Complexité d'algorithmes

1. Donnez, en fonction du paramètre B, le nombre d'affectations, additions, multiplications, comparaisons effectuées par l'algorithme `puissance` :

Algorithme 1 : puissance(**d** A : entier, **d** B : entier, **r** R : entier)

```
Données : A, B ∈ ℕ
Résultat : R = A^B
Variable X, R ∈ ℕ;
début
  X ← 0 ; R ← 1;
  tant que X < B faire
    R ← R * A ; X ← X+1
  fin tq
fin algorithme
```

2. Quel est, en fonction de N, le nombre d'affectations (lignes 1 et 2) effectuées par l'algorithme ci-dessous ?
(*) Même question en comptant à présent les affectations des lignes 1 et 2, et aussi les affectations « cachées » effectuées par les itérations « Pour » ?

```
  début
1   R ← 0 ;
    pour I de 1 à N faire
      pour J de 1 à I faire
2       R ← R + 1 ;
      fin pour
    fin pour
  fin algorithme
```

3. Quels sont les nombres minimum et maximum d'itérations exécutées par l'algorithme 2 sur la donnée T. Quel est le pire des cas pour cet algorithme. Dans ce pire des cas, combien de fois l'agorithme compare-t-il un élément de T avec e ? Donnez l'ordre de grandeur de la complexité de cet algorithme. Que calcule-t-il ?

Algorithme 2 : Truc(**d** T : Tableau d'entier, **d** e : entier) : entier

```
Données : T un tableau d'entier trié en ordre croissant ; e un entier
Résultat : entier ...
Variable I, J : entier;
début
  I ← 0 ;
  tant que I < taille(T) et T[I] < e faire
    I ← I + 1
  fin tq
  J ← I;
  tant que J < taille(T) et T[J] = e faire
    J ← J + 1
  fin tq
  renvoyer J − I;
fin algorithme
```

4. Donnez un algorithme minimisant le nombre de multiplications pour le problème :

Données : X un entier et $a_0, a_1, \ldots, a_n$ n+1 entiers correspondant aux coefficients d'un polynôme ; ces coefficients sont représentés par un tableau $a[0..n]$ d'entiers
Résultat : R est la valeur du polynôme au point X, $R = a_nX^n + a_{n-1}X^{n-1} + \ldots + a_2X^2 + a_1X + a_0$

5. (*) En vous inspirant de l'algorithme « Multiplication Russe » du cours, écrivez un algorithme qui étant donné 2 entiers $A, B \in \mathbb{N}$ calcule $R = A^B$. Donnez en fonction du paramètre B l'ordre de grandeur de la complexité de votre algorithme. Comparez-la à celle de l'algorithme 1.

6. Étant donné un tableau de `n` entiers, on cherche à savoir si l'un de ses éléments est égal à la somme des `n-1` autres éléments.

Ex : Dans le tableau

3	8	-6	3	-1	9

le $2^{ème}$ élément de valeur 8 est égal à la somme des 5 autres.

Une première méthode pour ce problème consiste pour chaque élément du tableau à calculer la somme des autres éléments et à vérifier si elle est égale à l'élément. Écrivez cet algorithme. Donnez sa complexité.

On peut améliorer, au sens de la complexité, l'algorithme précédent. Écrivez ce deuxième algorithme et donnez sa complexité.

7. Écrivez un algorithme `plusLgPrefSuff` qui étant donné un tableau $T[1..n]$ renvoie le plus grand entier $k \in [1, n-1]$ tel que le sous–tableau $T[1..k]$ est à la fois préfixe et suffixe de T. Si un tel sous–tableau n'existe pas l'algorithme doit renvoyer 0. [1]

   Par exemple pour le tableau $T_1 =$ | 1 | 4 | 2 | 4 | 5 | 1 | 4 | 2 | 4 | le plus grand sous-tableau préfixe et suffixe (différent de T_1) est | 1 | 4 | 2 | 4 |. L'algorithme renvoie donc 4.

   Pour résoudre ce problème, vous identiefierez un sous–problème que vous spécifierez. Vous en donnerez un algorithme et évaluerez sa complexité. Donnez alors la complexité de votre algorithme `plusLgPrefSuff`.

8. Soit le problème :

   Données : un tableau $T[1 \dots n]$ de valeurs prises dans $\mathbb{Z}$
   Résultat : $SomMax$: $SomMax = 0$ si tous les éléments de T sont négatifs, sinon $SomMax$ est la somme maximale des sous-Tableaux de T, c'est à dire $SomMax$ est la valeur max de $S(a, b)$ où $S(a, b) = \Sigma_{i=a}^{b} T[i]$

   Exemple : pour le tableau ci-dessous, la somme max est $SomMax = S(3, 11) = 190$.

1	*2*	**3**	*4*	*5*	*6*	*7*	*8*	*9*	*10*	**11**	*12*	*13*	*14*	*15*
31	-41	59	26	-53	58	97	-93	-23	84	35	-98	-80	-72	-85

   (a) **Algorithme brutal**. Donnez la complexité de l'algorithme suivant :

```
début
    SomMax ← 0 ;
    pour i = 1 à n faire
        pour j = i à n faire
            S ← 0
            pour k = i à j faire
                S ← S + T[k]
            fin pour
            SomMax ← Max(S, SomMax) ;
        fin pour
    fin pour
fin algorithme
```

   (b) **Un peu plus malin** : comment améliorer simplement l'algorithme précédent ? Complexité ?

   (c) **Encore un peu plus malin** : approche "Diviser et Résoudre". Pour trouver la sous-somme maximale d'un tableau $T[d \dots f]$, on calcule les sous-sommes maximales des sous-tableaux $T[d \dots (d+f)\ div\ 2]$ et $T[1+(d+f)\ div\ 2 \dots f]$. Où peut alors se situer la sous-somme maximale du tableau $T[d \dots f]$? En déduire un algorithme. Donnez sa complexité.

   (d) **Toujours plus malin.** Le 4ème algorithme maintient l'invariant suivant :
   « A l'itération i, $SomMax$ est le Max des $S(a, b)$ pour $1 \leq a \leq b \leq i$ et $SomMaxDroite$ est le Max des $S(a, i)$ pour $1 \leq a \leq i$. »
   Que faire pour maintenir l'invariant tout en faisant progresser l'indice i ? En déduire un algorithme optimal.

1. On pose $k \neq n$ car sinon il n'y aurait pas de problème. En effet pour tout tableau T de taille n, le sous–tableau $T[1..n]$, c'est à dire T, est à la fois préfixe et suffixe de T.