ALGO 1 - **Alice déménage mais à quel coût ?**

I – Définitions

1 - ALGORITHME

Définition : Un algorithme est un ensemble **fini** (au sens "pas infini") d'instructions **précises** permettant de résoudre **une classe de problèmes**.

L'algorithme doit être écrit **sans ambiguïté**, et **être transposable** dans un langage de programmation.

2 – Introduction au COÛT

2.1 Définition : le coût exprime la façon dont l'algorithme réagit à l'augmentation du nombre n de données à traiter.

On l'évalue d'abord dans le PIRE DES CAS, la pire situation qui puisse arriver.

2.2 Exemple de coût

Cas de la recherche aléatoire d'un carton précis :

Dans le pire des cas, temps infini si n est trop grand.

Cas de la recherche linéaire :

Si on a 1000 cartons, on aura au pire 1000 recherches.

→ Coût LINÉAIRE (noté n ou LIN)

Le meilleur des coûts est le coût CONSTANT (noté 1 ou CST) **:** le temps de réponse ne dépend pas du nombre de données.

2.3 Justification pour la recherche linéaire :

POUR chaque carton de gauche à droite (on le fait n fois)

Ouvre le carton. (coût CST)

Si c'est le 14 (coût CST)

Ramène les clés et arrête (coût CST)

Fin de SI

Coût CST*3

Coût n*CST*3

Fin du POUR

Dis que tu n'as pas trouvé (coût CST)

Coût CST

Le coût total est donc

n * (3 * CST) + CST

= n * 3*CST + CST

= **n** * 3 + 1 (en remplaçant la notation CST par 1)

Au total, le coût dépend de **n** : il est à coût LINÉAIRE .

2.4 Justification du coût du chargement du camion :

On réalise **n** fois une recherche linéaire (qui est à coût linéaire **n**) :

D'où **n** * **n**

$= n^2$

Donc coût **QUADRATIQUE**.

Si n est multiplié par 2 alors le temps est multiplié par 2^2 donc 4.

Si n est multiplié par 3 alors le temps est multiplié par 3^2 donc 9.

2.5 Exercice (voir exo 5 sur le site pour l'énoncé) :

L'analyse des opérations élémentaires donne ceci :

$$1 + n * (1 + 1) + 1$$
$$= 1 + n * 2 \quad + 1$$
$$= n*2 + 2$$

Au total l'algorithme est à coût LINEAIRE car il dépend de n.

3 - Définition : Algorithmique

L'algorithmique consiste à étudier des problèmes en vue de les résoudre à l'aide d'un algorithme le plus efficace possible.

On cherche donc :

- à estimer la rapidité d'exécution de l'algorithme.

  Cela revient à connaître **le coût de l'algorithme**.

- à vérifier que la réponse éventuelle soit juste.

  Cette vérification se nomme **la preuve de correction**.

- à vérifier qu'il s'arrête bien sur n'importe quelle entrée fournie.

  Cette vérification se nomme **la preuve de terminaison**.

II – Alice triche !

1 - Principe de la recherche dichotomique

On trie les données en ordre croissant.

On regarde celle du milieu et on peut donc :

* soit conclure qu'on vient de trouver la bonne valeur

* soit supprimer cette valeur et toutes celles à sa gauche

* '' '' '' droite

A chaque étape, on ne garde approximativement que la moitié des données restantes.

Multiplier les données par deux rajoute juste une étape supplémentaire !

2 - Coût logarithmique

Lorsque **n est doublé**, on réalise **une seule opération** en plus.

On nomme cela le **logarithme base 2**.

```
>>> from math import log2
>>> log2(1)
0.0                car 1 s'écrit 2^0
>>> log2(2)
1.0                car 2 s'écrit 2^1
>>> log2(4)
2.0                car 4 s'écrit 2^2
>>> log2(8)
3.0                car 8 s'écrit 2^3
>>> log2(1000000)
19.931568569324174  car 1000000 s'écrit 2^19.931568569324174...
```

Remarque : il suffit de 20 recherches avec 1 million de cartons triés.

3 - Fonction logarithmique base 2

Le $\log_2$ renvoie l'exposant X du nombre N écrit sous la forme $N = 2^X$

On dit que $\log_2(x)$ et 2^x sont **réciproques** car appliquer l'une sur l'autre, on retrouve le résultat initial :

$$\log_2(2^x) = x$$

$$2^{\log_2(x)} = x$$

III Optimiser, c’est compliqué

Coût de la recherche du meilleur chargement.

On choisit un carton parmi les 12, puis un parmi les 11 puis…

Le nombre de possibilités est **12 x 11 x 10 x … 2 x 1**

On obtient 479 001 600 possibilités.

On note ce calcul **12!** et on le nomme **factorielle de 12**.

Le **coût factoriel n!** est donc pire que tous les autres…

```
>>> from math import factorial
>>> factorial(12)
479001600
>>> factorial(24)
620448401733239439360000
```

CONCLUSION (coller la feuille récap fournie : ne pas recopier !)

ALGO 1 - DOCUMENT Bilan des coûts

(du meilleur coût au moins bon coût)

PROBLEME FACILE : réponse en un temps raisonnable

- Le coût **constant (lié à 1)**
  - **Si n double, le nombre d'actions reste le même**

- Le coût **logarithmique (base 2)**
  - **Si n double, on aura un nombre constant d'opérations en plus** (un sur l'exemple).
  - Avec n = 6, on a 5 actions par exemple.
  - Avec 2 fois plus (n=12) , on aura 5+1 = 6 actions.
  - Avec 4 fois plus (n = 24), on aura 5 +2 = 7 actions.
  - Avec 8 fois plus (n = 48), on aura 5 +3 = 8 actions.

- Le coût **linéaire** (lié à **n**)
  - **Si n double, le nombre d'actions double.**
  - Avec n = 6, on a 5 actions par exemple.
  - Avec 2 fois plus (n =12) , on aura 2*5 = 10 actions.
  - Avec 4 fois plus (n =24) , on aura 4*5 = 20 actions.
  - Avec 8 fois plus (n =48) , on aura 8 *5 = 40 actions.

- Le coût **quadratique** (lié à $\mathbf{n^2}$)
  - **Si n double, le nombre d'actions est multiplié par 4.**
  - Si n triple, le nombre d'actions est multiplié par 9.
  - ect ...
  - Avec n = 6, on a 5 actions par exemple.
  - Avec 2 fois plus (n =12) , on aura 4*5 = 20 actions.
  - Avec 4 fois plus (n =24) , on aura 16*5 = 80 actions.
  - Avec 8 fois plus (n =48) , on aura 64*5 = 320 actions.

- Les coûts **polynomiaux** (lié à $\mathbf{n^k}$)
  - **Si n double, le nombre d'actions est multiplié par une constante.**
  - coût linéaire : k = 1 → données x2 → opérations x2
  - coût quadratique : k = 2 → données x2 → opérations x4
  - coût cubique : k = 3 → données x2 → opérations x8 …

PROBLEME DIFFICILE : réponse mais en un temps déraisonnable

- Le coût **exponentiel** (lié à $k^{\mathbf{n}}$) : avec un tel coût, on estime que l'algorithme n'est pas exploitable. Il va mettre trop de temps pour répondre.
  - Notez bien que **n est l'exposant**, ce n'est pas polynomial.
  - Nous prendrons ici l'exemple simple de k=2 : $\mathbf{2^n}$.
  - Avec n = 6, on a $2^6 = 64$ actions.
  - Avec 2 fois plus (n =12) , on a $2^{12} = 4096$ actions.
  - Avec 4 fois plus (n =24) , on a $2^{24} = \mathbf{16}777216$ actions.

- Avec 8 fois plus (n =48) , on a 2^{48} = **28147497 6**710656

- Le coût **factoriel** (lié à la fonction factorielle **n!**) : encore moins exploitable
  - Il s’agit de calculer n * (n-1) * (n-2) … jusqu’à 2 * 1.
  - Avec n = 6, 6 ! = 6*5*4*3*2*1 = 720 actions
  - Avec 2 fois plus , on a 12! = **479**001600 actions.
  - Avec 4 fois plus , on a 24! = **620448401733239439**360000
  - Avec 8 fois plus , on a 48! = 124139155925360726708622890473733750385214863546777600000000000

PROBLEME INDECIDABLE

C’est un problème pour lequel il est impossible de créer un algorithme qui réponde à chaque fois sans risquer de boucler à l’infini.

Pour certaines entrées, on peut donc avoir une réponse, mais pour d’autres boucler…

En conséquence, si on n’a pas encore eu de réponse, on ne peut pas savoir :

- si la machine travaille et va répondre… un jour
- si la machine est partie définitivement sur une boucle sans fin…

ALGO 1 - DOCUMENT Bilan des coûts (du meilleur coût au moins bon coût)

Sur cette fiche : **n désigne le nombre de données** et **op le nombre d'opérations élémentaires pour répondre.**

PROBLEMES FACILES : réponse en un temps **raisonnable**

- Le coût **constant** **(noté 1** ou **CST)** : si **n*2** alors **op inchangé**

- Le coût **logarithmique 2** **(**noté $\log_2 n$ ou **lg n) :** si **n*2** alors **op+1**
  - Avec n=6, on a op=5 opérations par exemple.
  - Si **n*2** (6→ 12 données) alors **op = 5+1 = 6** opérations.
  - Si **n*4** (6→ 24 données) alors **op = 5+2 = 7** opérations.
  - Si **n*8** (6→ 48 données) alors **op = 5+3 = 8** opérations.

- Les coûts **polynomiaux** (notés n^k) : si **n*2** alors **op*constante**
  si **n*2** alors **op*2^k**

- ----> dont le coût **linéaire** (noté **n**) : si **n*2** alors **op*2** = $op*2^1$
  - Correspond au cas polynomial k=1
  - Avec n=6, on a op=5 opérations par exemple.
  - Si **n*2** (6→ 12 données) alors **op = 5*2 = 10** opérations.
  - Si **n*4** (6→ 24 données) alors **op = 5*4 = 20** opérations.
  - Si **n*8** (6→ 48 données) alors **op = 5*8 = 40** opérations.

- ---→ dont le coût **quadratique** (noté n^2) : si **n*2** alors **op*4** = $op*2^2$
  - Correspond au cas polynomial k=2
  - Avec n=6, on a op=5 opérations par exemple.
  - Si **n*2** (6→ 12 données) alors **op = 5*2² = 20** opérations.
  - Si **n*4** (6→ 24 données) alors **op = 5*4² = 80** opérations.
  - Si **n*8** (6→ 48 données) alors **op = 5*8² = 320** opérations.

- ---→ puis les autres coûts **polynomiaux** (notés n^k) avec k ≥ 3
  si **n*2** alors **op*2^k**

PROBLEMES DIFFICILES : réponse mais en un temps **déraisonnable**

- Le coût **exponentiel 2** (noté 2^n) : si **n*2** alors **op*(2^n)** si **n+1** alors **op*2**
  - Notez bien que **n est l'exposant,** ce n'est pas polynomial.
  - Avec n = 6, on a 2^6 = 64 actions.
  - Si on passe de 1 à 2 données (n+1 avec n=1), on a op*2 soit 5*2=10
  - … de 2 à 3 données (n+1 avec n=2), on a op*2 soit 10*2=20
  - … de 3 à 4 données (n+1 avec n=3), on a op*2 soit 20*2=40
  - … de 4 à 8 données (n*2 avec n=4), on a op*2^4 soit 40*16=640
  - … de 8 à 16 données (n*2 avec n=8), on a op*2^8 soit 640*256=163840

- Le coût **factoriel** (noté **n!**) : si **n*2** alors **plus d'opérations que op*((n+1)^n)**
  - Par définition **n ! = n * (n-1) * (n-2) … 3 * 2 * 1**.
  - Avec n = 6, 6 ! = 6*5*4*3*2*1 = 720 actions
  - Avec n = 12, 12! = **479**001600 actions.
  - Avec n= 24, 24! = **620448401733239439**360000
  - Avec n = 48, 48! = 124139155925360726708622890473733750385214863546777600000000000

PROBLEME INDECIDABLE : boucle infinie sur certaines entrées !

C'est un problème pour lequel il est impossible de créer un algorithme qui réponde à chaque fois sans risquer de boucler à l'infini.

Pour certaines entrées, on peut donc avoir une réponse, mais pour d'autres boucler… En conséquence, si on n'a pas encore eu de réponse, on ne peut pas savoir :

- si la machine travaille et va répondre… un jour
- si la machine est partie définitivement sur une boucle sans fin…

IV – Problème NP (optionnel)

Caractéristiques des problèmes NP

1. Vérifier qu'**une proposition est une solution** du problème **est facile**

   *Facile ? En informatique, cela veut dire **strictement moins qu'exponentiel**.*

2. Par contre, **rechercher la solution optimale** parmi toutes les solutions possibles est **difficile**

   *Difficile ? En informatique, cela veut dire **exponentiel ou plus**.*

3. Personne n'a réussi à démontrer qu'il n'existe PAS d'algorithme plus rapide pour trouver la solution optimale.

4. Il a été démontré que s'il existe un algorithme efficace sur un problème NP-complet, il existe des solutions efficaces pour tous !

5. Une petite simplification de l'énoncé du problème le rend résolvable facilement...