L1 Informatique et Electronique - SI1: Algorithmique et complexité expérimentale
Première session 2018-2019
2 heures - Tous documents autorisés
Le sujet comporte 4 pages. Le barème est donné à titre indicatif.

QCM (questions 1 à 10, 1 point par question) : noircir les cases correspondant aux bonnes réponses sur le sujet.
Partie rédactionnelle (question 11 à 13, 10 points) : à rédiger sur la copie anonymisée et dérouler les algorithmes sur les pages prévues à cet effet.

Dans le QCM chaque question a exactement une bonne réponse. Une bonne réponse compte pour 1 point et une mauvaise réponse pour -0,5 point.

Pour chaque question, **noircissez** (comme ceci ■ et non comme cela ☒) la case correspondant à la bonne réponse.

☐0	☐0	☐0	☐0	☐0	☐0	☐0	☐0
☐1	☐1	☐1	☐1	☐1	☐1	☐1	☐1
☐2	☐2	☐2	☐2	☐2	☐2	☐2	☐2
☐3	☐3	☐3	☐3	☐3	☐3	☐3	☐3
☐4	☐4	☐4	☐4	☐4	☐4	☐4	☐4
☐5	☐5	☐5	☐5	☐5	☐5	☐5	☐5
☐6	☐6	☐6	☐6	☐6	☐6	☐6	☐6
☐7	☐7	☐7	☐7	☐7	☐7	☐7	☐7
☐8	☐8	☐8	☐8	☐8	☐8	☐8	☐8
☐9	☐9	☐9	☐9	☐9	☐9	☐9	☐9

⟵ **Noircir** les cases correspondant à votre numéro d'étudiant ci-contre. Par sécurité, recopiez le ci-dessous.

Numéro étudiant :

. . . .

Question 1 Si N est la taille des données d'entrée, dans quelle classe de complexité trouvera-t-on les fonctions les plus efficaces?

- ☐ $O(N^2)$
- ☐ $O(1)$
- ☐ $O(log_2(N))$
- ☐ $O(N^3)$
- ☐ $O(2^N)$
- ☐ $O(N \cdot log_2(N))$
- ☐ $O(N)$

Question 2 Si N est la taille des données d'entrée, $O(6N + N^2 + 3N \cdot log_2(N) + 2)$ est égal à:

- ☐ $O(N^2)$
- ☐ $O(1)$
- ☐ $O(N)$
- ☐ $O(2^N)$
- ☐ $O(N^3)$
- ☐ $O(log_2(N))$
- ☐ $O(N \cdot log_2(N))$

Question 3 Si N est la taille du tableau `t`, quelle est la complexité au pire cas, la plus précise, pour la fonction `f` ?

```
void f(int[] t){
  int m= t.length/2;
  int res= 0;
  while (m>0){
    m=m-2;
    res=res+1;
  }
}
```

- ☐ $O(1)$
- ☐ $O(N \cdot log_2(N))$
- ☐ $O(2^N)$
- ☐ $O(log_2(N))$
- ☐ $O(N)$
- ☐ $O(N^2)$
- ☐ $O(N^3)$

Question 4 Si N est la taille du tableau `t`, quelle est la complexité au pire cas, la plus précise, pour la fonction `f` ?

```
void f(int[] t){
  int res=0;
  for(int i=0; i<t.length; i++){
    res=res+t[i];
  }
  for(int j=0; j<t.length; j++){
    res=res+t[j];
  }
}
```

- ☐ $O(N^2)$
- ☐ $O(2^N)$
- ☐ $O(1)$
- ☐ $O(N)$
- ☐ $O(N \cdot log_2(N))$
- ☐ $O(N^3)$
- ☐ $O(log_2(N))$

Question 5 Si N est la taille du tableau `t`, quelle est la complexité au pire cas, la plus précise, pour la fonction `f` ?

```
void f(int[] t){
  for(int i=0; i < t.length; i++){
    for(int j=0; j < 6; j++){
      t[i]=t[i]+t[j]*2;
  }
}
```

- ☐ $O(1)$
- ☐ $O(N \cdot log_2(N))$
- ☐ $O(log_2(N))$
- ☐ $O(2^N)$
- ☐ $O(N^2)$
- ☐ $O(N)$
- ☐ $O(N^3)$

Question 6 Si N est la taille du tableau `t`, quelle est la complexité au pire cas, la plus précise, pour la fonction `f` ?

```
int f(int[] t){
  int j=t.length-1;
  int temp;
  temp= t[j];
  t[j]=t[j-1];
  t[j-1]= temp;
}
```

- ☐ $O(log_2(N))$
- ☐ $O(1)$
- ☐ $O(N)$
- ☐ $O(N^2)$
- ☐ $O(N \cdot log_2(N))$
- ☐ $O(2^N)$
- ☐ $O(N^3)$

Question 7 Quel est le pire cas pour la fonction `g` suivante:

```
boolean g(int[] t1, int[] t2){
  if (t1.length!=t2.length) return false;
  else {
    for (int i=0; i<t1.length;i++){
       if (t1[i]!=t2[i]) return false;
    }
    return true;
  }
}
```

- ☐ `t1={7,6}` et `t2={6,7}`
- ☐ `t1={6,6}` et `t2={7,7}`
- ☐ `t1={6,6,7}` et `t2={6,6}`
- ☐ `t1={6,6,6}` et `t2={6,6}`
- ☐ `t1={6,6}` et `t2={6,7}`
- ☐ `t1={6,6}` et `t2={6,6,7}`
- ☐ `t1={6,6}` et `t2={6,6}`
- ☐ `t1={6,6}` et `t2={6,6,6}`

Question 8 Si N est la valeur de l'entier `i`, quelle est la complexité au pire cas, la plus précise, pour la fonction `f`?

```
int f(int i){
  boolean trouve=false;
  int j=i;
  while(!trouve && j>0){
    if (j*j<=i) { trouve=true;}
     j=j/2;
  }
  return j;
}
```

- ☐ $O(2^N)$
- ☐ $O(N)$
- ☐ $O(log_2(N))$
- ☐ $O(N^2)$
- ☐ $O(N^3)$
- ☐ $O(1)$
- ☐ $O(N \cdot log_2(N))$

Question 9 Quel est le pire cas pour la fonction `g` suivante:

```
void g(int[] t){
  int i=t.length-1;
  while(i<t.length && i>0){
    if (t[i]>=0 && t[i]<t.length){
      i=t[i];
    } else { i=0; }
  }
}
```

- ☐ `t={2,2,2,0}`
- ☐ `t={18,0,3,1}`
- ☐ `t={0,2,0,1}`
- ☐ `t={1,2,3,18}`

Question 10 Si N est la taille du tableau `t`, quelle est la complexité au pire cas, la plus précise, pour la fonction `f` ?

```
void f(int[] t){
  for(int i=0; i < t.length; i++){
    t[i]=t[i]*t[i];
  }
}
```

- ☐ $O(N \cdot log_2(N))$
- ☐ $O(1)$
- ☐ $O(log_2(N))$
- ☐ $O(N^3)$
- ☐ $O(2^N)$
- ☐ $O(N^2)$
- ☐ $O(N)$

Numéro d'anonymat : ....

L1 Informatique et Électronique – Première Session 2018-2019

SI1 : Algorithmique et complexité Expérimentale

2 heures - Documents autorisés

Pour les algorithmes et programmes, on attend de la syntaxe Java. Le barème est donné à titre indicatif. **Il n'est nécessaire de dérouler un algorithme que lorsque cela est demandé explicitement**. Si vos algorithmes/programmes utilisent des fonctions vues en **cours magistral**, il n'est pas nécessaire de redonner leur code, donnez simplement leur signature/entête.

Question 11 (5pts) On propose la fonction `g` dont le code est donné au verso de cette feuille. Pour simplifier, on va étudier la complexité de cette fonction dans le cas où les deux tableaux d'entiers `t1` et `t2` sont même longueur. On appelle N cette longueur.

1. Proposez des valeurs pour les paramètres `t1` et `t2` pour une exécution au pire cas pour `g` avec $N = 1$, $N = 2$ et $N = 3$. Justifiez.
2. Sur les feuilles suivantes, déroulez l'exécution de la fonction `g` sur les valeurs de taille $N = 1$ et $N = 2$.
3. Sur les feuilles suivantes, déroulez l'exécution de la fonction `g` **de façon abstraite** sur ces deux valeurs.
4. Déduisez-en la fonction $f(N)$ qui donne le nombre d'instructions abstraites pour une valeur de N. Justifiez.
5. Donnez la complexité au pire cas de `g`, justifiez.

Question 12 (3pts) Dans cette partie, on considère des tableaux `int[]` dont tous les éléments ont des valeurs comprises entre 0 et 4. On souhaite programmer une fonction `int[] compter(int[] t)` qui pour un tel tableau `t` rend un tableau de taille 5 qui donne pour chaque élément le nombre de fois qu'il apparaît dans `t`. Par exemple si `t={1}` alors `compter(t)` rendra le tableau `{0,1,0,0,0}`. Si `t={1,1,3,0,1,1,3,1,1,4}` alors `compter(t)` rendra le tableau `{1,6,0,2,1}` (soit un 0, six 1, aucun 2, deux 3 et un 4).

1. Donnez le code de la fonction `compter` ;
2. Donnez la complexité au pire cas de `compter`, justifiez ;

Question 13 (2pts) Cette question utilise la fonction `compter` de la question précédente. On considère à nouveau des tableaux `int[]` dont tous les éléments ont des valeurs comprises entre 0 et 4. On souhaite proposer une fonction de tri `void tri(int[] t)` dont la **complexité dans le pire cas sera linéaire** pour ce type de tableaux. A partir de `t`, on peut utiliser la fonction `compter` pour comptabiliser les éléments présents dans `t`, puis, à partir de ce décompte reconstruire un tableau `t` trié. Par exemple, on a vu que si `t={1,1,3,0,1,1,3,1,1,4}` alors `compter(t)` rendra le tableau `{1,6,0,2,1}`. A partir de `{1,6,0,2,1}`, on peut modifier `t` en `{0,1,1,1,1,1,1,3,3,4}` (une fois l'entier 0, six fois l'entier 1, 0 fois l'entier 2, etc.) qui est une version triée de `t`.

1. Donnez le code de la fonction `tri` ;
2. Justifiez la complexité linéaire dans le pire des cas pour votre fonction `tri`.

```
boolean g(int[] t1, int[] t2){
  for (int i=0; i<t1.length; i++){
      boolean t=false;
      for (int j=0;j<t2.length; j++){
          if (t1[i]==t2[j]){
             t=true;}}
      if (!t) return false;
  }
  return true;}
```

```
boolean g(int[] t1, int[] t2){
  for (int i=0; i<t1.length; i++){
      boolean t=false;
      for (int j=0;j<t2.length; j++){
          if (t1[i]==t2[j]){
             t=true;}}
      if (!t) return false;
  }
  return true;}
```