TP 15 VARI1 : Algorithmes

1 Le tri par sélection, le tri à bulles et le tri de la bibliothèque standard

Exercice 1 Soit le code à droite. Modifier ce code pour écrire un programme `Exo1.java` qui permet de trier $n = 20$ nombres aléatoires par **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 affiché (ligne « `real` »). N'oublier pas de *commenter les lignes qui affichent les tableaux*, car l'affichage de $n = 150000$ nombres ralentit le programme.

`time java Exo2`

→ → → prendre la ligne « `real` »

Faire une **moyenne** sur 3 exécutions.

Exercice 3 Trouver la valeur de n à partir de laquelle le programme précédent nécessite plus de 10 secondes environ !!

```
import java.util.*;
class TriSelect{
   static int[] tab;
   static int intervIndiceMin(int idxDebut,..){
        //prendre le code du cours 15, diapo 1:
        //mais il faut chercher min ou max pour
        //faire un tri décroissant?
   }
   public static void main(String[] args){
      int n  = 20;
      tab = new int[n];
      for(int i=0;i<n;i++)
          tab[i] = (int)(50*Math.random());
      //Afficher le tableau initial
      System.out.println(Arrays.toString(tab));
      //Réaliser le tri avec une boucle:
      for(int i=0;i<n;i++){
          int j = intervIndiceMin(i,n-1);
          int tmp = tab[j];          //ESSAYER DE
          tab[j]  = tab[i];          //TOUT
          tab[i]  = tmp;             //COMPRENDRE
      }
      //Afficher le tableau trié
      System.out.println(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 **par 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 10 secondes ? Comparer avec l'exercice 3, quel algorithme est plus rapide ?

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

Exercice 7 Écrire un fichier `Exo7.java` et initialiser le tableau `tab` comme à l'exercice 1. Faire appel à la fonction de tri de la bibliothèque standard `Java` pour trier par ordre décroissant. Il faut :

- inverser les nombres (ex, 5→ −5) ;
- appeler `java.util.Arrays.sort(tab)` ;
- inverser les nombres de nouveau.

Exercice 8 Trouver la valeur de n à partir de laquelle le programme précédent nécessite plus de 10 secondes ? Comparer avec l'exercice 3 et l'exercice 6, quel algorithme est plus rapide ?

Exercice 9 Modifier le programme précédent et remplir le tableau `tab` avec des nombres aléatoires de 1 à 1000000, c. à. d., remplacer `50*Math.random()` par `1000000*Math.random()`. Quel est le nouveau temps de calcul du tri de la bibliothèque standard ?

2 Sac-à-dos : algorithme glouton

Exercice 10 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 par ordre décroissant, utiliser un `tri à bulle` ou le tri de la bibliothèque standard via `Arrays.sort(..)`.

```
class Exo10{
  public static void main(String[] args){
    int n    = 15;   //nombre d'articles
    int Pmax = 10;   //poids max (capacité)
    int[] val   = {8, 7,14, 6,13, 5,12, 4,11, 3,10, 2, 9, 1, 8};
    int[] poids = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
    int lePlusRentable= 0;
    //à remplir: trouver le plus rentable article (max val)
    System.out.println("Article le rentable est à l'indice "+lePlusRentable);
    int valMax        = 0;
    //à remplir: trouver la valeur max des (10?) articles à mettre dans le sac
    //....
    System.out.println("Le profit (valeur) maximal est :"+valMax);
  }
}
```

Exercice 11 On considère un problème de sac-à-dos avec des poids et des valeurs variées. Démarrer à partir du même code, mais initialiser les poids et les valeurs :

```
    val = new int[n]; poids=new int[n];
    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.*
Après avoir triés les articles, commencer à les mettre dans le sac, à partir du plus rentable jusqu'au moins rentable !
Observation : on ne peut **pas** utiliser le tri de la bibliothèque standard `Arrays.sort(...)`, car il faut échanger en même temps les cases des tableaux `val` et `poids`.

3 Sac-à-dos : programmation dynamique

Exercice 12 Un guide de montagne part en excursion et détermine le contenu de son sac à dos. 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.

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

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

	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 13 Écrire un programme qui permet de calculer le tableau/matrice à gauche, c. à. d., résoudre le sac-à-dos par programmation dynamique.