Nom :

Prénom :

Groupe :

EA4 – Éléments d'algorithmique
Partiel du 28 février 2018 – Sujet B
Durée : 2 heures

Aucun document autorisé
Appareils électroniques éteints et rangés

Exercice 1 :
On considère la permutation $\sigma = 7\ 1\ 4\ 2\ 5\ 3\ 6$. Calculer son inverse.

Donner une décomposition de σ en produit de transpositions.

Exercice 2 :
Décrire le déroulement de l'algorithme de Karatsuba pour calculer le produit 1011×1102, en représentant en particulier l'arbre des appels récursifs. Prendre le produit de chiffres comme cas de base.

Exercice 3 :

Soit `T` le tableau suivant :

1	23	37	15	41	3	8	14

Appliquer l'algorithme de tri fusion (*MergeSort*) à `T`. Combien de comparaisons d'éléments sont effectuées (exactement) ?

Appliquer l'algorithme de tri rapide (*QuickSort*) à `T` dans sa version simple (pas en place, avec `T[0]` comme pivot). Combien de comparaisons d'éléments sont effectuées (exactement) ?

Exercice 4 :

On considère l'algorithme suivant :

```
def F(n) :
    return 1 if n < 4 else 2 * F(n-1) + F(n-4)
```

Soit $C(n)$ le nombre d'opérations arithmétiques sur des entiers effectuées lors de l'exécution de `F(n)`. Donner une définition de $C(n)$ par récurrence.

En déduire que $C(n)$ est croissante, puis que $C(n) \in \Omega(2^{n/4})$.

Proposer un algorithme `Fbis(n)` calculant la même valeur que `F(n)` de manière plus efficace.

Quel est l'ordre de grandeur du nombre d'opérations arithmétiques sur des entiers effectuées par `Fbis(n)` ?

Est-ce une mesure pertinente de sa complexité en temps ?

Exercice 5 :

Dans cet exercice, on représente des ensembles par des tableaux *sans doublon*, et on considère la fonction suivante :

```
def foo(E, F) :
    res = []
    for e in E :
        if e not in F : res.append(e)
    return res
```

Que calcule la fonction `foo`, si `E` et `F` sont deux tels tableaux ?

Indiquer un choix d'opération(s) élémentaire(s) pertinent pour évaluer sa complexité (en temps).

Quel est l'ordre de grandeur de sa complexité en temps ? Justifier.

Supposons maintenant que les ensembles sont représentés par des tableaux *triés* (et toujours sans doublon). Proposer un algorithme le plus efficace possible pour effectuer le même calcul et indiquer sa complexité.

Exercice 6 :

On s'intéresse au problème suivant : étant donné une liste `L` de nombres (non nécessairement entiers) de longueur n, déterminer le nombre d'éléments distincts dans `L`.

Décrire un algorithme naïf permettant de résoudre ce problème sans modifier la liste `L`, et avec mémoire auxiliaire constante.

Quel est l'ordre de grandeur de la complexité en temps de cet algorithme ? Justifier.

Comment résoudre ce problème avec une complexité strictement meilleure ? Laquelle ?

Exercice 7 :

On considère un tableau `T` de n entiers distincts *circulairement trié*, c'est-à-dire tel que, pour un certain indice k (inconnu), `T[`k`:] + T[:`k`]` est trié en ordre croissant.

Étant donné une position i de `T`, comment tester *en temps constant* si $i > m$, où m est la position (inconnue *a priori*) du maximum de `T` ?

Décrire un algorithme aussi efficace que possible pour déterminer la position du maximum de `T`.

Quelle est sa complexité ?