Alain Troesch
Cours d'informatique, MPSI 4
Lycée Louis-Le-Grand (Paris)
Année scolaire 2015/2016

Informatique – Chapitre 5

Complexité

I. Introduction

- Un algorithme qui se termine en théorie, mais répond au bout de 1000 ans n'est d'aucune utilité !

I. Introduction

- Un algorithme qui se termine en théorie, mais répond au bout de 1000 ans n'est d'aucune utilité !
- Lancer un algorithme sur de grandes données est plus long ! On évalue le temps de réponse en fonction de la taille des données.

I. Introduction

- Un algorithme qui se termine en théorie, mais répond au bout de 1000 ans n'est d'aucune utilité !
- Lancer un algorithme sur de grandes données est plus long ! On évalue le temps de réponse en fonction de la taille des données.
- On compare la qualité des algorithmes pour des données de grande taille (comportement asymptotique)...

I. Introduction

- Un algorithme qui se termine en théorie, mais répond au bout de 1000 ans n'est d'aucune utilité !
- Lancer un algorithme sur de grandes données est plus long ! On évalue le temps de réponse en fonction de la taille des données.
- On compare la qualité des algorithmes pour des données de grande taille (comportement asymptotique)...
- ...mais suivant l'utilisation qu'on veut en faire, ce n'est pas toujours pertinent.

I. Introduction

- Un algorithme qui se termine en théorie, mais répond au bout de 1000 ans n'est d'aucune utilité !
- Lancer un algorithme sur de grandes données est plus long ! On évalue le temps de réponse en fonction de la taille des données.
- On compare la qualité des algorithmes pour des données de grande taille (comportement asymptotique)...
- ...mais suivant l'utilisation qu'on veut en faire, ce n'est pas toujours pertinent.
- D'ailleurs certains algorithmes mauvais asymptotiquement sont meilleurs sur les petites tailles.

I. Introduction

- Un algorithme qui se termine en théorie, mais répond au bout de 1000 ans n'est d'aucune utilité !
- Lancer un algorithme sur de grandes données est plus long ! On évalue le temps de réponse en fonction de la taille des données.
- On compare la qualité des algorithmes pour des données de grande taille (comportement asymptotique)...
- ...mais suivant l'utilisation qu'on veut en faire, ce n'est pas toujours pertinent.
- D'ailleurs certains algorithmes mauvais asymptotiquement sont meilleurs sur les petites tailles.
- On peut alors avoir intérêt à combiner plusieurs algorithmes (exemple des tris).

Rappel des notations asymptotiques (pour $u_n, v_n > 0$) :

Définition 1.1 (Comparaisons asymtotiques)

- Équilalence : $u_n \sim v_n \iff \frac{u_n}{v_n} \longrightarrow 1$. Autrement dit, (u_n) et (v_n) ont même ordre de grandeur asymptotique.

Rappel des notations asymptotiques (pour $u_n, v_n > 0$) :

Définition 1.1 (Comparaisons asymtotiques)

- Équilalence : $u_n \sim v_n \iff \frac{u_n}{v_n} \longrightarrow 1$. Autrement dit, (u_n) et (v_n) ont même ordre de grandeur asymptotique.
- Négligeabilité : $u_n = o(v_n) \iff \frac{u_n}{v_n} \longrightarrow 0$.

Rappel des notations asymptotiques (pour $u_n, v_n > 0$) :

Définition 1.1 (Comparaisons asymtotiques)

- Équilalence : $u_n \sim v_n \iff \frac{u_n}{v_n} \longrightarrow 1$. Autrement dit, (u_n) et (v_n) ont même ordre de grandeur asymptotique.
- Négligeabilité : $u_n = o(v_n) \iff \frac{u_n}{v_n} \longrightarrow 0$.
- Dominance : $u_n = O(v_n) \iff \frac{u_n}{v_n}$ est borné.

Rappel des notations asymptotiques (pour $u_n, v_n > 0$) :

Définition 1.1 (Comparaisons asymtotiques)

- Équilalence : $u_n \sim v_n \iff \frac{u_n}{v_n} \longrightarrow 1$. Autrement dit, (u_n) et (v_n) ont même ordre de grandeur asymptotique.
- Négligeabilité : $u_n = o(v_n) \iff \frac{u_n}{v_n} \longrightarrow 0$.
- Dominance : $u_n = O(v_n) \iff \frac{u_n}{v_n}$ est borné.
- $u_n = \Omega(v_n)$ s'il existe $m > 0$ tel que $\frac{u_n}{v_n} \geqslant m$ à partir d'un certain rang

Rappel des notations asymptotiques (pour $u_n, v_n > 0$) :

Définition 1.1 (Comparaisons asymtotiques)

- Équilalence : $u_n \sim v_n \iff \frac{u_n}{v_n} \longrightarrow 1$. Autrement dit, (u_n) et (v_n) ont même ordre de grandeur asymptotique.
- Négligeabilité : $u_n = o(v_n) \iff \frac{u_n}{v_n} \longrightarrow 0$.
- Dominance : $u_n = O(v_n) \iff \frac{u_n}{v_n}$ est borné.
- $u_n = \Omega(v_n)$ s'il existe $m > 0$ tel que $\frac{u_n}{v_n} \geqslant m$ à partir d'un certain rang
- $u_n = \Theta(v_n)$ s'il existe $m, M > 0$ tels que $m \leqslant \frac{u_n}{v_n} \leqslant M$ à partir d'un certain rang.

Interprétation des comparaisons asymptotiques

- O mesure les performances d'un algorithme

Interprétation des comparaisons asymptotiques

- O mesure les performances d'un algorithme
- Ω mesure les limitations d'un algorithme

Interprétation des comparaisons asymptotiques

- O mesure les performances d'un algorithme
- Ω mesure les limitations d'un algorithme
- Θ permet donc de classer de façon précise la complexité sur une échelle : un algorithme dont la complexité est en $\Theta(n^2)$ aura de façon effective un temps de réponse de l'ordre de n^2, ni plus lent, ni plus rapide.

II. Complexité en temps (modèle à coûts fixes)

II-1 Première approche

Problème : évaluer le temps de réponse d'un algorithme

Chacune des opérations suivantes a une certaine durée de traitement :

II. Complexité en temps (modèle à coûts fixes)

II-1 Première approche

Problème : évaluer le temps de réponse d'un algorithme

Chacune des opérations suivantes a une certaine durée de traitement :

- l'affectation

II. Complexité en temps (modèle à coûts fixes)

II-1 Première approche

Problème : évaluer le temps de réponse d'un algorithme

Chacune des opérations suivantes a une certaine durée de traitement :

- l'affectation
- les comparaisons

II. Complexité en temps (modèle à coûts fixes)

II-1 Première approche

Problème : évaluer le temps de réponse d'un algorithme

Chacune des opérations suivantes a une certaine durée de traitement :

- l'affectation
- les comparaisons
- les opérations sur des données numériques.

II. Complexité en temps (modèle à coûts fixes)

II-1 Première approche

Problème : évaluer le temps de réponse d'un algorithme

Chacune des opérations suivantes a une certaine durée de traitement :

- l'affectation
- les comparaisons
- les opérations sur des données numériques.

La complexité en temps est obtenue en sommant tous les temps d'execution des différentes opérations effectuées lors de l'exécution d'un algorithme.

Coût des opérations

- On appelle coût d'une opération le temps d'exécution de cette opération.

Coût des opérations

- On appelle coût d'une opération le temps d'exécution de cette opération.
- Le coût dépend de l'opération effectuée

Coût des opérations

- On appelle coût d'une opération le temps d'exécution de cette opération.
- Le coût dépend de l'opération effectuée
- D'où la nécessité d'avoir une constante pour chaque type d'opération, pour un calcul exact : C_+, C_-, C_*, $C_{\leftarrow}$, $C_<$...

Coût des opérations

- On appelle coût d'une opération le temps d'exécution de cette opération.
- Le coût dépend de l'opération effectuée
- D'où la nécessité d'avoir une constante pour chaque type d'opération, pour un calcul exact : C_+, C_-, C_*, $C_{\leftarrow}$, $C_<$...
- Le coût peut aussi dépendre de la taille des données, si celles-ci peuvent devenir grandes.

Coût des opérations

- On appelle coût d'une opération le temps d'exécution de cette opération.
- Le coût dépend de l'opération effectuée
- D'où la nécessité d'avoir une constante pour chaque type d'opération, pour un calcul exact : C_+, C_-, C_*, $C_{\leftarrow}$, $C_<$...
- Le coût peut aussi dépendre de la taille des données, si celles-ci peuvent devenir grandes.

Définition 2.1 (Modèle à coût fixe)

Cela consiste à considérer que la durée des opérations ne dépend pas de la taille des objets.

Définition 2.2 (Complexité en temps)

La complexité en temps d'un algorithme est une fonction C dépendant de la taille n des données (éventuellement de plusieurs variables s'il y a en entrée des objets pouvant être de tailles différentes) et estimant le temps de réponse en fonction de n et des constantes $C_\bullet$ définies ci-dessus.

Algorithme 1 : Évaluation naïve d'un polynôme

Entrée : T : tableau, coefficients d'un polynôme P, a : réel
Sortie : $P(a)$: réel.

$S \leftarrow 0$;
pour $i \leftarrow 0$ **à** n **faire**
    $b \leftarrow 1$;
    pour $j \leftarrow 1$ **à** i **faire**
        $b \leftarrow b \times a$
    fin pour
    $S \leftarrow S + T[i] \times b$
fin pour
renvoyer S

Algorithme 1 : Évaluation naïve d'un polynôme

Entrée : T : tableau, coefficients d'un polynôme P, a : réel
Sortie : $P(a)$: réel.

$S \leftarrow 0$;
pour $i \leftarrow 0$ **à** n **faire**
  $b \leftarrow 1$;
  pour $j \leftarrow 1$ **à** i **faire**
    $b \leftarrow b \times a$
  fin pour
  $S \leftarrow S + T[i] \times b$
fin pour
renvoyer S

Exercice 1

1. Justifier la correction de l'algorithme ci-dessus.

Algorithme 1 : Évaluation naïve d'un polynôme

Entrée : T : tableau, coefficients d'un polynôme P, a : réel
Sortie : $P(a)$: réel.

$S \leftarrow 0$;
pour $i \leftarrow 0$ **à** n **faire**
    $b \leftarrow 1$;
    pour $j \leftarrow 1$ **à** i **faire**
        $b \leftarrow b \times a$
    fin pour
    $S \leftarrow S + T[i] \times b$
fin pour
renvoyer S

Exercice 1

1. Justifier la correction de l'algorithme ci-dessus.
2. $C(n) = n^2\left(\frac{C_+}{2} + \frac{C_*}{2} + C_{\leftarrow}\right) + n\left(\frac{3}{2}C_+ + \frac{C_*}{2} + 2C_{\leftarrow}\right) + (4C_{\leftarrow} + 2C_+ + C_*)$.

- Dans la plupart des cas, une expression si précise est inutile :

- Dans la plupart des cas, une expression si précise est inutile :
- on en retient : $C(n) = \Theta(n)$.

- Dans la plupart des cas, une expression si précise est inutile :
- on en retient : $C(n) = \Theta(n)$.
- On dit que l'algorithme un coût, ou une complexité, quadratique.

- Dans la plupart des cas, une expression si précise est inutile :
- on en retient : $C(n) = \Theta(n)$.
- On dit que l'algorithme un coût, ou une complexité, quadratique.
- Ainsi, multiplier les données par 10 multiplie le temps de réponse par 100.

- Dans la plupart des cas, une expression si précise est inutile :
- on en retient : $C(n) = \Theta(n)$.
- On dit que l'algorithme un coût, ou une complexité, quadratique.
- Ainsi, multiplier les données par 10 multiplie le temps de réponse par 100.
- Essai informatique.

Définition 2.3 (Complexités de référence)

- Coût constant : $C(n) = \Theta(1)$

Définition 2.3 (Complexités de référence)

- Coût constant : $C(n) = \Theta(1)$
- Coût logarithmique : $C(n) = \Theta(\ln(n))$

Définition 2.3 (Complexités de référence)

- Coût constant : $C(n) = \Theta(1)$
- Coût logarithmique : $C(n) = \Theta(\ln(n))$
- Coût linéaire : $C(n) = \Theta(n)$

Définition 2.3 (Complexités de référence)

- Coût constant : $C(n) = \Theta(1)$
- Coût logarithmique : $C(n) = \Theta(\ln(n))$
- Coût linéaire : $C(n) = \Theta(n)$
- Coût quasi-linéaire : $C(n) = \Theta(n \ln(n))$

Définition 2.3 (Complexités de référence)

- Coût constant : $C(n) = \Theta(1)$
- Coût logarithmique : $C(n) = \Theta(\ln(n))$
- Coût linéaire : $C(n) = \Theta(n)$
- Coût quasi-linéaire : $C(n) = \Theta(n \ln(n))$
- Coût quadratique : $C(n) = \Theta(n^2)$

Définition 2.3 (Complexités de référence)

- Coût constant : $C(n) = \Theta(1)$
- Coût logarithmique : $C(n) = \Theta(\ln(n))$
- Coût linéaire : $C(n) = \Theta(n)$
- Coût quasi-linéaire : $C(n) = \Theta(n \ln(n))$
- Coût quadratique : $C(n) = \Theta(n^2)$
- Coût cubique : $C(n) = \Theta(n^3)$

Définition 2.3 (Complexités de référence)

- Coût constant : $C(n) = \Theta(1)$
- Coût logarithmique : $C(n) = \Theta(\ln(n))$
- Coût linéaire : $C(n) = \Theta(n)$
- Coût quasi-linéaire : $C(n) = \Theta(n \ln(n))$
- Coût quadratique : $C(n) = \Theta(n^2)$
- Coût cubique : $C(n) = \Theta(n^3)$
- Coût polynomial : $C(n) = \Theta(n^\alpha)$, $\alpha > 0$

Définition 2.3 (Complexités de référence)

- Coût constant : $C(n) = \Theta(1)$
- Coût logarithmique : $C(n) = \Theta(\ln(n))$
- Coût linéaire : $C(n) = \Theta(n)$
- Coût quasi-linéaire : $C(n) = \Theta(n \ln(n))$
- Coût quadratique : $C(n) = \Theta(n^2)$
- Coût cubique : $C(n) = \Theta(n^3)$
- Coût polynomial : $C(n) = \Theta(n^\alpha)$, $\alpha > 0$
- Coût exponentiel : $C(n) = \Theta(x^n)$, $x > 1$.

Temps approximatif de calcul

Hypothèses : Donnée de taille $n = 10^6$, 1 milliard d'opérations par seconde, le terme dans le O donne le nombre d'opérations.

Temps approximatif de calcul

Hypothèses : Donnée de taille $n = 10^6$, 1 milliard d'opérations par seconde, le terme dans le O donne le nombre d'opérations.

- $O(1)$: 1 ns

Temps approximatif de calcul

Hypothèses : Donnée de taille $n = 10^6$, 1 milliard d'opérations par seconde, le terme dans le O donne le nombre d'opérations.

- $O(1)$: 1 ns
- $O(\ln(n))$: 15 ns

Temps approximatif de calcul

Hypothèses : Donnée de taille $n = 10^6$, 1 milliard d'opérations par seconde, le terme dans le O donne le nombre d'opérations.

- $O(1)$: 1 ns
- $O(\ln(n))$: 15 ns
- $O(n)$: 1 ms

Temps approximatif de calcul

Hypothèses : Donnée de taille $n = 10^6$, 1 milliard d'opérations par seconde, le terme dans le O donne le nombre d'opérations.

- $O(1)$: 1 ns
- $O(\ln(n))$: 15 ns
- $O(n)$: 1 ms
- $O(n \ln n)$: 15 ms

Temps approximatif de calcul

Hypothèses : Donnée de taille $n = 10^6$, 1 milliard d'opérations par seconde, le terme dans le O donne le nombre d'opérations.

- $O(1)$: 1 ns
- $O(\ln(n))$: 15 ns
- $O(n)$: 1 ms
- $O(n \ln n)$: 15 ms
- $O(n^2)$: 15 min

Temps approximatif de calcul

Hypothèses : Donnée de taille $n = 10^6$, 1 milliard d'opérations par seconde, le terme dans le O donne le nombre d'opérations.

- $O(1)$: 1 ns
- $O(\ln(n))$: 15 ns
- $O(n)$: 1 ms
- $O(n \ln n)$: 15 ms
- $O(n^2)$: 15 min
- $O(n^3)$: 30 ans

Temps approximatif de calcul

Hypothèses : Donnée de taille $n = 10^6$, 1 milliard d'opérations par seconde, le terme dans le O donne le nombre d'opérations.

- $O(1)$: 1 ns
- $O(\ln(n))$: 15 ns
- $O(n)$: 1 ms
- $O(n \ln n)$: 15 ms
- $O(n^2)$: 15 min
- $O(n^3)$: 30 ans
- $O(2^n)$: 10^{300000} milliards d'années !

Temps approximatif de calcul

Hypothèses : Donnée de taille $n = 10^6$, 1 milliard d'opérations par seconde, le terme dans le O donne le nombre d'opérations.

- $O(1)$: 1 ns
- $O(\ln(n))$: 15 ns
- $O(n)$: 1 ms
- $O(n \ln n)$: 15 ms
- $O(n^2)$: 15 min
- $O(n^3)$: 30 ans
- $O(2^n)$: 10^{300000} milliards d'années !

L'amélioration des complexités est un enjeu important !

Remarque 2.4 (Doublement de la taille des données)

- $\Theta(1)$: constant

Remarque 2.4 (Doublement de la taille des données)

- $\Theta(1)$: constant
- $\Theta(\ln(n))$: $+C$

Remarque 2.4 (Doublement de la taille des données)

- $\Theta(1)$: constant
- $\Theta(\ln(n))$: $+C$
- $\Theta(n)$: $\times 2$

Remarque 2.4 (Doublement de la taille des données)

- $\Theta(1)$: constant
- $\Theta(\ln(n))$: $+C$
- $\Theta(n)$: $\times 2$
- $\Theta(n^2)$: $\times 4$

Remarque 2.4 (Doublement de la taille des données)

- $\Theta(1)$: constant
- $\Theta(\ln(n))$: $+C$
- $\Theta(n)$: $\times 2$
- $\Theta(n^2)$: $\times 4$
- $\Theta(n^3)$: $\times 8$

Remarque 2.4 (Doublement de la taille des données)

- $\Theta(1)$: constant
- $\Theta(\ln(n))$: $+C$
- $\Theta(n)$: $\times 2$
- $\Theta(n^2)$: $\times 4$
- $\Theta(n^3)$: $\times 8$
- ça ne répond plus pour un algorithme exponentiel !

II-2. Simplification du calcul de la complexité

- Dans l'exemple, on ne s'est pas servi des valeurs des $C_\bullet$, seulement de leur positivité pour conclure.

II-2. Simplification du calcul de la complexité

- Dans l'exemple, on ne s'est pas servi des valeurs des $C_\bullet$, seulement de leur positivité pour conclure.
- On montre ici que la valeur des constantes ne modifie pas le comportement asymptotique en Θ, à partir du moment où elles sont > 0.

Proposition 2.5 (Effet de la modification d'une des constantes)

Soit C_1 la complexité d'un algorithme. Soit C_2 la complexité obtenue en remplaçant une constante $C_\bullet > 0$ par une constante $C'_\bullet > 0$. On a alors $C_2 = \Theta(C_1)$.

Proposition 2.5 (Effet de la modification d'une des constantes)

Soit C_1 la complexité d'un algorithme. Soit C_2 la complexité obtenue en remplaçant une constante $C_\bullet > 0$ par une constante $C'_\bullet > 0$. On a alors $C_2 = \Theta(C_1)$.

Corollaire 2.6

Modifier strictement positivement les constantes n'a pas d'effet sur le comportement asymptotique de la complexité.

Proposition 2.5 (Effet de la modification d'une des constantes)

Soit C_1 la complexité d'un algorithme. Soit C_2 la complexité obtenue en remplaçant une constante $C_\bullet > 0$ par une constante $C'_\bullet > 0$. On a alors $C_2 = \Theta(C_1)$.

Corollaire 2.6

Modifier strictement positivement les constantes n'a pas d'effet sur le comportement asymptotique de la complexité.

Corollaire 2.7 (Calcul simplifié de la complexité asymptotique)

On considère $C_\bullet = 1$ pour toute opération.

Proposition 2.5 (Effet de la modification d'une des constantes)

Soit C_1 la complexité d'un algorithme. Soit C_2 la complexité obtenue en remplaçant une constante $C_\bullet > 0$ par une constante $C'_\bullet > 0$. On a alors $C_2 = \Theta(C_1)$.

Corollaire 2.6

Modifier strictement positivement les constantes n'a pas d'effet sur le comportement asymptotique de la complexité.

Corollaire 2.7 (Calcul simplifié de la complexité asymptotique)

On considère $C_\bullet = 1$ pour toute opération.

Exemple 2.8

Reprendre le calcul de la complexité asymptotique de l'évaluation naïve d'un polynôme sous cet angle.

II-3. Amélioration de la complexité de l'exemple donné

- Un même problème peut souvent être résolus de plusieurs façons.

II-3. Amélioration de la complexité de l'exemple donné

- Un même problème peut souvent être résolus de plusieurs façons.
- Mais les différents algorithmes ne sont pas nécessairement équivalents du point de vue de la complexité.

II-3. Amélioration de la complexité de l'exemple donné

- Un même problème peut souvent être résolus de plusieurs façons.
- Mais les différents algorithmes ne sont pas nécessairement équivalents du point de vue de la complexité.
- Exemple : amélioration de l'évaluation polynomiale par un calcul plus efficace des puissances :

II-3. Amélioration de la complexité de l'exemple donné

- Un même problème peut souvent être résolus de plusieurs façons.
- Mais les différents algorithmes ne sont pas nécessairement équivalents du point de vue de la complexité.
- Exemple : amélioration de l'évaluation polynomiale par un calcul plus efficace des puissances :
- $a^{13} = a \times (a^6)^2 = a \times ((a^3)^2)^2 = a \times ((a \times a^2)^2)^2$.

II-3. Amélioration de la complexité de l'exemple donné

- Un même problème peut souvent être résolus de plusieurs façons.
- Mais les différents algorithmes ne sont pas nécessairement équivalents du point de vue de la complexité.
- Exemple : amélioration de l'évaluation polynomiale par un calcul plus efficace des puissances :
- $a^{13} = a \times (a^6)^2 = a \times ((a^3)^2)^2 = a \times ((a \times a^2)^2)^2$.
- On passe de 12 opérations à 5 opérations.

Fonction 2 : exp-rapide(a,n)

Entrée : a entier ou réel, n entier
Sortie : a^n
si $n = 0$ **alors**
  renvoyer *1* et sortir
fin si
$y \leftarrow a$;
$z \leftarrow 1$;
tant que $n > 0$ **faire**
  si n *est impair* **alors**
    $n \leftarrow n - 1$;
    $z \leftarrow z * y$
  sinon
    $n \leftarrow \frac{n}{2}$;
    $y \leftarrow y * y$
  fin si
fin tant que
renvoyer z

Algorithme 3 : Évaluation polynomiale par exponentiation rapide

Entrée : T : tableau des coefficients d'un polynôme P, a réel
Sortie : $P(a)$
$S \leftarrow 0$;
pour $i \leftarrow 0$ **à** n **faire**
| $S \leftarrow S + T[i] \times$ exp-rapide(a, i)
fin pour

Algorithme 3 : Évaluation polynomiale par exponentiation rapide

Entrée : T : tableau des coefficients d'un polynôme P, a réel
Sortie : $P(a)$
$S \leftarrow 0$;
pour $i \leftarrow 0$ **à** n **faire**
  $S \leftarrow S + T[i] \times$ exp-rapide(a, i)
fin pour

Exercice 2

1. Terminaison et correction de l'algorithme d'exponentiation rapide.

Algorithme 3 : Évaluation polynomiale par exponentiation rapide

Entrée : T : tableau des coefficients d'un polynôme P, a réel
Sortie : $P(a)$
$S \leftarrow 0$;
pour $i \leftarrow 0$ **à** n **faire**
  $S \leftarrow S + T[i] \times$ exp-rapide(a, i)
fin pour

Exercice 2

1. Terminaison et correction de l'algorithme d'exponentiation rapide.
2. Coût de l'exponentiation : $C(2^k) \leqslant C(n) < C(2^{k+1}) + 2k$

Algorithme 3 : Évaluation polynomiale par exponentiation rapide

Entrée : T : tableau des coefficients d'un polynôme P, a réel
Sortie : $P(a)$
$S \leftarrow 0$;
pour $i \leftarrow 0$ **à** n **faire**
  $S \leftarrow S + T[i] \times$ exp-rapide(a, i)
fin pour

Exercice 2

1. Terminaison et correction de l'algorithme d'exponentiation rapide.
2. Coût de l'exponentiation : $C(2^k) \leqslant C(n) < C(2^{k+1}) + 2k$
3. En déduire $C(n) = \Omega(\ln(n))$.

Algorithme 3 : Évaluation polynomiale par exponentiation rapide

Entrée : T : tableau des coefficients d'un polynôme P, a réel
Sortie : $P(a)$
$S \leftarrow 0$;
pour $i \leftarrow 0$ **à** n **faire**
  $S \leftarrow S + T[i] \times$ exp-rapide(a, i)
fin pour

Exercice 2

1. Terminaison et correction de l'algorithme d'exponentiation rapide.
2. Coût de l'exponentiation : $C(2^k) \leqslant C(n) < C(2^{k+1}) + 2k$
3. En déduire $C(n) = \Omega(\ln(n))$.
4. Coût de l'évaluation polynomiale : $\Omega(n \ln(n))$.

On peut éviter de recommencer le calcul des puissances depuis le début, en factorisant au fur et à mesure :

On peut éviter de recommencer le calcul des puissances depuis le début, en factorisant au fur et à mesure :

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x^n$$
$$= a_0 + x(a_1 + x(a_n + \cdots + x(a_{n-1} + x a_n))).$$

Algorithme 4 : Hörner

Entrée : T, coefficients d'un polynôme P, a : réel
Sortie : $P(a)$
$S \leftarrow T[n]$;
pour $i \leftarrow n-1$ **descendant à** 0 **faire**
| $S \leftarrow S * a + T[i]$
fin pour
renvoyer (S)

Algorithme 4 : Hörner

Entrée : T, coefficients d'un polynôme P, a : réel
Sortie : $P(a)$
$S \leftarrow T[n]$;
pour $i \leftarrow n-1$ **descendant à** 0 **faire**
  $S \leftarrow S * a + T[i]$
fin pour
renvoyer (S)

Exercice 3

1. Montrer la correction de l'algorithme de Hörner.

Algorithme 4 : Hörner

Entrée : T, coefficients d'un polynôme P, a : réel
Sortie : $P(a)$
$S \leftarrow T[n]$;
pour $i \leftarrow n-1$ **descendant à** 0 **faire**
| $S \leftarrow S * a + T[i]$
fin pour
renvoyer (S)

Exercice 3

1. Montrer la correction de l'algorithme de Hörner.
2. Montrer que l'algorithme de Hörner a un coût linéaire ($\Theta(n)$).

III. Complexité dans le meilleur ou le pire des cas, en moyenne

III-1. Complexité dans le meilleur et le pire des cas

La complexité dépend parfois de l'entrée de façon plus complexe que seulement par sa taille.

III. Complexité dans le meilleur ou le pire des cas, en moyenne

III-1. Complexité dans le meilleur et le pire des cas

La complexité dépend parfois de l'entrée de façon plus complexe que seulement par sa taille.

Algorithme 5 : Test de primalité

Entrée : p : entier > 1
Sortie : b booléen, résultat du test de primalité
$i \leftarrow 2$;
$m = \sqrt{p}$;
tant que $i \leqslant m$ **faire**
  si $p \mod i = 0$ **alors**
    renvoyer *False* et sortir
  fin si
  $i \leftarrow i + 1$
fin tant que
renvoyer *True*

Exercice 4

1. Justifier la terminaison et la correction de cet algorithme

Exercice 4

1. Justifier la terminaison et la correction de cet algorithme
2. Montrer que si p est pair (ce qui peut arriver pour p grand !) le temps de réponse est en $\Theta(1)$ (en négligeant le coût du calcul de $\sqrt{p}$)

Exercice 4

1. Justifier la terminaison et la correction de cet algorithme
2. Montrer que si p est pair (ce qui peut arriver pour p grand !) le temps de réponse est en $\Theta(1)$ (en négligeant le coût du calcul de $\sqrt{p}$)
3. Montrer que si p est premier (Euclide m'a affirmé un jour qu'on peut aussi trouver des p aussi grand qu'on veut vérifiant cela), le temps de réponse est en $\Theta(\sqrt{p})$ (dans le modèle à coût constant, qui devient discutable ici pour des grandes valeurs de p).

Définition 3.1 (Complexité dans le pire et le meilleur des cas)

- Complexité dans le meilleur des cas : nombre minimal $C_{\_}(n)$ d'opérations à effectuer pour des objets de taille n.

Définition 3.1 (Complexité dans le pire et le meilleur des cas)

- Complexité dans le meilleur des cas : nombre minimal $C_-(n)$ d'opérations à effectuer pour des objets de taille n.
- Complexité dans le pire des cas : nombre maximal $C_+(n)$ d'opérations à effectuer pour des objets de taille n.

Définition 3.1 (Complexité dans le pire et le meilleur des cas)

- Complexité dans le meilleur des cas : nombre minimal $C_-(n)$ d'opérations à effectuer pour des objets de taille n.
- Complexité dans le pire des cas : nombre maximal $C_+(n)$ d'opérations à effectuer pour des objets de taille n.

Pour T de taille n : $C_-(n) \leqslant C(T) \leqslant C_+(n)$

Définition 3.1 (Complexité dans le pire et le meilleur des cas)

- Complexité dans le meilleur des cas : nombre minimal $C_-(n)$ d'opérations à effectuer pour des objets de taille n.
- Complexité dans le pire des cas : nombre maximal $C_+(n)$ d'opérations à effectuer pour des objets de taille n.

Pour T de taille n : $C_-(n) \leqslant C(T) \leqslant C_+(n)$

Exemple 3.2

En considérant que la taille des entiers est donnée par le nombre de leur chiffres, pour le test de primalité, on obtient :

$$C_-(n) = \Theta(1) \qquad \text{et} \qquad C_+(n) = \Theta(10^{\frac{n}{2}}).$$

Définition 3.1 (Complexité dans le pire et le meilleur des cas)

- Complexité dans le meilleur des cas : nombre minimal $C_-(n)$ d'opérations à effectuer pour des objets de taille n.
- Complexité dans le pire des cas : nombre maximal $C_+(n)$ d'opérations à effectuer pour des objets de taille n.

Pour T de taille n : $C_-(n) \leqslant C(T) \leqslant C_+(n)$

Exemple 3.2

En considérant que la taille des entiers est donnée par le nombre de leur chiffres, pour le test de primalité, on obtient :

$$C_-(n) = \Theta(1) \qquad \text{et} \qquad C_+(n) = \Theta(10^{\frac{n}{2}}).$$

Ainsi, $C(n) = \Omega(1)$ et $C(n) = O(10^{\frac{n}{2}})$.

Exercice 5

On propose une amélioration au tri à bulle vu dans un exercice du chapitre précédent, consistant à compter le nombre d'échanges effectués lors d'un passage du début à la fin du tableau, et à s'arrêter lorsqu'aucun échange n'est effectué (ce qui signifie que les termes sont dans l'ordre). Déterminer la complexité dans le meilleur et dans le pire des cas, en donnant à chaque fois un exemple de configuration initiale associée.

Remarque 3.3

De nombreux tris (mais pas tous) sont plus rapides sur des tableaux presque triés (obtenus par exemple en ajoutant un petit nombre d'éléments à un tableau initialement trié, c'est une situation très fréquente, correspondant par exemple à l'ajout de quelques éléments dans une base qu'on maintient triée). Certains algorithmes, réputés plus lents sur des tableaux généraux, peuvent être plus rapides dans cette situation que des algorithmes récursifs réputés plus rapides dans des situations générales. Ainsi, avant de se précipiter vers un algorithme réputé rapide, il est important d'analyser le contexte dans lequel l'algorithme doit être utilisé.

III-2. Complexité en moyenne

Définition 3.4 (Complexité en moyenne)

Complexité en moyenne $C_m(n)$: espérance de la variable $C(n)$, nombre d'opérations, défini sur l'espace probabilisé des objets de taille n.

III-2. Complexité en moyenne

Définition 3.4 (Complexité en moyenne)

Complexité en moyenne $C_m(n)$: espérance de la variable $C(n)$, nombre d'opérations, défini sur l'espace probabilisé des objets de taille n.

Très souvent, on considère la mesure de probabilité uniforme sur l'ensemble des objets.

III-2. Complexité en moyenne

Définition 3.4 (Complexité en moyenne)

Complexité en moyenne $C_m(n)$: espérance de la variable $C(n)$, nombre d'opérations, défini sur l'espace probabilisé des objets de taille n.

Très souvent, on considère la mesure de probabilité uniforme sur l'ensemble des objets.

Exercice 6

On considère un alphabet $\mathcal{A}$ à k lettres. On effectue, sur des tableaux constitués de caractères de l'alphabet $\mathcal{A}$, la recherche du premier a. Déterminer la complexité moyenne de cet algorithme, sachant que les tableaux considérés sont remplis aléatoirement.

III-3. Algorithmes randomisés

Souvent, pires cas = cas fréquents dans la vie courante.

III-3. Algorithmes randomisés

Souvent, pires cas = cas fréquents dans la vie courante.

On introduit alors un aléa pour que le risque d'être dans un tel cas soit moindre.

III-3. Algorithmes randomisés

Souvent, pires cas = cas fréquents dans la vie courante.
On introduit alors un aléa pour que le risque d'être dans un tel cas soit moindre.

Définition 3.5 (Algorithme randomisé)

On parle d'algorithme randomisé lorsqu'on a introduit dans l'algorithme un aléa dont le but est d'éviter les cas extrêmes.

Fonction 6 : trirapidenonrandomisé(T)

Entrée : T : tableau à trier
Sortie : T : tableau trié
Comparer tous les éléments autres que $T[0]$ à $T[0]$ et séparer en 2 tableaux ;
U : tableau des plus petits ;
V : tableau des plus grands ;
$U \leftarrow$ trirapidenonrandomisé(U) ;
$V \leftarrow$ trirapidenonrandomisé(V) ;
renvoyer concaténation de U, $[T[0]]$ et V.

Fonction 6 : trirapidenonrandomisé(T)

Entrée : T : tableau à trier
Sortie : T : tableau trié
Comparer tous les éléments autres que $T[0]$ à $T[0]$ et séparer en 2 tableaux ;
U : tableau des plus petits ;
V : tableau des plus grands ;
$U \leftarrow$ trirapidenonrandomisé(U) ;
$V \leftarrow$ trirapidenonrandomisé(V) ;
renvoyer concaténation de U, $[T[0]]$ et V.

- Meilleur des cas = moyenne = $O(n\ln(n))$ (admis)

Fonction 6 : trirapidenonrandomisé(T)

Entrée : T : tableau à trier
Sortie : T : tableau trié
Comparer tous les éléments autres que $T[0]$ à $T[0]$ et séparer en 2 tableaux ;
U : tableau des plus petits ;
V : tableau des plus grands ;
$U \leftarrow$ trirapidenonrandomisé(U) ;
$V \leftarrow$ trirapidenonrandomisé(V) ;
renvoyer concaténation de U, $[T[0]]$ et V.

- Meilleur des cas = moyenne = $O(n \ln(n))$ (admis)

Exercice 7

Montrer que le temps de calcul du tri rapide sur un tableau trié (dans un sens ou dans l'autre) est en $O(n^2)$.

Fonction 7 : trirapiderandomisé(T)

Entrée : T : tableau à trier
Sortie : T : tableau trié
Choisir i aléatoirement dans $[\![0, n-1]\!]$ (indices du tableau) ;
Comparer tous les éléments autres que $T[i]$ à $T[i]$ et séparer en 2 tableaux ;
U : tableau des plus petits ;
V : tableau des plus grands ;
$U \leftarrow$ trirapiderandomisé(U) ;
$V \leftarrow$ trirapiderandomisé(V) ;
renvoyer concaténation de U, $[T[i]]$ et V.

Fonction 7 : trirapiderandomisé(T)

Entrée : T : tableau à trier
Sortie : T : tableau trié
Choisir i aléatoirement dans $[\![0, n-1]\!]$ (indices du tableau) ;
Comparer tous les éléments autres que $T[i]$ à $T[i]$ et séparer en 2 tableaux ;
U : tableau des plus petits ;
V : tableau des plus grands ;
$U \leftarrow$ trirapiderandomisé(U) ;
$V \leftarrow$ trirapiderandomisé(V) ;
renvoyer concaténation de U, $[T[i]]$ et V.

De cette façon, on subit moins les effets de bord.

IV. Limitations du modèle à coûts fixes

- Le modèle à coûts fixes peut être discutable si on est amené à faire des opérations élémentaires sur des grands objets.

IV. Limitations du modèle à coûts fixes

- Le modèle à coûts fixes peut être discutable si on est amené à faire des opérations élémentaires sur des grands objets.
- Exemple : comparer Hörner non modulaire et Hörner modulaire.

IV. Limitations du modèle à coûts fixes

- Le modèle à coûts fixes peut être discutable si on est amené à faire des opérations élémentaires sur des grands objets.
- Exemple : comparer Hörner non modulaire et Hörner modulaire.
- En particulier, pour le calcul modulaire, il est important de faire la réduction modulaire à chaque étape du calcul pour ne pas travailler sur des nombres trop gros.

IV. Limitations du modèle à coûts fixes

- Le modèle à coûts fixes peut être discutable si on est amené à faire des opérations élémentaires sur des grands objets.
- Exemple : comparer Hörner non modulaire et Hörner modulaire.
- En particulier, pour le calcul modulaire, il est important de faire la réduction modulaire à chaque étape du calcul pour ne pas travailler sur des nombres trop gros.
- Il est donc nécessaire de tenir compte de la taille des entiers, et de la complexité des opérations élémentaires :

IV. Limitations du modèle à coûts fixes

- Le modèle à coûts fixes peut être discutable si on est amené à faire des opérations élémentaires sur des grands objets.
- Exemple : comparer Hörner non modulaire et Hörner modulaire.
- En particulier, pour le calcul modulaire, il est important de faire la réduction modulaire à chaque étape du calcul pour ne pas travailler sur des nombres trop gros.
- Il est donc nécessaire de tenir compte de la taille des entiers, et de la complexité des opérations élémentaires :
- $m + n$: complexité $\max(\lg(m), \lg(n))$,

IV. Limitations du modèle à coûts fixes

- Le modèle à coûts fixes peut être discutable si on est amené à faire des opérations élémentaires sur des grands objets.
- Exemple : comparer Hörner non modulaire et Hörner modulaire.
- En particulier, pour le calcul modulaire, il est important de faire la réduction modulaire à chaque étape du calcul pour ne pas travailler sur des nombres trop gros.
- Il est donc nécessaire de tenir compte de la taille des entiers, et de la complexité des opérations élémentaires :
- $m + n$: complexité $\max(\lg(m), \lg(n))$,
- mn : complexité $\lg(m) \times \lg(n)$.

Algorithme 8 : Hörner 2

Entrée : T, coefficients d'un polynôme P, a : réel, K base du modulo

Sortie : $P(a)$

$S \leftarrow T[n]$;

pour $i \leqslant n-1$ **descendant à** 0 **faire**

| $S \leftarrow S * a + T[i]$

fin pour

renvoyer $(S \mod K)$

Algorithme 8 : Hörner 2

Entrée : T, coefficients d'un polynôme P, a : réel, K base du modulo
Sortie : $P(a)$
$S \leftarrow T[n]$;
pour $i \leqslant n-1$ **descendant à** 0 **faire**
  $S \leftarrow S * a + T[i]$
fin pour
renvoyer $(S \mod K)$

Algorithme 9 : Hörner 3

Entrée : T, coefficients d'un polynôme P, a : réel, K base du modulo
Sortie : $P(a)$
$S \leftarrow T[n] \mod K$;
pour $i \leftarrow n-1$ **descendant à** 0 **faire**
  $S \leftarrow ((S * a + T[i]) \mod K$
fin pour
renvoyer (S)

V. Complexité en mémoire, ou en espace

- Autre problème limitant : l'encombrement en mémoire :

V. Complexité en mémoire, ou en espace

- Autre problème limitant : l'encombrement en mémoire :
- ralentissements, saturations.

V. Complexité en mémoire, ou en espace

- Autre problème limitant : l'encombrement en mémoire :
- ralentissements, saturations.
- l'étude de la complexité en mémoire vise à comprendre la place mémoire nécessaire pour que l'algorithme tourne.

V. Complexité en mémoire, ou en espace

- Autre problème limitant : l'encombrement en mémoire :
- ralentissements, saturations.
- l'étude de la complexité en mémoire vise à comprendre la place mémoire nécessaire pour que l'algorithme tourne.
- On abordera très peu ce point de vue.

VI. Étude de quelques algorithmes de recherche

But :

VI. Étude de quelques algorithmes de recherche

But :

- dégager certains algorithmes intervenant dans de multiples contextes.

VI. Étude de quelques algorithmes de recherche

But :

- dégager certains algorithmes intervenant dans de multiples contextes.
- A voir comme des briques élémentaires.

VI. Étude de quelques algorithmes de recherche

But :

- dégager certains algorithmes intervenant dans de multiples contextes.
- A voir comme des briques élémentaires.
- D'ailleurs souvent déjà implémenté.

VI. Étude de quelques algorithmes de recherche

But :

- dégager certains algorithmes intervenant dans de multiples contextes.
- A voir comme des briques élémentaires.
- D'ailleurs souvent déjà implémenté.
- C'est d'autant plus important d'avoir une idée de leur complexité.

VI-1. Recherche du maximum d'une liste

VI-1. Recherche du maximum d'une liste

Problème : Étant donné une liste de réels, trouver le maximum de cette liste (ou de façon similaire, son minimum).

VI-1. Recherche du maximum d'une liste

Problème : Étant donné une liste de réels, trouver le maximum de cette liste (ou de façon similaire, son minimum).

Idée de résolution : Progresser dans le tableau en mémorisant le plus grand élément rencontré jusque-là. On peut aussi mémoriser sa place.

Algorithme 10 : Recherche maximum

Entrée : T, tableau de réels de taille n
Sortie : $\max(T)$
$M \leftarrow T[0]$;
pour $i \leftarrow 1$ **à** $n-1$ **faire**
    si $T[i] > M$ **alors**
        $M \leftarrow T[i]$
    fin si
fin pour
renvoyer (M)

Algorithme 10 : Recherche maximum

Entrée : T, tableau de réels de taille n
Sortie : $\max(T)$
$M \leftarrow T[0]$;
pour $i \leftarrow 1$ **à** $n-1$ **faire**
  si $T[i] > M$ **alors**
    $M \leftarrow T[i]$
  fin si
fin pour
renvoyer (M)

Exercice 8

Montrer que l'algorithme `Recherche maximum` est correct, et que son coût est en $\Theta(n)$.

VI-2. Recherche d'un élément dans une liste

VI-2. Recherche d'un élément dans une liste

Problème : Étant donné une liste, trouver un terme particulier dans cette liste, s'il y est.

VI-2. Recherche d'un élément dans une liste

Problème : Étant donné une liste, trouver un terme particulier dans cette liste, s'il y est.

Idée de résolution : Parcourir la liste jusqu'à ce qu'on trouve l'élément souhaité.

VI-2. Recherche d'un élément dans une liste

Problème : Étant donné une liste, trouver un terme particulier dans cette liste, s'il y est.

Idée de résolution : Parcourir la liste jusqu'à ce qu'on trouve l'élément souhaité.
Variantes : si on veut toutes les occurrences, on parcourt la liste en entier ; si on veut la dernière occurrence, on parcourt le tableau à partir de la fin.

Algorithme 11 : Recherche première occurrence

Entrée : T, tableau de réels de taille n ; a élément à chercher dans T

Sortie : i : indice de la première occurrence de T, ou `None` si pas d'occurrence

$i \leftarrow 0$;
tant que $i < n$ *et* $T[i] \neq a$ **faire**
 | $i \leftarrow i + 1$
fin tant que
si $i < n$ **alors**
 | **renvoyer** (i)
fin si

Algorithme 11 : Recherche première occurrence

Entrée : T, tableau de réels de taille n ; a élément à chercher dans T
Sortie : i : indice de la première occurrence de T, ou `None` si pas d'occurrence

$i \leftarrow 0$;
tant que $i < n$ *et* $T[i] \neq a$ **faire**
  $i \leftarrow i + 1$
fin tant que
si $i < n$ **alors**
  renvoyer (i)
fin si

Tests booléens dans le mode paresseux.

Exercice 9

1. Montrer la terminaison et la correction de l'algorithme `Recherche première occurrence`

Exercice 9

1. Montrer la terminaison et la correction de l'algorithme `Recherche première occurrence`
2. Montrer que sa complexité dans le pire des cas est en $\Theta(n)$, et dans le meilleur des cas en $\Theta(1)$.

Exercice 9

1. Montrer la terminaison et la correction de l'algorithme `Recherche première occurrence`
2. Montrer que sa complexité dans le pire des cas est en $\Theta(n)$, et dans le meilleur des cas en $\Theta(1)$.
3. Montrer que si T est un tableau constitué de N objets supposés équiprobables, et que a est un de ces objets, la complexité en moyenne tend vers N lorsque n tend vers ∞. Ainsi, la complexité moyenne est en $\Theta(1)$, si on considère N comme une constante. Si on est amené à faire varier à la fois la taille n du tableau et le nombre N d'objets différents, on peut écrire $C_m(n, N) = \Theta(N)$.

Exercice 9

1. Montrer la terminaison et la correction de l'algorithme `Recherche première occurrence`
2. Montrer que sa complexité dans le pire des cas est en $\Theta(n)$, et dans le meilleur des cas en $\Theta(1)$.
3. Montrer que si T est un tableau constitué de N objets supposés équiprobables, et que a est un de ces objets, la complexité en moyenne tend vers N lorsque n tend vers ∞. Ainsi, la complexité moyenne est en $\Theta(1)$, si on considère N comme une constante. Si on est amené à faire varier à la fois la taille n du tableau et le nombre N d'objets différents, on peut écrire $C_m(n, N) = \Theta(N)$.
4. Comment modifier l'algorithme ci-dessus pour qu'il renvoie la dernière occurrence ? Toutes les occurrences. Quelle est la complexité dans ce cas ?

Remarque 6.1

Si le tableau T est utilisé essentiellement pour des tests d'appartenance, il faut se demander si la structure de tableau est vraiment adaptée : il est peut-être plus intéressant de considérer une structure d'ensemble, le test d'appartenance s'y faisant plus rapidement.

VI-3. Recherche dans une liste triée

VI-3. Recherche dans une liste triée

Problème : Étant donné un tableau T trié et un élément a, trouver le plus petit indice i tel que $a \leqslant T[i]$.

Idée 1 : parcourir le tableau dans l'ordre pour repérer i. Pas plus efficace que si la liste n'est pas triée.

VI-3. Recherche dans une liste triée

Problème : Étant donné un tableau T trié et un élément a, trouver le plus petit indice i tel que $a \leqslant T[i]$.

Idée 1 : parcourir le tableau dans l'ordre pour repérer i. Pas plus efficace que si la liste n'est pas triée.

Idée 2 : Dichotomie, en coupant à chaque étape le tableau en 2, afin de n'en garder que la moitié pertinente. À chaque étape, on réduit le nombre de possibilités de moitié.

Algorithme 12 : Recherche dans tableau trié

Entrée : T, tableau de taille n, trié ; x élément à rechercher
Sortie : i : indice minimal tel que $T[i] \geqslant x$

$a \leftarrow 0$, $b \leftarrow n-1$;
si $T[b] < x$ **alors**
  renvoyer (n)
sinon si $T[a] \geqslant x$ **alors**
  renvoyer (0)
sinon
  tant que $b - a > 1$ **faire**
    $c = \lfloor \frac{a+b}{2} \rfloor$;
    si $T(c) \geqslant x$ **alors**
      $b \leftarrow c$
    sinon
      $a \leftarrow c$
    fin si
  fin tant que
fin si
renvoyer (b)

Exercice 10

1. Prouver la terminaison et la correction de cet algorithme

Exercice 10

1. Prouver la terminaison et la correction de cet algorithme
2. Justifier que sauf dans le cas où les tests initiaux sont positifs, son coût est en $\Theta(\ln(n))$.

Exercice 10

1. Prouver la terminaison et la correction de cet algorithme
2. Justifier que sauf dans le cas où les tests initiaux sont positifs, son coût est en $\Theta(\ln(n))$.
3. Expliquer comment améliorer l'algorithme si on ne recherche pas nécessairement la première occurrence.

Exercice 10

1. Prouver la terminaison et la correction de cet algorithme
2. Justifier que sauf dans le cas où les tests initiaux sont positifs, son coût est en $\Theta(\ln(n))$.
3. Expliquer comment améliorer l'algorithme si on ne recherche pas nécessairement la première occurrence.
4. Expliquer comment récupérer la dernière occurrence.

Exercice 10

1. Prouver la terminaison et la correction de cet algorithme
2. Justifier que sauf dans le cas où les tests initiaux sont positifs, son coût est en $\Theta(\ln(n))$.
3. Expliquer comment améliorer l'algorithme si on ne recherche pas nécessairement la première occurrence.
4. Expliquer comment récupérer la dernière occurrence.
5. Comment récupérer l'ensemble de toutes les occurrences ?

Autres algorithmes

Autres algorithmes

- Recherche d'un motif dans un texte

Autres algorithmes

- Recherche d'un motif dans un texte
- Tris

Autres algorithmes

- Recherche d'un motif dans un texte
- Tris
- Tous les algorithmes d'analyse numérique