TP DUT Informatique

À la fin de la séance, envoyez vos fichiers `.java` et les réponses aux questions (dans des fichiers `.txt`) à `dp.cnam@gmail.com` avec le sujet :

[DUTALGO] TP noté 1 NOM PRENOM

Attention : le non-respect de cette consigne est pénalisé de 10%.
Chaque devoir est individuel. On va utiliser un logiciel de détection automatique de plagiat.

Informations techniques PC Suse :

(a) Pour démarrer une session : utilisateur **licencep** et mot de passe **7002n***.

(b) Pour démarrer un *terminal* : `l'icône lézard` en haut à droite → Terminal → Konsole.

(c) Pour ouvrir un gestionaire/navigateur de fichiers : `l'icône lézard` → Système → Dolphin.

(d) Pour modifier un fichier, clic droit sur le fichier → Ouvrir avec Kate

(e) Pour compiler un programme `Java` : utiliser l'IDE `drjava`, ou `netbeans`, ou ouvrir un terminal, se placer dans le dossier du programme et utiliser `javac NOMPROGRAMME.java`. Pour l'exécuter, taper `java NOMPROGRAMME`

1 Comparer deux méthodes de tri

Exercice 1 Soit le (pseudo-)code à droite. Modifier ce code pour écrire un programme Java `Exo1.java` capable de trier $n = 20$ nombres aléatoires en **ordre décroissant.**

Exercice 2 Modifier le code et écrire un nouveau fichier `Exo2.java` où on met n=150000. Lancer le programme à l'aide de la commande Linux ci-après et noter le temps de calcul. N'oublier de commenter les lignes qui affichent les tableau. `time java Exo2`
Faire une **moyenne** sur 5 exécutions.

Exercice 3 Trouver la valeur de n à partir de laquelle le programme précédent nécessite plus de 50 secondes ?

```
int[] tab;
public static int indiceMinIntervalle(int i1, int i2){
    int indiceMin = i1;
    for(int i=i1+1; i<=i2; i++)
        if(tab[i]<tab[indiceMin])
            indiceMin = i;
    return indiceMin;
}
public static void main(){
   int n = 20;
   tab = new int[n];
   for(int i=0;i<n;i++)
       tab[i] = (int)(50*java.lang.Math.random());
   //Affichage tableau de départ,
   System.out.println(java.util.Arrays.toString(tab));
   //Tri:
   for(int i=0;i<n;i++){
       int indiceMin = indiceMinIntervale(i,n-1);
       int tmp = tab[indiceMin];
       tab[indiceMin] = tab[i];
       tab[i]       = tmp;
    }
   //Affichage tableau trié
   System.out.println(java.util.Arrays.toString(tab));
}
```

Exercice 4 Utiliser le code à droite pour écrire un programme Java `Exo4.java` qui permet de réaliser un tri à bulle **en ordre décroissant** sur un tableau avec $n = 20$ cases.

Exercice 5 Modifier le code et écrire un nouveau fichier `Exo5.java` où on met n=150000. Lancer le programme à l'aide de la commande Linux ci-après et noter le temps de calcul, faire aussi une moyenne. `time java Exo5`

Exercice 6 Trouver la valeur de n à partir de laquelle le programme précédent nécessite plus de 50 secondes ? Comparer avec l'exercice 3, quel algorithme est plus rapide ?

```
boolean triFini = false;
while(!triFini){
    triFini = true;
    for(int i=0;i<n-1;i++)
        if(tab[i]>tab[i+1]){
            triFini = false;
            int tmp = tab[i];
            tab[i] = tab[i+1];
            tab[i+1]=tmp;
        }
}
```

N'hésitez pas à regarder le cours disponible à `http://cedric.cnam.fr/~porumbed/dutalgo/`

2 Algorithmes de tri plus rapides !

Exercice 7 Inspirez vous du code ci-après pour implémenter un tris par dénombrement **en ordre décroissant**. Écriré un fichier `Exo7.java` et modifier la valeur de n pour mettre $n = 150000 \times 100$, c.à.d., 100 fois plus que la valeur utilisée aux exercices précédents. Vérifier le temps de calcul avec la commande Linux ci-après et n'oublier pas de faire une moyenne sur 5 exécutions.

```
time java Exo7
```

```
int[] tab;//mettre en static
//ajouter public static void main...
int n = 20;
tab = new int[n];
for(int i=0;i<n;i++)
    tab[i] = (int)(50*java.lang.Math.random());
//Affichage tableau de départ
println(java.util.Arrays.toString(tab));
//Tri:
int[] apparitions = new int[50];
for(int i=0;i<n;i++)
    apparitions[tab[i]]++;
int pos = 0;
for(int val=0;val<50;val++)
    for(int j=0;j<apparitions[val];j++){
        tab[pos] = val;
        pos++;
    }
//Affichage tableau trié
println(java.util.Arrays.toString(tab));
```

Exercice 8 Trouver la valeur de n à partir de laquelle le programme précédent nécessite plus de 50 secondes ? Comparer avec l'exercice 3, quel algorithme est plus rapide ?

Exercice 9 Modifier le programme précédent et écrire un nouveau fichier `Exo9.java` dans lequel on génère des nombres aléatoires de 1 à 1000000 pour le mettre le tableau `tab`. Donner le temps de calcul du tri par dénombrement pour ce cas (attention : il faut modifier le code si la valeur max est 1000000).

Exercice 10 Écrire un fichier `Exo10.java` pour tester la vitesse de la méthode de tri de la bibliothèque standard `Java`. Attention, trier en ordre décroissant et utiliser $n = 150000 \times 100$. Il faut :

- inverser les nombres (ex, $5 \rightarrow -5$) ;
- Appeler `java.util.Arrays.sort(tab)` ;
- inverser les nombres de nouveau.

Indiquer le temps de calcul ; n'oubliez pas de faire une moyenne sur 5 exécutions.

Exercice 11 Trouver la valeur de n à partir de laquelle le programme précédent nécessite plus de 50 secondes ? Comparer avec l'exercice 7, quel algorithme est plus rapide : le tri par dénombrement ou le tri de la bibliothèque standard ?

Exercice 12 Modifier le programme précédent et écrire un nouveau fichier `Exo12.java` dans lequel on génère des nombres aléatoires de 1 à 1000000 pour le mettre le tableau `tab`. Quel est le nouveau temps de calcul de la méthode de tri de la bibliothèque standard ?

3 Sac-à-dos : algorithme glouton

Exercice 13 On considère un problème de `sac-à-dos` simplifié où tous les poids valent 1. Modifier le programme ci-après pour le faire afficher :

- l'article le plus rentable (max val/kg), donc celui de *valeur maximale* car les poids valent tous 1
- la solution trouvée par l'algorithme glouton. Rappel : l'algorithme glouton mets les articles dans le sac un par un, à partir du plus rentable jusqu'au moins rentable. Comme tous les poids valent 1, on peut mettre les $Pmax$ plus rentables articles dans le sac. Vous aller uniquement avoir besoin de trier les valeurs en ordre décroissant, utiliser le `tri à bulle` présenté à la séance précédente.

```
1  class Exo13{
2  public static void main(String[] args){
3      int[] val;
4      int[] poids;
5      int Pmax = 15;
6      int n  = 20;
7      val = new int[n];
8      poids = new int[n];
9      for(int i=0;i<n;i++){
10         val[i]      = (i+1)*7%Pmax;
11         poids[i]    = 1;
12     }
13     int lePlusRentable= 0;
14     //à remplir: trouver le plus rentable article (max val)
15     System.out.println("Article le rentable est à l'indiceest à l'indice  "+
           lePlusRentable);
16     int valMax          = 0;
17 //à remplir: trouver la valeur max des articles qu'on peut mettre dans le sac
18     System.out.println("Le profit maximal est :"+valMax);
19     }
20 }
```

N'oublier pas d'envoyer à la fin de la séance vos fichiers `.java` à `dp.cnam@gmail.com` avec le sujet :

[DUTALGO] TP noté 2 NOM PRENOM

Exercice 14 On modifie les données du programme précédent. Cette fois ci, on considère que tous les profits valent 1. Démarrer à partir du même code ci-dessus, mais remplacer dès le début les lignes 9-12 par :

```
    for(int i=0;i<n;i++){
        val[i]      = 1;
        poids[i]    = (i+1)*7%Pmax+1;
    }
```

Répondre aux même deux questions :

- trouver l'article le plus rentable (max val/kg), donc celui de poids min, car les valeurs valent 1
- Trouver une solution à l'aide de l'algorithme glouton. Rappel : l'algorithme glouton mets les articles dans le sac un par un, à partir du plus rentable jusqu'au moins rentable. Comme tous les valeurs valent 1, le plus rentable est l'article le moins lourd. **Il faut trier les poids en ordre croissant :** utiliser un `tri à bulles`. Il faut ensuite faire une boucle pour ajouter les articles un par un, tant que la capacité $Pmax$ n'est pas dépassée.

Exercice 15 On considère un problème de sac-à-dos avec des poids et des valeurs variées. Démarrer à partir de même code, mais initialiser les poids et les valeurs avec :

```
    for(int i=0;i<n;i++){
        val[i]      = (i+2)*13%5;
        poids[i]    = (i+1)*7%Pmax+1;
    }
```

Répondre aux même questions :

- Afficher l'article le plus rentable (max val/kg)
- Déterminer une solution à l'aide de l'algorithme glouton. Il faut trier les articles dans l'ordre décroissant de leur rentabilité (val/kg).
  - Utiliser un `tri à bulles` : à chaque inversion, il faut *inverser les poids et les valeurs.*

  Une fois les articles triés, commencer à les mettre dans le sac, à partir du plus rentable jusqu'au moins rentable !

Il ne faut pas hésiter à consulter le cours disponible en ligne à
`http://cedric.cnam.fr/~porumbed/dutalgo/`

4 Sac-à-dos : programmation dynamique

N'oublier pas d'envoyer à la fin de la séance vos fichiers `.java` à `dp.cnam@gmail.com` avec le sujet :

[DUTALGO] TP noté 2 NOM PRENOM

Exercice 16 Un guide de montagne part en excursion et détermine le contenu de son sac à dos. Il décide que le poids de nourriture qu'il peut emporter ne doit pas dépasser $Pmax = 8$ kilos. Il dispose de trois types d'aliments, de valeur nutritive variable et de poids variés. Les valeurs et les poids sont résumées dans le tableau en bas à droite.

Remplir d'abord le tableau en bas avec des valeurs (nutritives) pour résoudre le problème de sac-à-dos par la programmation dynamique. Écrire la réponse finale dans un ficher texte `Exo16.txt`!!
Rappel : on utilise une fonction $f(i,p)$ = valeur totale maximum d'un ensemble d'objets choisis parmi les i premiers, et de poids total p. La formule est
$f(i+1,p) = \max(f(i,p), \quad \underbrace{f(i, p - poids[i]) + profit[i]}_{\text{objet d'indice } i \text{ sélectionné, si } poids[i] \leq p} \quad)$

	valeur	poids(kg)
Article 1	7	8
Article 2	3	4
Article 3	6	4

	Poids total								
	0	1	2	3	4	5	6	7	8
$i = 3$ articles utilisés									
$i = 2$ articles utilisés									
$i = 1$ articles utilisés									
$i = 0$ articles utilisés	0	0	0	0	0	0	0	0	0

Exercice 17 Inspirez vous du code ci-après pour résoudre le `sac-à-dos` sur des données aléatoires à l'aide de la programmation dynamique. Écriré un fichier `Exo17.java` qui calcule et affiche la valeur maximale $valeurMax$.

```
1  int Pmax = 1000;                          //la capacité Poids max
2  int n  = 150;                             //nombre d'articles
3  int[][] mat = new int[n+1][Pmax+1]; //n+1 lignes, Pmax+1 colonnes
4  val    = new int[n];
5  poids = new int[n];
6  for(int i=0;i<n;i++){
7      val[i]    = (int)(100*java.lang.Math.random());
8      poids[i] = (int)(Pmax/10*java.lang.Math.random());
9  }
10 for(int i=0;i<n;i++)
11     for(int p=0;p<=Pmax;p++)
12         if(poids[i]<=p)
13             mat[i+1][p] = Math.max(mat[i][p],
14                                    mat[i][p-poids[i]]+val[i]);
15 int valeurMax = 0;
16 for(int p=0;p<=Pmax;p++)
17     valeurMax = Math.max(valeurMax,mat[n][Pmax]);
```

Exercice 18 Trouver la capacité $Pmax$ à partir de laquelle le programme nécessite plus d'une demi-seconde.

Exercice 19 Trouver le nombre n à partir duquel le programme nécessite plus d'une demi-seconde.

Exercice 20 Modifier le code du programme précédent pour le faire utiliser des valeurs fractionnaires de type `double` (càd, mettre `double[][] mat = new ...`). Vous observez une augmentation du temps de calcul ?