Étude des algorithmes

Exercice 1. On s'intéresse à l'algorithme ci-dessous, qui prend en entrée un entier naturel n et retourne $\sum_{k=0}^{n} k$:

```
def somme(n):
    s=0
    for i in range(n+1):
        s=s+i
    return(s)
```

1. Déterminer une précondition et une postcondition appropriées.
2. Montrer que l'algorithme termine.
3. Étudier la correction de l'algorithme.

Exercice 2. On s'intéresse à l'algorithme suivant, qui prend en entrée une liste d'entiers `T` et un entier `elt` et teste si `elt` appartient à `T` (en s'arrêtant dès que `elt` a été trouvé) :

```
def recherche(T,elt):
    i=0
    n=len(T)
    trouve=False
    while i<n and not trouve:
        trouve= T[i]==elt
        i=i+1
    return(trouve)
```

1. Déterminer une précondition et une postcondition appropriées.
2. Montrer que l'algorithme termine.
3. Étudier la correction de l'algorithme.

Exercice 3. On donne ci-dessous un algorithme de recherche d'un maximum dans un tableau de nombres flottants T, en utilisant la méthode du candidat :

```
def maximum(T):
    max=T[0]
    n=len(T)
    for i in range(1,n):
        if T[i]>max:
            max=T[i]
    return(max)
```

Proposer des spécifications d'entrée et de sortie, puis étudier la terminaison et la correction de cet algorithme.

Exercice 4. On considère l'algorithme de tri bulle, qui prend en entrée un tableau T :

```
def tribulle(T):
    n=len(T)
    for i in range(0,n-1):
        for j in range(0,n-1-i):
            if T[j] > T[j+1]:
                T[j],T[j+1]=T[j+1],T[j]
    return(T)
```

Soit n la taille du tableau T et t_n le tests effectués par l'algorithme dans le pire des cas. Déterminer une majoration asymptotique de t_n.

Exercice 5. On rappelle le principe de l'algorithme de tri fusion : soit T un tableau à trier de taille n. Si $n = 0$ ou $n = 1$, T est déjà trié. Sinon, on partage le tableau en deux sous-tableaux de tailles environ égales. On trie récursivement chaque sous-tableau, puis on fusionne les sous-tableaux triés obtenus.
L'algorithme de fusion associé prend en paramètres deux tableaux de nombres (déjà triés) T_1 et T_2 et retourne un tableau T contenant les éléments de T_1 et T_2 triés.
On propose les implémentations suivantes de ces deux algorithmes :

```
def fusion(T1,T2):
    T=[]
    while not(T1==[]) and not(T2==[]):
        if T1[0]<=T2[0]:
             T=T+[T1[0]]
             T1=T1[1:]
        else:
            T=T+[T2[0]]
            T2=T2[1:]
    if T1==[]:
        T=T+T2
    else:
        T=T+T1
    return(T)

def trifusion(T):
    if len(T)<=1:
        return(T)
    else:
        n=len(T)
        p=int(n/2)
        T1=T[:p]
        T2=T[p:]
        return(fusion(trifusion(T1),trifusion(T2)))
```

On note $N(n)$ le nombre de tests effectué par `trifusion(T)` dans le pire des cas.

1. On note n_1 et n_2 les tailles respectives des tableaux T_1 et T_2. Dans le pire des cas, quel est le nombre de tests effectué par `fusion(T1,T2)` ?
2. (a) Que valent $N(0)$ et $N(1)$?
   (b) Soit $p \in \mathbb{N}^*$, justifier qu'on a $N(2^p) \leqslant 2N(2^{p-1}) + 4 \times 2^p$.
   (c) Soit $p \in \mathbb{N}$, déterminer un majorant de $\dfrac{N(2^p)}{2^p}$ ne dépendant que de p.
   (d) Pour $n \in \mathbb{N}^*$ quelconque, en déduire une majoration asymptotique de $N(n)$.