```java
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

import java.util.Random;

public class Projet {
  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 = 1 + (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;
    if (m == 1) { // Si m = 1, p est nécéssairement égal à 0. La p-ème valeur de T[i:j] est T[i]
      return T[i];
    } else { // Si m > 1, p est nécéssairement dans l'intervalle [0:m]
      int k = segmenter(T, i, j); // On cherche une partition de T[i:j] pour comparer p+i à son
résultat
      int ppi = p + i;
      if (i ⩽ ppi && ppi < k) { // Si i ⩽ p+i < k la p-ème valeur de T est la p-ème valeur de T[i:k]
        return qselRecursif(ppi - i, T, i, k);
      } else if (k ⩽ ppi && ppi < k + 1) { // Si p+i == k la p-ème valeur de T est T[k]
        return T[k];
      } else { // Si k ⩽ p+i < j la p-ème valeur de T est la (p+i-(k+1))-ème valeur de T[i:k]
        return qselRecursif(ppi - (k + 1), T, k + 1, j);
      }
```

```
  }
}

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;
  int pprime = p, i = 0, j = n;
  int k, pppi;
  while (!(i == j)) {
    k = segmenter(T, i, j);
    pppi = pprime + i;
    if (i ⩽ pppi && pppi < k) {
      j = k;
      pprime = p - i;
    } else if (k ⩽ pppi && pppi < k + 1) {
      return T[k];
    } else {
      i = k + 1;
      pprime = p - i;
    }
  }
  return T[pprime];
}

public static int quickSelectIteratif(int p, int[] T) {
  return qselIteratif(p - 1, T);
}

static int medianeRecursive(int[] T) {
  int n = T.length;
  return quickSelectRecursif(1 + (n - 1) / 2, T);
}

static int medianeIterative(int[] T) {
  int n = T.length;
  return quickSelectIteratif(1 + (n - 1) / 2, T);
}

static int segmenter(int[] T, int i, int j) {
  // Calcule une permutation des valeurs de T[i:j] qui vérifie
  // I(k, i, j): T[i:k] ⩽ T[k:k+1] ⩽ T[k+1:j], et retourne l'indice k
  int k = i, jp = k + 1;
  while (jp < j) {
    if (T[jp] > T[k]) {
      jp++;
    } else {
      permuter(T, jp, k + 1);
      permuter(T, k + 1, k);
      k++;
      jp++;
    }
  }
  return k;
}

static int hasard(int i, int j) {
  return i + random.nextInt(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;
  }
```

```
  }

  // Un ensemble de n objets aléatoires à valeurs et tailles dans [0:vsup+1] et
  // [1:tsup+1]
  static Objet[] ObjetsAleatoires(int n, int vsup, int tsup) {
    Objet[] E = new Objet[n]; // ensemble de n objets
    for (int k = 0; k < n; k++) {
      int i = k; // On assigne indice à la variable
      int v = random.nextInt(vsup + 1); // On assigne à v une valeur aléatoire dans [0:vsup+1]
      int t = 1 + random.nextInt(tsup + 1); // On assigne à t une taille aléatoire dans [1:tsup+1]
      float d = v / t; // On calcule la densité aléatoire par rapport aux t et v précédents
      E[k] = new Objet(i, v, t, d); // On initialise l'objet aléatoire avec les propriétés précédentes
    }
    return E;
  }

  static int valeurDuSac(boolean[] sac, Objet[] objets) {
    int vds = 0; // valeur du sac
    // Pour chaque objet dans le tableau objets, si ce dernier rentre dans le sac,
    // c'est-à-dire que la valeur de sac à son indice initial est TRUE, alors on
    // ajoute sa valeur à la valeur du sac
    for (Objet obj : objets) {
      if (sac[obj.i]) {
        vds += obj.v;
      }
    }
    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, r = c;
    boolean[] sac = new boolean[n]; // Sac est un tableau parallèle au tableau objets d'ordre initial
tel que si
                                    // T[i] rentre dans le sac alors sac[i] = true

    // Pour chaque objet, si sa taille est inferieur à l'espace restant, on assigne
    // TRUE à l'indice initial de ce dernier dans le tableau sac
    for (Objet obj : objets) {
      int taille = obj.t;
      if (taille ⩽ r) {
        sac[obj.i] = true;
        r -= taille;
      }
    }
    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]
    if (j - i < 2)
      return;
    int k = spvd(objets, i, j);
    qspvd(objets, i, k);
    qspvd(objets, k + 1, j);
  }

  static void qspdd(Objet[] objets, int i, int j) {
    // quicksort par densites décroissantes
    if (j - i < 2)
      return;
```

```
    int k = spdd(objets, i, j);
    qspdd(objets, i, k);
    qspdd(objets, k + 1, j);
  }

  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]
    int k = i, jp = k + 1;
    while (jp < j) {
      if (objets[jp].v ⩽ objets[k].v) {
        jp++;
      } else {
        permuter(objets, jp, k + 1);
        permuter(objets, k + 1, k);
        k++;
        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]
    int k = i, jp = k + 1;
    while (jp < j) {
      if (objets[jp].d ⩽ objets[k].d) {
        jp++;
      } else {
        permuter(objets, jp, k + 1);
        permuter(objets, k + 1, k);
        k++;
        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 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();
  }
}
```