Vous savez désormais calculer le nombre d'opérations effectuées par un algorithme ainsi que la complexité de celui-ci. Il nous reste à voir comment faire le lien avec son temps de calcul. En effet après avoir étudié un algorithme on souhaiterait tout de même savoir si une implémentation de celui-ci (un programme) va plutôt mettre 1 seconde ou 1 heure à s'exécuter. Comme nous l'avons vu cela dépend d'un certain nombre de facteurs (le langage utilisé, la machine utilisée...) donc on ne pourra faire que des estimations, pas un calcul exact.

Vitesse de calcul d'un processeur

Le processeur est l'unité de calcul de l'ordinateur. Vous pouvez l'imaginer comme une grosse boîte noire qui va prendre des données (sous forme de signaux électriques) et donner des résultats (toujours sous la forme de signaux électriques). Les données peuvent venir de la mémoire de l'ordinateur (ou de celle du processeur) et correspondent par exemple à la valeur de certaines variables. Le processeur peut faire un certain nombre de calculs sur les données, par exemple les opérations mathématiques de base.

Le fait de prendre des données, faire un calcul, et donner un résultat s'appelle un *cycle de calcul* du processeur. Un processeur est capable de faire un très grand nombre de cycles de calcul à chaque seconde. On mesure ce nombre en "Hz" qui est une unité de mesure indiquant un "nombre de fois par seconde", c'est-à-dire une fréquence. Comme le processeur est très rapide, on utilise plutôt (de nos jours) le GHz : 1GHz correspond à 1 milliard de cycles par seconde ! Le processeur de votre ordinateur a typiquement une fréquence de quelques GHz (de 1 à 4).

Un cours d'un cycle de calcul, le processeur est capable de calculer des choses simples (comme une addition), mais il lui faudra parfois plusieurs cycles pour calculer des choses plus complexes comme une division. Dès qu'il faut aller lire ou modifier la valeur d'une variable il faut également plus de temps car accéder à la mémoire est beaucoup plus lent : le temps d'aller lire la valeur d'une variable, le processeur à eu le temps d'exécuter plusieurs dizaines de cycles !

En pratique les choses sont un plus complexes avec les processeurs actuels (multi-coeurs, multi-threads, caches...) que cette petite explication mais elle est suffisamment proche de la réalité pour que vous puissiez utiliser ce modèle simplifié d'un processeur dans vos raisonnements.

Temps de calcul

Si on considère un programme typique effectuant des opérations diverses (lecture, calculs, écritures) alors le nombre d'opérations pouvant être effectuées à chaque seconde sur un processeur à 1GHz est d'environ **1 à 10 millions**.

Notez que ce nombre peut monter à 100 millions si on ne fait que des opérations mathématiques simples ou descendre si on utilise un langage assez lent comme Python. Il faudra donc adapter ce nombre "le mieux possible", selon le programme qu'on étudie et le langage qu'on utilise. En pratique vous verrez qu'on est rarement à un facteur 10 près quand il s'agit de choisir le bon algorithme. Vous

verrez dans le chapitre sur la "Complexité avancée" comment mesurer le temps pris par les opérations de base, afin de pouvoir faire des estimations plus précises.

Sachant combien d'opérations peuvent être faites en une seconde sur le processeur, il nous suffit donc d'estimer le nombre d'opérations que va effectuer le programme pour avoir une estimation de son temps de calcul : on applique une simple règle de proportionalité ! On va donc diviser le nombre d'opérations estimé par 1 à 10 millions (pour un processeur à 1GHz), pour obtenir le temps de calcul. Tout ceci n'est pas très précis mais fonctionne bien en pratique pour les exercices que nous vous poserons, avec une limite de temps pour l'exécution de votre programme. La question est alors bien souvent de savoir si on doit utiliser un algorithme linéaire ou quadratique (ou une autre complexité) afin que le programme se termine assez vite.

Sur un exemple, supposons que le nombre d'opérations de l'algorithme vaut environ *5N* et que *N* vaut 100 000 alors :

- un algorithme linéaire fera 500 000 (5*100 000) opérations soit entre 0.05s (500 000/10 millions) et 0.5s (500 000/1 million) de temps de calcul,
- un algorithme quadratique fera 50 000 000 000 (5*100 000*100 000) opérations soit entre environ 1h20 et 13h de temps de calcul.

La différence est tellement grande que le calcul approché que nous avons fait n'est pas un problème.

Dans chacun des cas ci-dessous, indiquez quel(s) algorithme(s) peuvent être utilisés si on veut respecter la contrainte de temps.

Exercice 1

Pour un certain exercice, on a deux grandeurs 1<=N<=1000 et 1<=M<=100 000. Quels algorithmes peuvent, a priori, s'exécuter en moins de 1s sur un processeur à 1Gz ?

1. Un algorithme faisant environ N*M opérations de base.
2. Un algorithme faisant environ 10M opérations de base.
3. Un algorithme faisant environ (N*M)/100 opérations de base.
4. Un algorithme faisant environ N*M/20+50M opérations de base.
5. Un algorithme faisant environ M/N opérations de base.

Afficher/cacher la solution

Solution de l'exercice :

Dans chaque cas, calculons le nombre d'opérations (pour les valeurs de *N* et *M* donnant le plus grand résultat) puis une valeur approchée du temps de calcul, en considérant 1 ou 10 millions d'opérations par seconde.

> Dans les 4 premiers cas, le plus grand nombre d'opération est obtenu quand N et M sont les plus grands possibles :

1. On a environ 100 millions d'opérations, donc environ 10s à 100s de calcul. A priori, il ne pourra pas s'exécuter en une seconde.
2. On a environ 1 million d'opérations, donc environ 0.1s à 1s de calcul. A priori, il pourra s'exécuter en une seconde.
3. On a environ 1 million d'opérations, donc environ 0.1s à 1s de calcul. A priori, il pourra s'exécuter en une seconde.
4. On a environ 5+5=10 millions d'opérations, donc environ 1s à 10s de calcul. A priori, il ne pourra pas s'exécuter en une seconde.

5. Dans le pire cas, N vaut 1 et M vaut 100 000. On a environ 100 000 opérations, donc environ 0.01s à 0.1s de calcul. A priori, il pourra s'exécuter en une seconde.

Estimation plus précise

Nous avons vu qu'il était difficile d'estimer précisément à l'avance le temps de calcul d'un algorithme à cause des différents facteurs (le langage, la machine...) qui interviennent. Cependant, il suffit en général de connaître la complexité de l'algorithme et **un** temps d'exécution de l'algorithme pour pouvoir estimer les choses bien plus précisement.

En effet on sait que si un algorithme est linéaire en *N* alors son temps de calcul est tout simplement multiplié par 2 si la valeur de *N* est multipliée par 2. Si le temps de calcul de l'algorithme est de 1s pour *N=1000* alors pour *N=2000* le temps de calcul sera de 2s et pour *N=10000* il sera de 10s.

A vous de jouer sur les exemples ci-dessous :

Exercice 2

Un algorithme linéaire en *N* a mis 1 seconde à s'exécuter pour *N=100 000*. Quel sera le temps de calcul si N vaut 100 millions ? On souhaite que l'algorithme termine en moins d'une heure, quelle est la plus grande valeur de *N* pour laquelle c'est le cas ?

Afficher/cacher la solution

Solution de l'exercice :

Comme l'algorithme est linéaire, son nombre d'opérations est proportionnel à *N*. Quand on passe de *N=100 000* à *N=100 000 000*, on multiplie par 1000 la valeur de *N*, donc par 1000 le nombre d'opérations et donc le temps de calcul. Il mettra donc 1000s à s'exécuter.

Pour passer de 1s à 1 heure, on a multiplié le temps de calcul (donc le nombre d'opérations) par 3600. Comme l'algorithme est linéaire en *N*, on doit donc multiplier *N* par 3600 c'est-à-dire *N=360 000 000*.

Exercice 3

Un algorithme quadratique en *N* a mis 3 secondes à s'exécuter pour *N=100 000*. Quel sera le temps de calcul si N vaut 1 million ? On souhaite que l'algorithme termine en moins d'une minute, quelle est la plus grande valeur de N pour laquelle c'est le cas ?

Afficher/cacher la solution

Solution de l'exercice :

Comme l'algorithme est quadratique, son nombre d'opérations est proportionnel à N^2. Quand on passe de *N=100 000* à *N=1 000 000*, on multiplie par 10 la valeur de *N*, donc par 100 le nombre d'opérations et donc le temps de calcul. Il mettra donc 300s à s'exécuter.

Pour passer de 3s à 1 minute, on a multiplié le temps de calcul (donc le nombre d'opérations) par 20. La question est donc de savoir par combien il faut multiplier *N* pour que N^2 soit multiplié par 20. On cherche donc une valeur qui, élevée au carré, donne 20. C'est donc la racine carrée de 20, qui vaut environ 4.5. La plus grande valeur de *N* possible est donc environ 450 000.

Exercice 4

Un algorithme dont la complexité est proportionnelle à $N\sqrt{N}$ a mis 10 secondes à s'exécuter pour *N=1 000 000*. Quel sera le temps de calcul si N vaut 10 milliards ? On souhaite que l'algorithme termine en moins de 24 heures, quelle est la plus grande valeur de *N* pour laquelle c'est le cas ?

Afficher/cacher la solution

Solution de l'exercice :

Quand *N* passe de 1 million à 10 milliards, on multiplie par 10000 sa valeur, donc le nombre d'opérations passe de $N\sqrt{N}$ à $10000N\sqrt{10000N}$ soit $10000\sqrt{10000}N\sqrt{N}$. Au final le nombre d'opérations (donc le temps de calcul) a été multiplié par 1 million donc l'algorithme mettra 10 millions de secondes à s'exécuter soit environ 116 jours !

Pour passer de 10s à 24 heures (soit 3600*24=86400 secondes), on a multiplié le temps de calcul (donc le nombre d'opérations) par environ 8600. La question est donc de savoir par combien il faut multiplier *N* pour que $N\sqrt{N}$ soit multiplié par 8600. Supposons qu'on multiplie *N* par *k* alors on doit avoir $kN\sqrt{kN} = 8600N\sqrt{N}$ c'est-à-dire $k\sqrt{k} = k^{3/2} = 8600$ et donc $k = 8600^{2/3}$ soit environ 420. La plus grande valeur de N possible est donc environ 420 millions.

Complexité et nombre d'opérations

Pour estimer le temps de calcul nous avons donc besoin du nombre d'opérations effectué par l'algorithme. Nous n'avons pas besoin du nombre exact d'opérations, juste d'une estimation pas trop mauvaise. Par exemple si un algorithme est linéaire en *N* il faut estimer si le nombre d'opération est plus proche de *10N* ou *1000N*. Que ce soit *10N* ou *15N* ne changerait pas grand chose en pratique !

Par contre entre un algorithme faisant *10N* opérations et un autre en faisant *1000N* la différence est notable, même si la complexité est à chaque fois proportionnelle à *N*. Si le facteur devant la variable (le 10 ou le 1000), ce qu'on appelle la *constante de la complexité*, est très grand, il est préférable de dire que l'algorithme à une complexité proportionnelle à *N* **avec une grande constante** (voir indiquer explicitement la constante). Cela permet de mieux décrire la complexité de l'algorithme.