Contrôle terminal info1 / Semestre 1

R1.01.P2 : Structure de données et algorithmes

Nom du responsable :	**Kamp J-F**
Date du contrôle :	**18/01/2022**
Durée du contrôle :	**2 heures**
Nombre total de pages :	**12 pages**
Impression :	**Recto**
Documents autorisés :	**AUCUN**
Calculatrice autorisée :	**non**
Réponses :	**Répondre sur la copie**

Le soin apporté à la rédaction de la copie entrera dans la notation. En particulier, votre code doit être lisible et indenté en respectant au mieux les conventions de style et de nommage du langage *java*.

Utilisez les brouillons, toute rature sur la copie sera sanctionnée.

Les réponses se font sur la copie.

Merci d'inscrire vos nom, prénom, groupe sur chaque page.

Le barème est donné à titre indicatif.

NOM : **PRENOM :** **Grpe :**

Efficacité (9 points)

Pour cette question voici le récapitulatif du nombre exact d'opérations élémentaires, utilisé en cours et en TDs :

- Une déclaration *int i* vaut 1 opération.
- Une affectation *a = 3* vaut 1 opération.
- Une addition ou multiplication ou division ou soustraction vaut 1 opération.
- Un test de comparaison (<=, <, >=, >, !=, ==) vaut 1 opération.
- Une opération logique (&&, ||, !) vaut 1 opération.
- Une auto-incrémentation *i++* vaut 2 opérations (car équivalent à *i = i + 1*).
- Une auto-décrémentation *i--* vaut 2 opérations (car équivalent à *i = i - 1*).
- Un accès à une case de tableau *tab[i]* vaut 0 opération.
- Un commentaire n'est pas une opération.

On ne comptabilisera NI la déclaration des paramètres, NI l'opération de retour (*return*) si elle existe.

Soit la méthode :

```
/**
 * Rôle à déterminer
 * @param tab un tableau d'entiers
 * @param n le nombre d'entiers dans le tableau
 */
void unAlgo ( int[] tab, int n ) {
      int k, m, tmp;

      for ( int i = 0; i < (n-1); i++ ) {

            m = tab[i];
            k = i;

            for ( int p = (i+1); p < n; p++ ) {

                  if ( tab[p] < m ) {
                        m = tab[p];
                        k = p;
                  }
            }
            tmp = tab[k];
            tab[k] = tab[i];
            tab[i] = tmp;
      }
}
```

NOM : **PRENOM :** **Grpe :**

Question 1 (1 point)

Écrire en une phrase courte ce que fait l'algorithme.

Expliquer son principe en 2 – 3 phrases maximum.

Montrer ce principe sur un exemple de tableau simple et de petite taille.

Réponses :

Question 2 (1 point)

En page 2, écrire sur l'algorithme le nombre d'opérations élémentaires exécutées à chaque instruction. Il n'est pas nécessaire d'écrire le nombre de fois que ces opérations élémentaires sont exécutées (ceci apparaîtra dans vos calculs à la question 3).

Question 3 (3 points)

On s'intéresse uniquement à la boucle interne (`for ( int p = (i+1); p < n; p++ ) {…}`) :

- quel est le pire des cas pour cette boucle interne ?

  Réponse en une phrase courte :

- quel est le nombre de tours (*nbT2*) en fonction de *n* et de *i* de cette boucle interne :

  Réponse :

Calculez *K(n)*, le nombre d'opérations élémentaires exactes en fonction de **n et de i** dans le pire des cas de la boucle interne.

Toutes les étapes de votre calcul doivent apparaître dans votre réponse (attention à la clarté et au soin SVP !).

Réponse :

Question 4 (4 points)

Calculez *F(n)*, le nombre d'opérations élémentaires exactes en fonction de **n** dans le pire des cas de l'algorithme *unAlgo*. Il est conseillé de commencer d'abord par exprimer *F(n)* en fonction de *K(n)*. L'expression finale de *F(n)* ne peut dépendre que de **n**.

En déduire l'ordre de grandeur de la complexité de l'algorithme *unAlgo* dans le pire des cas.

Toutes les étapes de votre calcul doivent apparaître dans votre réponse (attention à la clarté et au soin SVP !).

Réponse :

NOM : **PRENOM :** **Grpe :**

Écriture d'un algorithme (11 points)

Pour cette partie, aucune méthode codée lors des séances de TDs et TPs ne peut être réutilisée.

La *JavaDoc* n'est pas demandée.

Présentation de l'algorithme

L'algorithme de tri de Shell permet de trier un tableau par ordre croissant des valeurs comme beaucoup d'autres algorithmes étudiés en R1.01.P2.

Il ressemble assez bien au tri à bulles sur le principe car il procède également par échanges de contenus de cases de tableau. Mais la différence fondamentale avec le tri à bulles vient d'un échange qui ne se fait pas avec le voisin directement à droite mais avec un voisin plus éloigné qui se trouve à une distance en nombre de cases appelée **gap** (écart).

Étapes pour un gap fixé

Exemple : à la Figure 1, pour *gap = 3* sur un tableau de 11 cases à trier on commence par identifier une sous-suite à partir de la case zéro (*indDeb = 0*) qui se compose des cases 0, 3 (0 + gap = 3), 6 (3 + gap = 6) et 9 (6 + gap = 9) :

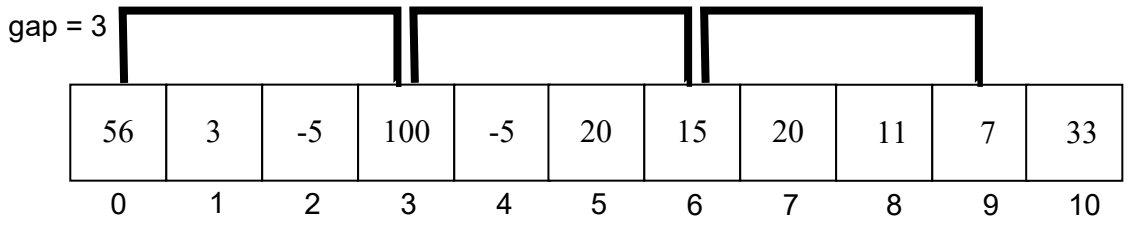


Figure 1 : sous-suite pour *gap=3* et *indDeb = 0*

Sur cette sous-suite, on effectue un tri à bulles complet. L'écart (**gap**) imposera d'abord d'examiner le contenu des cases 0 et 3 puis le contenu des cases 3 et 6 et pour finir le contenu des cases 6 et 9 (au-delà, 9 + gap = 12 et c'est un indice impossible puisque l'on sort du tableau). Cette opération d'échange entre les cases 0, 3, 6 et 9 se poursuit tant que cette sous-suite (0, 3, 6 et 9) n'est pas triée. Au final, nous obtenons pour l'étape *gap = 3* et à partir de la case zéro (*indDeb = 0*), le tableau suivant (en gras les valeurs triées de la sous-suite 0, 3, 6, 9) :

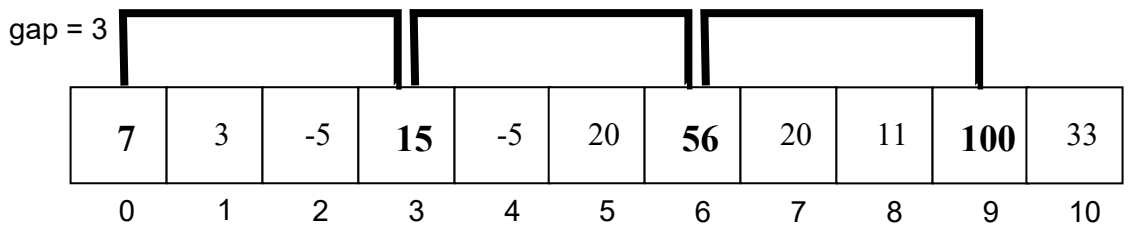


Figure 2 : sous-suite triée pour *gap=3* et *indDeb = 0*

Toujours avec un *gap = 3*, les étapes suivantes consistent à effectuer exactement le même tri à bulles complet mais cette fois en partant de la case un (*indDeb = 1*) puis deux (*indDeb = 2*).

Inutile d'aller au-delà (tel que *indDeb = 3*) puisque l'on retombe sur la même sous-suite que pour *indDeb = 0*. L'enchaînement des étapes pour *gap = 3* est donc terminé.

A la Figure 3, on montre, pour l'exemple, la sous-suite à trier en partant de *indDeb = 1*.

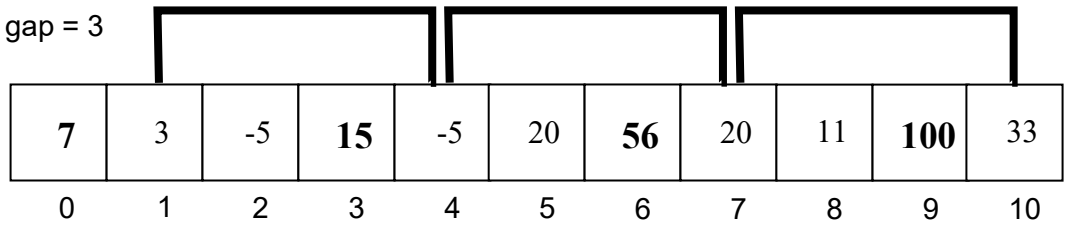


Figure 3. sous-suite à trier pour *gap=3* et *indDeb = 1*

Évidemment, le tableau n'est pas encore complètement trié : seuls les sous-suites avec *gap = 3* sont triées.

L'algorithme complet

L'ensemble des étapes pour un **gap** donné (expliqué au paragraphe précédent) se répète pour des **gaps** de + en + petits.

Par convention (et sans rentrer dans les détails), on supposera un **gap** initialisé à K (entier > 1 et que l'on ne vous demande PAS de déterminer). Après chaque série d'étapes pour un **gap** fixé (voir paragraphe précédent), une nouvelle série recommence avec un nouveau *gap = gap / 2* (*gap* de la série précédente divisé par deux, la division étant entière).

Lorsque pour finir **gap = 1** (les divisions entières successives de *gap* par deux atteindront nécessairement 1), une toute dernière série se déroule avec ce dernier *gap = 1* et le tableau est définitivement trié.

NOM : **PRENOM :** **Grpe :**

Question1 (1 point)

Partant de l'organisation habituelle de l'espace de développement (/ws, /src, /class, /javaDoc), et sachant que la classe qui contient la méthode *triShell* s'appelle *TrisTableau.java* et qu'elle se trouve dans le bon répertoire, écrire la commande de compilation que vous devriez taper dans le répertoire /ws.

Réponse :

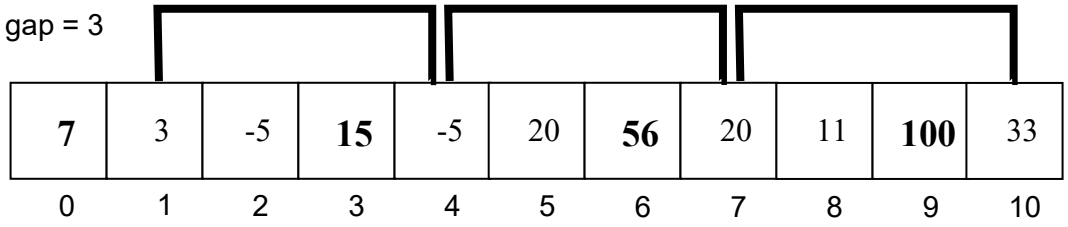
NOM : **PRENOM :** **Grpe :**

Question2 (2 points)

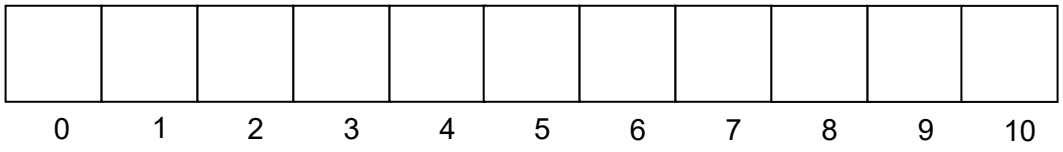
Partant de la Figure 3 (*gap = 3*, *indDeb = 1*), dessinez les différents états du tableau pour arriver au final à un tableau définitivement trié.

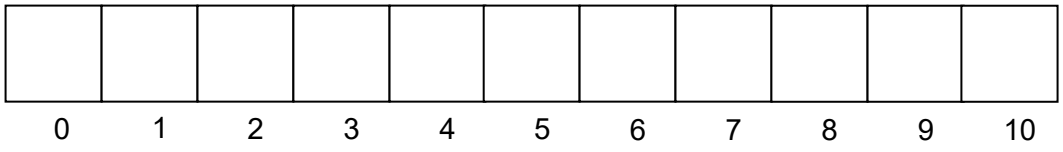
Faire apparaître clairement la sous-suite à trier avec le symbole 

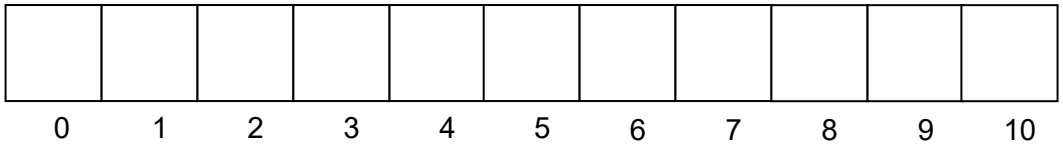
Situation de départ (sous-suite à trier pour *gap=3* et *indDeb = 1*) :



Réponse (à vous de remplir les cases, le nombre d'étapes est volontairement exagéré) :







NOM : **PRENOM :** **Grpe :**

0	1	2	3	4	5	6	7	8	9	10

0	1	2	3	4	5	6	7	8	9	10

0	1	2	3	4	5	6	7	8	9	10

0	1	2	3	4	5	6	7	8	9	10

0	1	2	3	4	5	6	7	8	9	10

0	1	2	3	4	5	6	7	8	9	10

Question3 (4 points)

Écrire la méthode qui effectue l'étape de tri d'une sous-suite connaissant *indDeb* et *gap*. A l'issue de l'exécution de cette méthode la sous-suite doit être forcément triée à partir de *indDeb* : il s'agit donc d'écrire un tri à bulles complet sur la sous-suite.

La signature de la méthode est la suivante (**ne pas traiter les cas d'erreur**) :

```
/**
 * Tri d'une sous-suite connaissant indDeb et gap
 * @param tab le tableau à trier
 * @param n le nombre de valeurs dans le tableau
 * @param indDeb indice à partir duquel commence la sous-suite
 * @param gap distance (écart) entre 2 cases successives de la sous-suite
 */

void trierSousSuite ( int[] tab, int n, int indDeb, int gap )
```

Réponse :

Question4 (4 points)

Ecrire la méthode de tri *triShell* qui effectue un tri par ordre croissant des entiers contenus dans le tableau.

La signature de la méthode est la suivante (**ne pas traiter les cas d'erreur**) :

```
/**
 * Tri complet d'un tableau par ordre croissant des valeurs
 * @param tab le tableau à trier
 * @param n nombre de valeurs dans le tableau
 * @param K la constante qui est la valeur initiale donnée à gap
 */

void triShell ( int[] tab, int n, int K )
```

Cette méthode fait appel à *trierSousSuite* dès qu'il est nécessaire de trier une sous-suite par un tri à bulles pour un *gap* donné et un indice de début de sous-suite donné (*indDeb*).

Réponse :