Dans tout le TD, lorsqu'il est demandé de déterminer la complexité, le but est de déterminer le plus petit a tel que (pour un exemple où l'algorithme aurait un tableau en paramètre): "il existe une constante c_1 telle que pour tout tableau t de $n \geq 1$ cases, $m \leq c_1 n^a$ où m est le nombre d'opération de l'algorithme".

1 Calcul de complexité d'un aglorithme donné

Exercice 1. Parcours partiel vs total
On considère les deux algorithmes de recherche suivants

```
public static boolean rechTotal(int[] t, int x){
  boolean trouve = false;
  for(int i=0;i<t.length;i++){
    if(t[i]==x){
      trouve=true;
    }
  }
  return trouve;
}

public static boolean rechPartiel(int[] t, int x){
  boolean trouve = false;
  int i=0;
  while(!trouve && i<t.length){
    if(t[i]==x){
      trouve=true;
    }
    i++;
  }
  return trouve;
}
```

Question 1.1.
Déterminez la complexité de rechTotal(t,x).

Question 1.2.
Déterminez la complexité de rechPartiel(t,x).

Conclusion : le parcours partiel et total ont la même complexité dans notre modèle, car on considère le pire des cas.

Exercice 2. Algorithmes génétiques

Question 2.1.
Calculez la complexité de l'algorithme génétique (pour le problème du ramassage des picèes) vu en cours. Considérez la version où l'on utilise

- l'algorithme de calculNext vu en cours
- des individus type "gdbh"
- un croisement et une mutation "naifs" (de votre choix, ou suivant la version décrite par votre encadrant.e

Exercice 3. Une boucle mais ...

On considère l'algorithme suivant :

```
public static void printCouples(int n){

  int a = n;
  int b = n;
  while(b >= 0){
    println(a + "␣" + b);
    a−−;
    if(a==0){
      b−−;
      a=n;
    }
  }
}
```

Question 3.1.
Pour tout n, montrer que $m \geq n^2$ (comme nous l'avons vu en cours, cela impliquera qu'il n'existe pas de constante c_1 telle que $m \leq c_1 n$ pour tout $n \geq 1$).

Question 3.2.
Déterminez la complexité de cet algorihme.

2 Recherche d'un algorithme de complexité imposée

Exercice 4. 3-SUM

Question 4.1.
Ecrire une méthode `boolean somme(int[]t)` qui retourne vrai ssi il existe trois entiers distincts i, j et k tels que $t[i] + t[j] + t[k] = 0$ et qui s'execute en temps cubique. C'est à dire, montrez qu'il existe une constante c telle que pour tout tableau t de $n \geq 1$ cases, $m \leq cn^3$ où m est le nombre d'opération de somme(t).

L'objectif des deux questions suivantes est d'améliorer l'algorithme précédent pour obtenir une complexité quadratique (en n^2).

Question 4.2.
Ecrire une méthode `boolean aux(int[]t, int x)` qui étant donné un tableau **trié par ordre croissant** retourne vrai ssi il existe deux i, j distincts tels que $t[i] + t[j] = x$ et qui s'exécute en temps linéaire. C'est à dire, montrez qu'il existe une constante c telle que pour tout tableau t de $n \geq 1$ cases, $m \leq cn$ où m est le nombre d'opération de aux(t). Indication, commencez par tester $t[0]+t[t.length-1]$.

Question 4.3.
Ecrire la méthode `boolean sommeV2(int []t)` qui retourne vrai ssi il existe trois entiers distincts i, j et k tels que $t[i] + t[j] + t[k] = 0$ et qui s'exécute en temps quadratique. C'est à dire, montrez qu'il existe une constante c telle que pour tout tableau t de $n \geq 1$ cases, $m \leq cn^2$ où m est le nombre d'opération de sommeV2(t). Remarque : vous pouvez utiliser une méthode tri(t) que l'on suppose s'exécuter en $\leq c_1 n^2$ opérations.

Exercice 5. Etoile

On considère un groupe de n personnes numérotées de 0 à $n-1$, et on considère que certaines personnes du groupe en admirent d'autres. Plus précisément, on connaît tous les i, j dans $[0, n-1]$ (avec possiblement $i = j$) tels que la personne i admire la personne j (noté $i \to j$). On a pas d'hypothèse particulière sur ce qui se passe entre deux personnes i et j (ils peuvent s'admirer mutuellement, que un seulement admire l'autre, ou que aucun n'admire l'autre). On modélise cette information par un tableau t de $n \times n$ casess, tel que $t[i][j] = true$ ssi $i \to j$. On dit qu'une personne i est une star ssi tout le monde admire i (y compris elle même), et i n'admire personne d'autre.

Par exemple, 1 est une star dans le groupe représenté par le tableau ci-dessous (où la première ligne dénote t[0][0], t[0][1], t[0][2]):

$$\begin{pmatrix} f & t & t \\ f & t & f \\ f & t & f \end{pmatrix}$$

Question 5.1.
Est ce que tout groupe possède une star ? Est ce qu'un groupe peut posséder deux stars ?

Question 5.2.
Ecrire la méthode `boolean verifStar(boolean [][]t, int i)` qui retourne vrai ssi i est une star, et qui s'exécute en temps linéaire. C'est à dire, montrez qu'il existe une constante c_0 telle que pour tout tableau t de $n \times n$ cases et tout i, $m \leq c_0 n$ où m est le nombre d'opération de verifStar(t,i).

Question 5.3.
Ecrire la méthode `boolean existeStar(boolean [][]t)` qui retourne vrai ssi il existe une star dans le groupe représenté par t, et qui s'exécute en temps quadratique. C'est à dire, montrez qu'il existe une constante c_1 telle que pour tout tableau t de $n \times n$ cases, $m \leq c_1 n^2$ où m est le nombre d'opération de existeStar(t).

Question 5.4.
Ecrire la méthode `boolean existeStarV2(boolean [][]t)` qui retourne vrai ssi il existe une star dans le groupe représenté par t, et qui s'exécute en temps linéaire. C'est à dire, montrez qu'il existe une constante c_2 telle que pour tout tableau t de $n \times n$ cases, $m \leq c_2 n$ où m est le nombre d'opération de existeStarV2(t).

Exercice 6. Nono le robot

On considère la fonction échange suivante:

```
void echange (int[]t, int i , int j){
// pre−requis : 0 <= i < t.length et 0 <= j < t.length
// action : echange t[i] et t[j]
}
```

Question 6.1.
Ecrire la fonction echange et donner sa complexité.

Question 6.2.
Donner un algorithme de tri de tableau en ordre croissant n'utilisant que la fonction echange pour modifier le tableau, en essayant d'appeler "echange" le moins de fois possible.

- calculer la complexité de votre algorithme en ne comptant que le nombre d'appels à "echange"
- calculer la complexité de votre algorithme dans le modèle du cours (en ne comptant que les opérations élémentaires)

Question 6.3.
Existe-t-il un algorithme répondant à la question précédente tel que le nombre d'appels à la fonction "echange" dans le plus mauvais cas soit linéaire en fonction de n (où n est longueur du tableau) ?

Question 6.4.
Répondez aux mêmes questions, en remplaçant la fonction "echange" par la fonction "echangeConsecutifs" suivante :

```
void echangeConsecutifs (int[] t, int i ){
// pre−requis : 0 <= i < t.length − 1
// action : echange t[i] et t[i+1]
}
```

On souhaite maintenant remplacer la fonction "echange" par les deux fonctions suivantes :

```
 void nonoGauche (int[] t, int i ){
// pre−requis : 0 <= i < t.length − 1
// action : deplace la suite (t[i],t[i+1]) à gauche de t
..
 }

void nonoDroite (int[] t, int i ){
// pre−requis : 0 <= i < t.length − 1
// action : deplace la suite (t[i],t[i+1]) à droite de t
..
}
```

L'idée est que pour modifier le tableau on dispose d'un robot (qui s'appelle Nono) qui se balade "en dessous" du tableau, qui peut "retirer" du tableau n'importe quelle paire d'éléments consécutifs, et emmener cette paire soit tout à gauche, soit tout à droite. Le robot n'a pas le droit de prendre un seul élément au bord, il doit toujours en prendre exactement 2. Par exemple, si $t = [4, 2, 3, 1, 0]$ et que l'on appelle nonoGauche(t,2), on aura $t = [3, 1, 4, 2, 0]$.

Dans cette partie on supposera pour simplifier que t contient une permutation de $\{0, \ldots, n-1\}$. Remarquez tout d'abord que même avec cette simplification, il n'est pas évident que tout tableau puisse être trié avec un tel robot (cf dernière question de l'exercice). On définit

- le nombre d'inversions d'une permutation (donc d'un tableau t ici) comme $inv(t) = |\{\{i, j\} \text{ tels que } i < j \text{ et } t[i] > t[j]\ \}|$ (autrement dit, $inv(t)$ est le nombre de couples qui ne sont pas dans le bon ordre)
- la signature d'une permutation comme $\sigma(t) = 1$ ssi $inv(t)$ est pair, et -1 sinon

Nous allons voir que le fait que t puisse être trié par le robot est caractérisé par $\sigma(t)$. On admettra pour l'instant que si t est transformé en t' par un appel à nonoDroite ou nonoGauche, alors $\sigma(t') = \sigma(t)$.

Question 6.5.
Donner un algorithme qui, étant donné un tableau t (qui est une permutation) tel que $\sigma(t) = 1$, trie le tableau par ordre croissant, en n'utilisant que les fonctions nonoGauche et nonoDroite. De plus :

- calculer la complexité de votre algorithme en ne comptant que le nombre d'appels à "nonoDroite" et "nonoGauche"
- calculer la complexité de votre algorithme dans le modèle du cours (en ne comptant que les opérations élémentaires)

Question 6.6.
Existe-t-il un algorithme répondant à la question précédente tel que le nombre d'appels aux fonctions "nonoDroite" et "nonoGauche" soit linéaire en fonction de n (où n est longueur du tableau) ?

Question 6.7.
Bonus : prouvez que si t est transformé en t' par un appel à nonoDroite ou nonoGauche, alors $\sigma(t') = \sigma(t)$. En déduire que si $\sigma(t) = -1$, alors il n'est pas possible de trier t en utilisant uniquement "nonoDroite" et "nonoGauche". On observe ainsi que l'hypothèse $\sigma(t) = 1$ supposée ci-dessus pour pouvoir trier le tableau était bien nécessaire.