Révisions

Donnez un algorithme pour les problèmes suivants :

Algorithme 1 : recherche(**d** T : tableau d'entier, **d** x : entier, **r** present : booléen, **r** PP : entier)

Données : T 1 tableau d'entiers ; x un entier
Résultat : Si T contient un élément de valeur x, *present* est `true`, sinon *present* est `false` et PP est un des éléments de T les plus proches de x, c'est à dire $PP \in T$ et $\forall i \in [0..taille(T)[, |x - PP| \leq |x - T[i]|$

Algorithme 2 : plusPetitEcart(**d** T : tableau d'entier) : entier

Données : T : tableau d'entier
Résultat : renvoie le plus petit écart entre 2 éléments distincts de T, autrement dit $min\{|T[i] - T[j]|$ avec $i, j \in [0, taille(T)[$ et $i \neq j\}$

Algorithme 3 : LignesEgales ?(**d** T : Tableau) :Booléen

Données : $T[0...n-1, 0...n-1]$ un tableau de n lignes et n colonnes et composé de 0 et de 1
Résultat : renvoie Vrai si et seulement si T possède 2 lignes identiques, c'est à dire ssi $\exists i, j \in [0...n[$ tels que $i \neq j$ et $\forall k \in [0...n[, \ T[i,k] = T[j,k]$

Algorithme 4 : InsérerTabTrié(**dr** T : tableau d'entier, **d** m : entier, **d** e : entier)

Données : $m < taille(T) - 1$, le sous-tableau $T[0..m]$ est trié $\nearrow$; le sous-tableau $T[m+1..taille(T)[$ est *"vide"*
Résultat : modifie T en y insérant e parmi les m premiers éléments : le sous-tableau $T[0..m+1]$ est trié

Preuve d'arrêt

Algorithme 5 : BB(**d** N : entier, **r** f : entier)

Données : $N \in \mathbb{N}^*$
Résultat : $f = ?$
Variable : $n \in \mathbb{N}$

début
  $n \leftarrow N$; $f \leftarrow 7!$;
  tant que $n \neq 7$ **faire**
    si $n > 7$ **alors** $f \leftarrow f \times n$; $n \leftarrow n - 1$
    sinon
      $n \leftarrow n + 1$; $f \leftarrow f/n$
    fin si
  fin tq
fin algorithme

Algorithme 6 : AA(**d** n : entier) : entier

Données : $n \in \mathbb{N}$
Résultat : Renvoie ? ? ?
Variables : $i, r, x \in \mathbb{N}$

début
  i ← 0 ; r ← 1 ; x ← 0 ;
  tant que $i \leqslant n$ **faire**
    i ← i + 1 ; r ← r + x - i ; x ← x + 3;
  fin tq
  renvoyer r
fin algorithme

Algorithme 7 : PGCD(**d** N, M : entier) : entier

Données : $N, M \in \mathbb{N}^*$
Résultat : renvoie $pgcd(N, M)$
Variables : $n, m \in N$

début
  $n \leftarrow N$; $m \leftarrow M$;
  tant que $n \neq 0$ `et` $m \neq 0$ **faire**
    si $m > n$ **alors** $m \leftarrow m - n$
    sinon
      $n \leftarrow n - m$
    fin si
  fin tq
  renvoyer $m + n$
fin algorithme

Algorithme 8 : gcd(**d** n : entier, **d** p : entier) :entier

Données : $n, p \in \mathbb{N}, n \neq 0$ *ou* $p \neq 0$
Résultat : renvoie le pgcd de n et p

début
  si `p=0` **alors**
    renvoyer `n`
  sinon
    renvoyer `gcd(p,n mod p)`
  fin si
fin algorithme

1. Établir la preuve d'arrêt de l'algorithme 5. Que calcule cet agorithme ?
2. L'algorithme 6 s'arrête-t-il ? Pourquoi ?
3. On considère l'algorithme 7. Dites pour chacune des expressions suivantes si leur valeur est dans $\mathbb{N}$ et, dans l'affirmative, si elle décroît strictement à chaque itération. m ; n ; $m - n$; $n - m$; $|m - n|$; $m + n$; $m * n$
   Cet algorithme s'arrête-t-il ?
4. L'algorithme 8 s'arrête-t-il ? Pourquoi ?

Propriété invariante

1. Montrez que l'algorithme 12 s'arrête, démontrez les invariants $A = 3(X - C) + 1$; $B = 3(X - C)^2$.
   On admet sans le montrer l'invariant $Z = (X - C)^3$. Que calcule alors cet algorithme ?

2. Parmi les égalités suivantes, quelles sont celles vérifiées par l'itération de l'algorithme 10 : $(r_1 = i)$, $(r_1 = i-1)$, $(r_1 = fib(i))$, $(r_2 = r_1 + 1$ si $i \neq 1)$, $(r_2 = fib(i+1))$? En déduire la valeur à renvoyer pour que l'algorithme soit correct. Rappel fib(0)=0, fib(1)=1 et fib(n)=fib(n-1)+fib(n-2) pour $n > 1$.
3. Faites une trace de l'algorithme 7 pour les données $M = 48, N = 30$. Pour chaque itération calculez l'ensemble des entiers divisant à la fois m et n. Que constatez-vous ? Quelle propriété, faisant intervenir l'ensemble des diviseurs communs à N et M, semble invariante pour cette itération ?
4. Faites la trace d'exécution de l'algorithme 6 pour $n = 5$. Donnez un invariant liant les valeurs des variables x et i et un invariant liant les valeurs des variables r et i. Prouvez ces 2 invariants. En déduire la valeur de la variable r à la fin du tant que. Que calcule l'algorithme 6 ?
5. Écrivez sous forme d'inégalités, l'égalité $I = \lfloor\sqrt{N}\rfloor$.
   Faites une trace de l'algorithme 11 pour $N = 18$. Donnez un invariant liant les valeurs de Z et I, un autre liant R et I, un autre liant P et I. Comparez P et N. En déduire un encadrement de N en fin d'itération. Prouvez alors que l'algorithme 11 est correct.
6. (*) Montrez en donnant un invariant que l'algorithme 5 est correct.
7. Algorithme 13. À l'aide de traces, trouvez une relation liant les valeurs des variables V et i, puis celles de R et i. Prouvez ces invariants. Que calcule cet algorithme ?

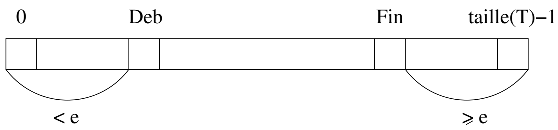
Écriture d'algorithme et de preuve

1. Donnez un algorithme calculant le produit de 2 entiers par simples additions successives. Vous utiliserez un `Tant Que` pour votre itération. Faites-en la preuve.
2. On cherche un algorithme pour le problème suivant :

   Algorithme 9 : rechDichoVersion2(**d** T : tableau d'entier,**d** e : entier, **r** present : booléen, **r** ind : entier)

   Données : T un tableau d'entiers triés ;$e \in \mathbb{N}$
   Résultat : `present` =Vrai si et seulement si $e \in T$; Si `present` alors `ind`= $\min\{i \in [0 \ldots taille(T)[: T[i] = e\}$

   Modifiez l'algorithme de recherche dichotomique vu en cours pour avoir l'invariant suivant :



3. On cherche un algorithme `plusProche` qui étant donné T un tableau d'entiers trié par ordre croissant et e un entier renvoie un des éléments de T les plus proches de e.
4. Un paquet de jeu de cartes (représenté par un tableau d'entiers (différents 2 à 2)) a été trié, puis coupé une fois (coupe non vide). On obtient par exemple le tableau

3	4	5	6	7	9	1	2

   Écrivez un algorithme qui trouve par dichotomie le lieu de coupe. Sur l'exemple le lieu de coupe est situé entre les indices 5 et 6, on convient de renvoyer l'indice 6.

Algorithme 10 : f(**d** n : entier) :entier
Données : n un entier strictement positif
Résultat : Renvoie $fib(n)$
Variables : $i \in \mathbb{N}, r_1 \in \mathbb{N}, r_2 \in \mathbb{N}, r_3 \in \mathbb{N}$;
début
  $i \leftarrow 0$; $r_1 \leftarrow 0$; $r_2 \leftarrow 1$;
  tant que $i < n$ **faire**
    $r_3 \leftarrow r_1$; $r_1 \leftarrow r_2$; $r_2 \leftarrow r_3 + r_2$; $i \leftarrow i + 1$;
  fin tq
  renvoyer ????;
fin algorithme

Algorithme 11 : RacEnt(**d** N :entier,**r** I :entier)
Données : $N \in \mathbb{N}$
Résultat : $I = \lfloor\sqrt{N}\rfloor$
Variables : $Z, R, P \in \mathbb{N}$
début
  $I \leftarrow 0$; $P \leftarrow 0$; $Z \leftarrow 1$; $R \leftarrow 1$;
  tant que $R \leq N$ **faire**
    $I \leftarrow I + 1$; $Z \leftarrow Z + 2$; $P \leftarrow R$; $R \leftarrow R + Z$;
  fin tq
fin algorithme

Algorithme 12 : CC(**d** X :entier, **r** Z :entier)
Données : $X \in \mathbb{N}$
Résultat : $Z \in \mathbb{N}$
Variables : $A, B, C \in \mathbb{N}$
début
  $A \leftarrow 1$; $B \leftarrow 0$; $C \leftarrow X$; $Z \leftarrow 0$;
  tant que $C > 0$ **faire**
    $Z \leftarrow Z + A + B$; $B \leftarrow B + 2 \times A + 1$;
    $A \leftarrow A + 3$; $C \leftarrow C - 1$;
  fin tq
fin algorithme

Algorithme 13 : g(**d** n : entier) :entier
Données : $n \in \mathbb{N}^*$
Résultat : Renvoie l'entier ? ? ?
Variables : $R,\ V,\ i \in \mathbb{N}$;
début
  $R \leftarrow 1; V \leftarrow 1; i \leftarrow 1;$
  tant que $i < n$ **faire**
    $V \leftarrow 2 * V$; $R \leftarrow R + V$; $i \leftarrow i + 1$;
  fin tq
  renvoyer R;
fin algorithme

Complexité d'algorithmes

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

Algorithme 14 : 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 15 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 15 : 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. Soit T un tableau d'entiers. On cherche dans T la longueur du plus grand sous-tableau trié par ordre croissant. Formellement on cherche à calculer `lgSousTabTrié(T)` défini par :

$$max(\{j-i+1 : i \in [0 \ldots taille(T)-1], j \in [i \ldots taille(T)-1] \; et \; T[i] \leqslant T[i+1] \leqslant T[i+2] \leqslant \cdots \leqslant T[j-1] \leqslant T[j]\})$$

Par exemple si T est le tableau

13	5	19	14	16	19	8

`lgSousTabTrié(T)=3` car

14	16	19

est le plus long sous tableau trié par ordre croissant de T.

(a) Un premier algorithme pour ce problème est :

Algorithme 16 : lgSousTabTrié1(**d** T : tableau d'entiers) : entier

```
Données : T un tableau d'entiers
Résultat : Renvoie la longueur d'un plus long sous-tableau trié par ordre croissant de T
Variables : i, j, lgMax ∈ ℕ;
début
    lgMax ← 0 ;
    pour i de 0 à taille(T)-1 faire
        j ← i+1 ;
        tant que j⩽taille(T)-1 et T[j-1] ⩽ T[j] faire
            si lgMax < j-i+1 alors
                lgMax ← j-i+1 ;
            fin si
            j ← j+1;
        fin tq
    fin pour
    renvoyer lgMax;
fin algorithme
```

Quel est le meilleur des cas pour cet algorithme ? Quelle est la complexité dans le meilleur des cas ?
Quel est le pire des cas pour cet algorithme ? Quelle est la complexité dans le pire des cas ?

(b) Écrivez un second algorithme pour `lgSousTabTrié` qui améliore significativement la complexité dans le pire des cas de `lgSousTabTrié1`. Quelle est cette complexité dans le pire des cas ? Vous donnerez des invariants significatifs vérifiés par votre algorithme.

6. 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.