Algorithme 1

Notion de complexité algorithmique

La **complexité algorithmique** consiste en l'étude formelle de la quantité de ressources (par exemple de temps ou d'espace) nécessaire à l'exécution de cet algorithme.

La quantité en temps est le temps d'exécution de l'algorithme en fonction des données. Si ce temps est trop long, l'algorithme devient inutilisable.

La quantité en espace est la place mémoire nécessaire pour que l'algorithme fonctionne. Si la quantité de mémoire nécessaire est trop importante, l'algorithme devient difficile à utiliser car, les ordinateurs ont toujours de la mémoire en quantité finie.

Nous nous intéresserons uniquement à la complexité en temps dans le pire des cas. C'est-à-dire dans les situations où l'algorithme sera le plus long.

Nous avons vu quatre types de complexité.

La complexité linéaire ou en $\mathrm{O}(n)$ (se lit « grand o de *n* »).

- **Exemples d'algorithmes :** pire des cas dans la recherche séquentielle d'un élément dans une liste, l'utilisation de l'algorithme de tri par insertion sur une liste triée.
- **Ordre de grandeur :** Le temps d'exécution est linéaire, c'est-à-dire proportionnel à la taille des donnés. Par exemple, si un tableau de taille 1 000 demande 3 s d'exécution alors un tableau de taille 4 000 demandera $3\times 4=12$ s d'exécution.

La complexité peut être quadratique ou en $\mathrm{O}(n^2)$

- **Exemples d'algorithmes :** tri par sélection ou du tri par insertion dans le pire des cas.
- **Ordre de grandeur :** Un algorithme quadratique demande pour traiter une liste de taille *n* de l'ordre de n^2 opérations. Pour une liste de taille *2n*, le même algorithme aura besoin de l'ordre de $(2n)^n=4\times n^2$ opérations. En doublant la taille de la liste, on quadruple le nombre d'opérations. Par exemple, si une liste de taille 1 000 demande 3 s d'exécution, alors une liste de taille 4 000 demandera $4^2\times 3=16\times 3=48\,s$

La complexité peut être logarithmique ou en $\mathrm{O}(\log_2(n))$.

- **Exemples d'algorithmes :** la recherche dichotomique.
- **Ordre de grandeur :** Par exemple utiliser la recherche par dichotomie dans une tableau de taille *n* va se faire dans le pire des cas en $\log_2(n)$ étapes. Ce qui correspond approximativement aux nombres de bits nécessaires pour écrire *n* en binaire. Par exemple, si pour une liste de taille 128 il faut 3 s, alors pour une liste de taille $128^2=16\,384$ il

faudra $2\times3=6$ s. Autrement dit, appliquer un algorithme logarithmique sur une liste de taille 2^n demande de l'ordre de *n* étapes ; l'appliquer sur une liste de taille $2^n\times2=2^{n+1}$ demandera $n+1$ étapes.

La complexité peut être exponentielle, ou en $O(2^n)$.

- **Exemples d'algorithmes :** Explorer toutes les possibilités dans le problème du sac à dos. Ce problème consiste à remplir un sac à dos qui peut contenir au plus un certain poids avec des objets de valeurs et de poids connus afin que la valeur du contenu du sac à dos soit maximale. Si l'on explore toutes les possibilités de pour chaque objet de la liste on peut coder qu'il soit dans le sac à dos avec un 1 et avec un 0 s'il n'y est pas. S'il y a cinq objets une possibilité est 10111.
- **Ordre de grandeur :** Le nombre de possibilités équivaut au nombre de nombres entiers que l'on peut écrire sur 5 bits soit $2^5=32$. Pour n objets il y aurait 2^n possibilités. Pour chaque objet supplémentaire i y a deux fois plus de possibilités à explorer.

Les algorithmes les plus rapides sont donc de complexité logarithmes, suivies des algorithmes de complexité linéaire et enfin des algorithmes de complexité quadratique.
Dans la mesure du possible, les algorithmes de complexité exponentielle sont à éviter. Il existe d'autres complexités algorithmique, par exemple en $n\ln(n)$ pour de bons algorithmes de tri, qui ne seront pas étudiées ici.