p-ème valeur de T[ 0 : n ] est ...
  return ...;
}
```

Remarque : dans la vie courante, on ne parle pas de la 0-ème valeur mais plutôt de la 1ère valeur, de la 2ème valeur, etc., et de la n-ème valeur de $T[0:n]$. Pour se conformer à cet usage, on définit la fonction `quickSelect` ci-dessous :

```
int quickSelect(int p, int[] T) { // retourne la p-ème valeur de T[0:n], avec 1 <= p <= n
   int n = T.length ;
   return qsel(p-1, T, 0, n);
}
```

et de même pour la version itérative.

Médiane : la valeur médiane de $T[0:n]$ est la valeur de T qui permet de "partager" ses valeurs en deux sous-ensembles de tailles égales : autant de valeurs supérieures ou égales à la médiane que de valeurs inférieures ou égales. La médiane est une valeur statistique très importante car, contrairement à la moyenne, la médiane est insensible aux valeurs extrêmes (également appelées *outliers*[1].) La médiane est donc plus *robuste* que la moyenne.

```
int mediane(int[] T) { // retourne la valeur médiane de T
   int n = T.length ;
   return quickSelect(1+(n-1)/2, T) ;
   // Ou, si l'on préfère : return qselRecursif((n-1)/2, T, 0, T.length) ;
}
```

2. Remplir un sac.

La dernière partie du cours sera consacrée à la programmation dynamique (voir la description de l'unité.) Cette méthode algorithmique traite de problèmes d'optimisation. Un problème d'optimisation est le calcul de la valeur optimale (maximum ou minimum) d'un ensemble.

Exemples de problèmes que nous traiterons dans cette partie du cours :

- Vous disposez d'un certain nombre d'heures de révisions pour préparer les contrôles de fin de semestre. Comment les répartir sur les différentes unités pour obtenir une somme de notes maximum ?

  Il est ici question de l'ensemble des répartitions possibles des heures de révision sur les unités qui feront l'objet des contrôles de fin de semestre. À chaque répartition correspond une somme de notes. La somme des notes est la valeur de la répartition. On cherche une répartition de valeur maximum.

- Un magasin dispose d'une planche de longueur L qu'il peut débiter en tronçons de longueurs quelconques, comprises entre 1 et L (dans ce dernier cas, la planche est vendue en l'état.) Pour toute longueur l, $1 \leq l \leq L$, vous connaissez le bénéfice de la vente d'un tronçon de longueur l. On demande de calculer un débitage de la planche, de bénéfice total maximum.

  Il est ici question de tous les débitages possibles de la planche. La valeur d'un débitage est la somme des bénéfices des tronçons débités. On cherche un débitage de valeur maximum.

- Un ensemble d'objets est à disposition, chaque objet i a une taille t_i et une valeur v_i et vous disposez d'un sac d'une certaine contenance. On demande de calculer un sous-ensemble d'objets dont la somme des valeurs est maximum, sous la contrainte que ces objets puissent entrer dans le sac (la somme de leurs tailles est inférieure ou égale à la contenance du sac.)

  On parle ici de l'ensemble des sous-ensembles d'objets dont la somme des tailles est inférieure ou égale à la contenance du sac. La valeur d'un sac est la somme des valeurs des objets qu'il contient. On cherche un sac de valeur maximum.

La programmation dynamique garantit l'optimalité des solutions trouvées : la somme des notes, le bénéfice total du débitage, la valeur du sac, calculés par programmation dynamique, sont maximum.

Les méthodes "gloutonnes", quant à elles, construisent une solution en procédant par améliorations "locales." Les solutions gloutonnes ne retournent pas, dans le cas général[2], les solutions optimales, mais ce sont les premières auxquelles on pense...

Exemples de méthodes gloutonnes pour les problèmes ci-dessus :

- Pour chaque unité, ajouter une heure de révision ne fait pas baisser la note au contrôle, et parfois, cet ajout la fait même augmenter... L'augmentation éventuelle de la note d'une unité est fonction du temps que l'on consacre déjà à la réviser. En effet : ajouter une heure de révision à une unité à laquelle on consacre déjà 40 heures de révision aura moins d'influence sur la note qu'ajouter une heure de révision lorsqu'on n'y consacre que 5 heures. Ceci dépend bien sûr des unités en question.

  L'approche gloutonne alloue une heure supplémentaire à l'une des unités dont elle fera le plus augmenter la note. Les heures sont ainsi distribuées, une à une, jusqu'à épuisement des heures de révision disponibles.

- Soit l^*, $l^* \leq r = L$, la longueur du tronçon de bénéfice maximum. Couper un tronçon de cette longueur l^*. Il reste une planche de longueur $r = L - l^*$. Continuer de la sorte jusqu'à ce que $r = 0$.

- Soit i^* un objet de taille inférieure ou égale à la contenance $r = c$ de votre sac, objet qui, parmi ceux qui ne sont pas déjà dans le sac, est de valeur maximum. Le mettre dans le sac. Il reste une contenance $r = c - t_{i^*}$. Continuer de la sorte jusqu'à ce qu'aucun objet ne puisse entrer dans le sac de contenance r.

  On peut également définir la "densité de valeur" $d_i = \frac{v_i}{t_i}$ de l'objet i et procéder par densités de valeurs décroissantes (plutôt que par valeurs décroissantes.)

[1] `https://dictionary.cambridge.org/fr/dictionnaire/anglais/outlier`.

[2] Certains problèmes d'optimisation se résolvent par une méthode gloutonne, mais ces problèmes sont rares.

On demande de mettre en oeuvre la stratégie gloutonne de remplissage d'un sac par valeurs décroissantes, et par densités de valeurs décroissantes.

Un objet est la donnée de ses numéro, taille, valeur et densité de valeur. Pour le représenter on définit une classe : `class objet{int i, v, t; float d;}` (numéro, valeur, taille et densité de valeur). Les n objets sont dans un tableau $T[0:n]$ qui aura été créé par l'instruction `objet[] T = new objet[n]`.

1. Écrire une fonction `boolean[] sacGloutonParValeurs(objet[] T, int c)` de terme général $T[k]$ = " l'objet k est dans le sac de contenance c " ;
2. écrire une fonction `boolean[] sacGloutonParDensitesDeValeurs(objet[] T, int c)` de terme général $T[k]$ = " l'objet k est dans le sac de contenance c " ;
3. écrire une fonction `int valeurDuSac(boolean[] g, objet[] T)` qui retourne la valeur d'un sac glouton g construit sur l'ensemble d'objets représenté par le tableau T ;
4. écrire une fonction `objet[] aleatoires(int n, int vsup, int tsup)` qui retourne un tableau de n objets : l'objet i est en case i, sa valeur est au hasard dans l'intervalle $[0 : v_{\text{sup}}]$, sa taille est au hasard dans l'intervalle $[1 : t_{\text{sup}} + 1]$.

Vous comparerez les solutions gloutonnes par valeurs et gloutonnes par densités de valeurs à l'aide d'une validation statistique. Cette validation calcule les valeurs de sacs gloutons construits sur des ensembles d'objets aléatoires. Pour chaque ensemble d'objets aléatoires on calcule et on mémorise la valeur du sac glouton par valeurs, et de même pour le sac glouton par densités de valeur. On procède ainsi pour un grand nombre d'ensembles d'objets aléatoires. Puis on retourne la médiane des sacs gloutons par valeurs et la médiane des sacs gloutons par densités de valeurs[3].

Donner les valeurs médianes des deux approches gloutonnes pour $N = 10^6$ ensembles d'objets aléatoires, $n_{\text{sup}} = 10^2$, $c_{\text{sup}} = 10^3$, $t_{\text{sup}} = c/10$ où c est la contenance du sac choisie au hasard dans l'intervalle $[0 : c_{\text{sup}}]$, et $v_{\text{sup}} = 10^2$.

Le projet est à réaliser en binôme. Les monômes doivent être l'exception (nombre impair d'étudiants dans un groupe, etc.)

Les projets sont à envoyer à votre intervenant de TD de la semaine du 8 février. La date limite d'envoi est vendredi 12 février.

Votre courrier électronique contiendra quatre fichiers : le programme Projet1.java (texte source) avec vos commentaires, remarques, exemples d'exécution ; le même fichier au format PDF, l'histogramme des ‘‘sacs gloutons par valeurs'' dans un fichier valeursSGV.png et l'histogramme des ‘‘sacs gloutons par densités'' dans un fichier valeursSGD.png.
Ces quatre fichiers seront dans une archive 'zip' ‘‘nom1nom2.zip'' où nom1 et nom2 sont les noms du binôme OU nom.zip si le projet a été fait en solo.

Ne pas oublier d'indiquer également les noms du binôme dans le programme java.

L'objet de votre courrier électronique sera ‘‘Algorithmique : projet 1 nom1 nom2'' ou ‘‘Algorithmique : projet 1 nom''

Le programme java doit pouvoir être compilé dans un terminal Unix ou Linux par la seule commande ‘‘javac Projet1.java'' puis exécuté dans ce même terminal par la commande ‘‘java Projet1''.

Remarque importante : les sapiens (que nous sommes) ne savent pas lire les programmes non commentés ou mal commentés.

Pages suivantes : le squelette du programme.

[3] Une validation statistique complète nous dirait, en plus, si ces deux valeurs médianes sont significativement différentes. Elle retournerait les deux valeurs médianes et la "p-value" des valeurs des sacs gloutons par valeurs et gloutons par densités de valeurs. La notion de p-value relève d'un cours de statistiques plus que d'un cours d'algorithmique. Pour en savoir plus, je vous recommande la vidéo de Josh Starmer, `https://www.youtube.com/watch?v=5Z9OIYA8He8` de la société StatQuest.

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Random;
public class Projet1{
   static Random random = new Random();
   public static void main(String[] args){
      int vsup = (int)Math.pow(10,2), // les valeurs des objets seront au hasard dans [0:vsup+1]
          csup = (int)Math.pow(10,3), // la capacité du sac sera au hasard dans [0:csup+1]
          nsup = (int)Math.pow(10,2); // le nombre d'objets sera au hasard dans [0:nsup+1]

      int   N = (int)Math.pow(10,6); // nombre de sacs aléatoires de la validation statistique
      // (N est le nombre de "runs" de la validation statistique)

      int[]    valeursSGV = new int[N], // valeurs des N sacs aléatoires, "gloutons par valeurs"
               valeursSGD = new int[N]; // valeurs des N sacs aléatoires, "gloutons par densités"
      for (int r = 0; r < N; r++){ // r est le numéro du "run" de la validation statistique
         if (r%1000==0) System.out.print("."); // mettre un peu d'animation dans l'exécution...
         // génération d'un ensemble d'objets aléatoires en valeurs et tailles
         int n = random.nextInt(nsup+1); // nombre d'objets au hasard dans [0:nsup+1]
         int c = random.nextInt(csup+1),  // taille du sac aléatoire dans [0:csup+1]
             tsup  = c/10 ; // les objets seront de taille aléatoire dans [1:tsup+1]
         // Un ensemble de n objets aléatoires, valeurs dans [0:vsup+1] et tailles dans [1:tsup+1]
         objet[] objets = ObjetsAleatoires(n,vsup,tsup);
         // remarque : pas d'objet de taille 0. Ils conduiraient à une densité v/t infinie.
         boolean[] sgpv = sacGloutonParValeurs(objets, c); // sac de contenance c, "glouton par valeurs"
         boolean[] sgpd = sacGloutonParDensites(objets, c); // id., "glouton par densités de valeurs"
         int valeurSGV = valeurDuSac(sgpv, objets), // valeur du sac "glouton par valeur"
             valeurSGD = valeurDuSac(sgpd, objets);   // valeur du sac "glouton par densités"
         valeursSGV[r] = valeurSGV ;
         valeursSGD[r] = valeurSGD ;
      }
      System.out.println();
      int medianeRecursive_gpv = medianeRecursive(valeursSGV),
      medianeRecursive_gpd = medianeRecursive(valeursSGD);
      int medianeIterative_gpv = medianeIterative(valeursSGV),
      medianeIterative_gpd = medianeIterative(valeursSGD);
      float    moyenne_gpv = moyenne(valeursSGV),
               moyenne_gpd = moyenne(valeursSGD);
      System.out.printf("medianes récursive et iterative et moyenne des valeurs des sacs : \n");
      System.out.printf("Sacs glouton par valeurs : %d, %d, %d\n",
         medianeRecursive_gpv, medianeIterative_gpv, (int)moyenne_gpv);
      System.out.printf("Sacs glouton par densité : %d, %d, %d\n",
         medianeRecursive_gpd, medianeIterative_gpd, (int)moyenne_gpd);
      EcrireDansFichier(valeursSGV,"valeursSGV.csv");
      EcrireDansFichier(valeursSGD,"valeursSGD.csv");
      System.out.println("Valeurs des sacs \"gloutons par valeurs\" dans le fichier " + "valeursSGV.csv");
      System.out.println("Valeurs des sacs \"gloutons par densités\" dans le fichier " + "valeursSGD.csv");
      // Pour générer les histogrammes : ouvrir une fenêtre terminal,
      // se placer dans le répertoire contenant les fichiers "valeursSGV.csv" et "valeursSGD.csv"
      // et le fichier   histogramme.py
      // et lancer la commande  "python histogramme.py" (python 2.x)
      // ou "python3 histogramme.py" (python 3.x)
   }

// Exercice 1 : CALCUL DE LA MEDIANE
   static int qselRecursif(int p, int[] T, int i, int j){ int m = j-i; // 0 <= p < m
      ...
   }
   public static int quickSelectRecursif(int p, int[] T){ // 1 <= p <= n;
      int n = T.length;
      return qselRecursif(p-1, T, 0, n);
   }
  static int qselIteratif(int p, int[] T){ int n = T.length; // 0 <= p < n
  int pprime = ... ,  i =....,  j = ... ; // I(p', i, j)
    while (!(....)) { // I(p',i,j) et non arrêt
       int k = segmenter(T, i, j);
       int pppi = pprime + i ;
       if ( i < pppi && pppi < k ) // I(..., ..., ...)
         { ..... } // I(p', i, j)
       else
       if ( k <= pppi && pppi < k+1 ) // I(..., ..., ...)
         { ........ } // I(p', i, j)
       else //  k+1 <= pppi && pppi < j ) // I(..., ..., ...)
         { ........} // I(p', i, j)
    // I(p', i, j) et arrêt, donc la p-ème valeur de T[ 0 : n ] est ...
    return ...;
   }
   public static int quickSelectIteratif(int p, int[] T){ int n = T.length;
      // 1 <= p <= n;
      return qselIteratif(p-1, T);
   }
   static int medianeRecursive(int[] T){ int n = T.length;
   /* Retourne la valeur médiane de T[0:n]. C'est la valeur m du tableau
   telle que T contient autant de valeurs <= à m que de valeurs > à m.
   Exemple  : 0,1,2,3 ==> médiane = 1 (indice (4-1)/2 = 1)
              0,1,2 ==> médiane = 1 (indice (3-1)/2 = 1 )
   De façon générale : avec la convention 0 <= p < n, la valeur médiane est la
   p = (n-1)/2 ème valeur de T.
   */
      return quickSelectRecursif(1 + (n-1)/2, T);
      // ou si l'on préfère : qselRecursif((n-1)/2, T, 0, T.length)
   }
   static int medianeIterative(int[] T){ int n = T.length;
   /* Retourne la valeur médiane de T[0:n]. C'est la valeur du tableau
   telle que T contient autant de valeurs <= à la médiane que de valeurs >= à m.
   Exemple  : 0,1,2,3 ==> médiane = 1 (indice (4-1)/2 = 3/2 = 1)
              0,1,2 ==> médiane = 1 (indice (3-1)/2 = 2/2 = 1 )
   De façon générale, avec la convention 0 <= p < n, la valeur médiane est la
   p = (n-1)/2 ème valeur de T.
   */
      return quickSelectIteratif(1 + (n-1)/2, T);
      // ou si l'on préfère : qselIteratif((n-1)/2, T)
   }
```

```
    static int segmenter(int[] T, int i, int j){
    // calcule une permutation des valeurs de T[i:j] qui vérifie
    // T[i:k]<= T[k:k+1] < T[k+1:j], et retourne l'indice k.
      ...
      return k;
    }
    static int hasard(int i, int j){ Random r = new Random();
    // retourne un indice au hasard dans l'intervalle [i:j]
        return i + r.nextInt(j-i); // nextInt(j-i) est dans [0:j-i]
    }
    static void permuter(int[] T, int i, int j){
        int ti = T[i];
        T[i] = T[j];
        T[j] = ti;
    }

// EXERCICE 2 :
   // Un objet est défini par son numéro i, sa valeur v, sa taille t, sa densité v/t
   static class objet{int i, v, t; float d;
      objet(int i, int v, int t, float d){
         this.i = i; this.v = v; this.t = t; this.d = d;
      }
   }
   static int valeurDuSac(boolean[] sac, objet[] Objets){int n = sac.length;
      objet[] ooi = new objet[n]; // objets dans l'ordre initial :
      // ooi[0] est l'objet de numéro i = 0, ..., ooi[n-1] est l'objet de numéro n-1.
      ...
      // calcul de la valeur du sac
      int vds = 0; // valeur du sac
      ...
      return vds;
   }
   static boolean[] sac(objet[] Objets, int c){
   // Objets triés par valeurs décroissantes ou par densités décroissantes.
   // Retourne un sac glouton selon le critère du tri.
      int n = Objets.length, // cardinal de l'ensemble des objets
         r = c; // capacité disponible (restante) dans le sac ;
      boolean[] sac = new boolean[n];
      ...
      return sac;
   }
   static boolean[] sacGloutonParValeurs(objet[] Objets, int c){
      qspvd(Objets); // tri quicksort des objets par valeurs décroissantes
      return sac(Objets, c); // sac glouton par valeurs.
   }
   static boolean[] sacGloutonParDensites(objet[] Objets, int c){
      qspdd(Objets); // tri quicksort des objets par densités décroissantes
      return sac(Objets, c); // sac glouton par densités décroissantes
   }
   static void qspvd(objet[] Objets){// quickSort des objets par valeurs décroissantes
      qspvd(Objets, 0, Objets.length);
   }
   static void qspdd(objet[] Objets){// quickSort des objets par densités décroissantes
      qspdd(Objets, 0, Objets.length);
   }
   static void qspvd(objet[] Objets, int i, int j){
   // quicksort par valeurs décroissantes de Objets[i:j]
      ...
   }
   static void qspdd(objet[] Objets, int i, int j){// quicksort par densites décroissantes
      ...
   }
   static int spvd(objet[] Objets, int i, int j){
   // segmentation de Objets[i:j] par valeurs décroissantes
      // I(k,jp) :
      //   valeurs de Objets[i:k] >= valeurs de Objets[k] > valeurs de Objets[k+1:jp]
      ...
      return k;
   }
   static int spdd(objet[] Objets, int i, int j){
   // segmentation de Objets[i:j] par densités décroissantes
      // I(k,jp) :
      //   densités de Objets[i:k] >= densités de Objets[k:k+1] > densités de Objets[k+1:jp]
      ...
      return k;
   }
   static void permuter(objet[] Objets, int i, int j){
      objet x = Objets[i]; Objets[i] = Objets[j]; Objets[j] = x;
   }
   static void EcrireDansFichier(int[] V, String fileName){
      try { int n = V.length;
         PrintWriter ecrivain;
          ecrivain =  new PrintWriter(new BufferedWriter(new FileWriter(fileName)));
          for (int i = 0; i < n-1; i++)
               ecrivain.println(V[i]);
           ecrivain.println(V[n-1]);
          ecrivain.close();
       }
      catch(IOException e){ System.out.println("Erreur écriture");}
     }

    // CALCUL DE LA MOYENNE
    static float moyenne(int[] T){int n = T.length;
       float s = 0;
       for (int i = 0; i<n; i++) s = s + T[i];
       return s/n;
    }
```

```
    // PROCEDURES NON UTILISEES mais utiles ou agréables lors de la phase de mise au point du programme
    static void afficher(String s){System.out.print(s);};
    static void afficher(boolean[] B){ int n = B.length;
       for (int i = 0; i < n; i++) if (B[i]) System.out.print(i + " ");
       System.out.println();
    }
    static void afficher(int[] T){ int n = T.length;
       for (int i = 0; i < n; i++) System.out.print(T[i] + " ");
       System.out.println();
   }
   static void afficher(objet[] T){ int n = T.length;
      afficher("i-v-t-d : ");
      for (int i = 0; i < n; i++){ objet o = T[i];
         System.out.printf("%d-%d-%d-%f | ", o.i, o.v, o.t, o.d);
      }
      System.out.println();
   }
   static void newline(){System.out.println();}
}

/* Compilation et exécution du programme :
% javac Projet1.java
% java Projet1
médianes récursive et itérative et moyenne des valeurs des sacs :
Sacs glouton par valeurs : ---, ---, ---
Sacs glouton par densité : ---, ---, ---
Valeurs des sacs "gloutons par valeurs" dans le fichier valeursSGV.csv
Valeurs des sacs "gloutons par densités" dans le fichier valeursSGD.csv
%
*/
```