Plan du cours

1. Introduction
   - Temps "constant" VS temps dépendant de la taille des données
   - Un premier exemple : recherche de maximum
   - Un autre exemple simple : `in`
2. Complexité d'un algorithme
   - Complexité temporelle
   - Complexité spatiale
3. Exemples de calcul de complexité temporelle
   - Outils mathématiques
   - Tri sélection
   - Peut-on faire mieux

Opérations élémentaires et "temps constant"

- On dira qu'une instruction et effectuée en temps constant si son temps d'exécution peut-être considéré comme ne dépendant pas de la taille des données.
- Par exemple : les opérations d'affectation, d'additions, multiplications, test d'égalité, lecture d'un élément dans une liste ... sont effectuées en temps constant.

Opérations élémentaires et "temps constant"

- On dira qu'une instruction et effectuée en temps constant si son temps d'exécution peut-être considéré comme ne dépendant pas de la taille des données.
- Par exemple : les opérations d'affectation, d'additions, multiplications, test d'égalité, lecture d'un élément dans une liste ... sont effectuées en temps constant.
- On voit néanmoins sur les exemples précédents que rien ne dit que ces opérations s'effectuent dans des temps identiques.
- On peut donc les compter pour 1 au sens du "nombre d'opérations en temps constant" ou $O(1)$ pour tenir compte du fait précédent, et aussi simplifier certains calculs .

Opérations élémentaires et "temps constant"

- On dira qu'une instruction et effectuée en temps constant si son temps d'exécution peut-être considéré comme ne dépendant pas de la taille des données.
- Par exemple : les opérations d'affectation, d'additions, multiplications, test d'égalité, lecture d'un élément dans une liste ... sont effectuées en temps constant.
- On voit néanmoins sur les exemples précédents que rien ne dit que ces opérations s'effectuent dans des temps identiques.
- On peut donc les compter pour 1 au sens du "nombre d'opérations en temps constant" ou $O(1)$ pour tenir compte du fait précédent, et aussi simplifier certains calculs .
- Cela ne signifie bien sûr pas que les temps d'exécution de ces instructions seront les mêmes sur toute les machines ! Mais seulement que sur toute machine donnée, il sont bornés par des constantes.
- Par contre, la vitesse d'une machine ne peut que faire varier le temps d'exécution des opérations élémentaires "en temps constant", **pas le nombre d'opérations**.

Opérations qu'on considère comme "en temps constant".

- Les opérations arithmétiques `+,*,-,/,//`
- Les comparaisons : `<,!=,==` ...
- Les opérations logiques : `and, or, not`
- L'ajout d'un élément à une liste : `L.append()`
- L'accès à un élément d'une liste `L[.]`
- Les affectations : `a=, L[.]= ...`
- ...

Opérations qu'on considère comme "en temps constant".

- Les opérations arithmétiques `+,*,-,/,//`
- Les comparaisons : `<,!=,==` ...
- Les opérations logiques : `and, or, not`
- L'ajout d'un élément à une liste : `L.append()`
- L'accès à un élément d'une liste `L[.]`
- Les affectations : `a=, L[.]= ...`
- ...

MAIS .... la simplicité d'une fonction peut-être "apparente" ! et il y a des finesses.

- L'opération `L.pop(0)` ... n'est pas en temps constant.

Opérations qu'on considère comme "en temps constant".

- Les opérations arithmétiques `+,*,-,/,//`
- Les comparaisons : `<,!=,==` ...
- Les opérations logiques : `and, or, not`
- L'ajout d'un élément à une liste : `L.append()`
- L'accès à un élément d'une liste `L[.]`
- Les affectations : `a=, L[.]= ...`
- ...

MAIS .... la simplicité d'une fonction peut-être "apparente" ! et il y a des finesses.

- L'opération `L.pop(0)` ... n'est pas en temps constant.
- L'opération `L.pop(-1)` ... peut-être considérée comme en temps constant.

Analyse d'un cas simple : recherche de max

- On rappelle ci-dessous la fonction de recherche de max :
-
```
def max_liste(L) :
    max,k = L[0],0
    for k in range(1,len(L)) :
        if L[k]>max  :
            max,indice = L[k],k
    return max,indice
```

Analyse d'un cas simple : recherche de max

- On rappelle ci-dessous la fonction de recherche de max :
- ```
  def max_liste(L) :
      max,k = L[0],0
      for k in range(1,len(L)) :
          if L[k]>max  :
              max,indice = L[k],k
      return max,indice
  ```
- Si N est la longueur on peut essayer d'évaluer **le nombre d'opérations en temps constant** qu'elle effectue en fonction de N.
  - Ce nombre d'opérations peut **dépendre des éléments de la liste** et pas seulement de N. Par exemple pour :

    ```
    L = [100,1,2,3]
    ```

    c'est différent du cas :

    ```
    L = [1,2,3,100]
    ```
    ```

- On peut néanmoins **majorer** le nombre d'opérations ou plus précisément ici calculer le nombre d'opérations qu'on aura **dans le pire des cas**,

- On peut néanmoins **majorer** le nombre d'opérations ou plus précisément ici calculer le nombre d'opérations qu'on aura **dans le pire des cas**,
- c'est à dire le cas où le test renvoie `True` à chaque fois. La fonction effectue alors :
  - K_1 opérations en temps constant au départ (valeurs de variables, affectation, calcul d'une longueur ..)
  - Puis $N-1$ fois : la lecture de valeurs en mémoire, tests et affectations : K_2 opérations.
  - ce qui donne : $K_1 + K_2 \times (N-1) = K_2 N + K_1'$ opérations en temps constant.

- On peut néanmoins **majorer** le nombre d'opérations ou plus précisément ici calculer le nombre d'opérations qu'on aura **dans le pire des cas**,
- c'est à dire le cas où le test renvoie `True` à chaque fois. La fonction effectue alors :
  - K_1 opérations en temps constant au départ (valeurs de variables, affectation, calcul d'une longueur ..)
  - Puis $N-1$ fois : la lecture de valeurs en mémoire, tests et affectations : K_2 opérations.
  - ce qui donne : $K_1 + K_2 \times (N-1) = K_2 N + K_1'$ opérations en temps constant.
- Si on note $C(L)$ le nombre d'opérations en temps constant pour une liste L de longueur notée $|L| = N$

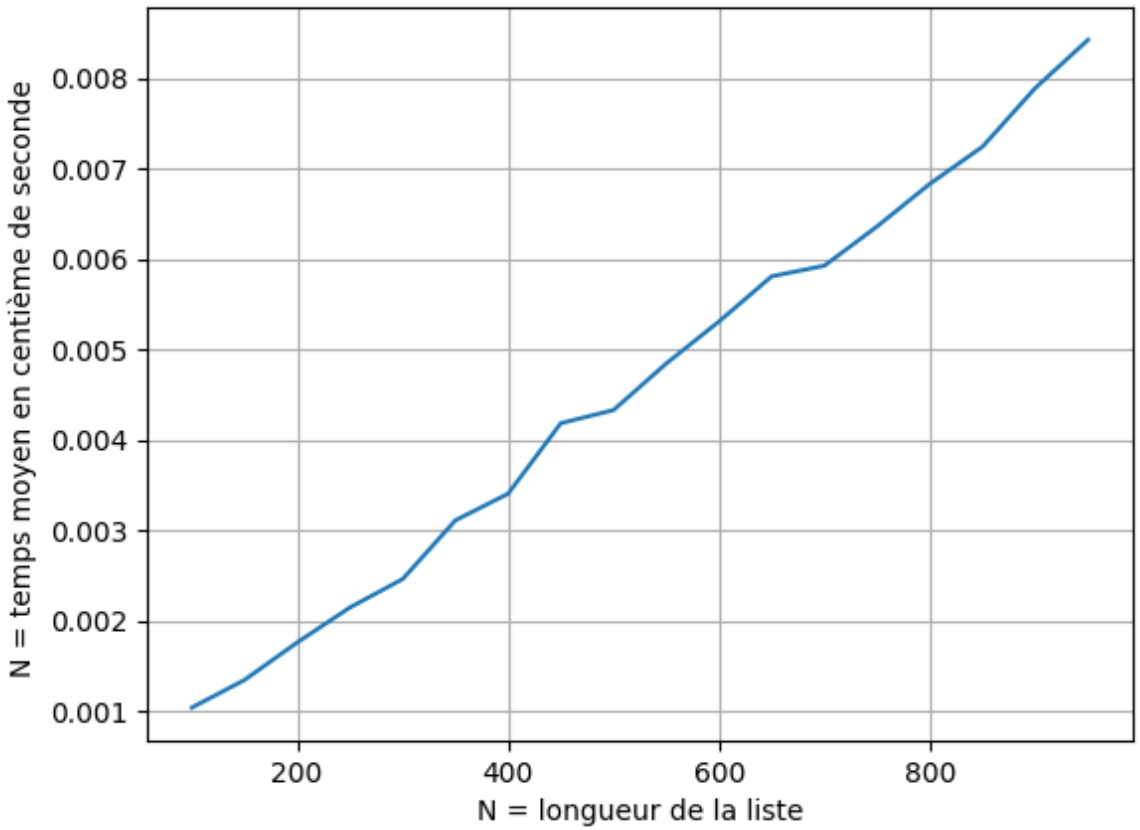
$$0 \leqslant C(L) \leqslant K_2 N + K_1' \text{ ou encore } C(L) = O(|L|)$$

- Par ailleurs, on peut par le même raisonnement voir que :

$$k_1 + k_2 N \leqslant C(L) \leqslant K_2 N + K_1' \text{ ou encore } C(L) = O(|L|)$$

Graphique : temps = f(len(L))

- Ce que "confirme" le graphique suivant obtenus par des tests de temps :



- Avec bien sûr des écarts entre la théorie et les temps mesurés ....

La commande in

- La recherche d'un élément dans une liste Python

```
>>> 7 in [2,6,-2,7,3]
```

n'est pas en temps constant.

La commande in

- La recherche d'un élément dans une liste Python

```
>>>  7 in [2,6,-2,7,3]
```

  n'est pas en temps constant.
- En fait il faut prendre un à un les éléments de la liste pour les comparer avec l'élément cherché.

La commande in

- La recherche d'un élément dans une liste Python

```
>>> 7 in [2,6,-2,7,3]
```

  n'est pas en temps constant.
- En fait il faut prendre un à un les éléments de la liste pour les comparer avec l'élément cherché.
- On aura donc un nombre d'opérations croissant linéairement avec la taille de la liste ...
- Complexité dans le pire des cas : $O(len(L))$

Definition

Estimer la complexité temporelle d'un algorithme, c'est estimer le nombre d'opérations en temps constant qui seront effectuées par cet algorithme en fonction des données d'entrée et notamment de leur la taille. On peut distinguer plusieurs évaluations de ce nombre :

1. le calcul exact.
2. une majoration/minoration/encadrement de ce nombre d'opérations
3. le calcul dans le pire des cas de données d'entrée c'est à dire celui qui va générer le plus d'opération.
4. le calcul du nombre moyen d'opérations qu'on obtiendrait si on pouvait tester sur toutes les données d'entrée possibles.

Definition

Estimer la complexité temporelle d'un algorithme, c'est estimer le nombre d'opérations en temps constant qui seront effectuées par cet algorithme en fonction des données d'entrée et notamment de leur la taille. On peut distinguer plusieurs évaluations de ce nombre :

1. le calcul exact.
2. une majoration/minoration/encadrement de ce nombre d'opérations
3. le calcul dans le pire des cas de données d'entrée c'est à dire celui qui va générer le plus d'opération.
4. le calcul du nombre moyen d'opérations qu'on obtiendrait si on pouvait tester sur toutes les données d'entrée possibles.

- Par exemple, dans le cas de la recherche du maximum d'une liste on avait vu que la complexité dépendant de cas particuliers possibles pour la liste, on a fait un calcul "dans le pire des cas" et un encadrement.
- En pratique, on est souvent dans cette configuration
  1. calcul exact difficile voir impossible.
  2. calcul en moyenne très difficile.
  3. donc dans la plupart des cas on cherchera une complexité dans le pire des cas où des majoration/minoration/encadrement.

Vocabulaire pour différents type de complexité

Dans ce qui suit $\mathtt{N}$ est la taille de la donnée que va traiter l'algorithme et à partir de laquelle on a évalué sa complexité $\mathtt{C}(\mathtt{N})$. On donne le vocabulaire usuel pour différents cas de complexité.

- $\mathtt{C}(\mathtt{N}) = \mathtt{O}(1)$: complexité constante.
- $\mathtt{C}(\mathtt{N}) = \mathtt{O}(\log(\mathtt{N}))$: complexité logarithmique
- $\mathtt{C}(\mathtt{N}) = \mathtt{O}(\mathtt{N})$: complexité linéaire
- $\mathtt{C}(\mathtt{N}) = \mathtt{O}(\mathtt{N}^2)$: complexité quadratique
- $\mathtt{C}(\mathtt{N}) = \mathtt{O}(\mathtt{N}^k)$, $k \in]2, +\infty[$: complexité polynomiale.
- $\mathtt{C}(\mathtt{N}) = \mathtt{O}(\alpha^{\mathtt{N}})$ avec $\alpha > 1$: complexité exponentielle.
- $\mathtt{C}(\mathtt{N}) = \mathtt{O}(\mathtt{N}!)$: complexité factorielle.

Questions I

Questions 1.

- Déterminer la complexité temporelle du calcul de la moyenne d'une liste de données $(x_i)_i$ en fonction du nombre d'éléments n qui la composent, puis exprimer le résultat sous la forme d'un O.

Questions II

- On considère la fonction Python ci-dessous :

```
def depasse(N) :
    n, s, k = 1, 0,1
    while s < N :
        s += 1/k
        k += 1
    return k
```

1. Expliquer graphiquement puis démontrer que pour tout entier naturel n :

$$\int_1^{n+1} \frac{1}{t} dt \leqslant \sum_{k=1}^{n} \frac{1}{k} \leqslant \int_1^n \frac{1}{t} dt + 1$$

2. En déduire un encadrement de la complexité $C(N)$ de cette fonction par rapport à l'entier N puis son expression sous la forme O.

Complexité spatiale

1. Un autre problème en informatique peut-être celui de l'espace mémoire nécessaire à l'exécution d'un algorithme.
2. Dans ce cas, il s'agit d'évaluer l'encombrement en mémoire en fonction de la taille des données d'entrée.
3. Ce problème n'a pas à être creusé dans vos programmes. On peut néanmoins donner l'exemple suivant pour un calcul de somme :

```
N,valeurs = 100, [1]
for k in range(2,N+1) :
    valeurs.append(1/k)
print(sum(valeurs))
```

Complexité spatiale

1. Un autre problème en informatique peut-être celui de l'espace mémoire nécessaire à l'exécution d'un algorithme.
2. Dans ce cas, il s'agit d'évaluer l'encombrement en mémoire en fonction de la taille des données d'entrée.
3. Ce problème n'a pas à être creusé dans vos programmes. On peut néanmoins donner l'exemple suivant pour un calcul de somme :

```
N,valeurs = 100, [1]
for k in range(2,N+1) :
    valeurs.append(1/k)
print(sum(valeurs))
```

```
N, valeurs = 100, 1
for k in range(2,N+1) :
    valeurs += (1/k)
print(valeurs)
```

4. Dans le premier cas, la création de la liste de valeurs contenant 100 entiers conduira à une occupation mémoire beaucoup plus importante en $O(N)$ que dans le deuxième cas où l'occupation mémoire ne "dépend pas"de la taille de la donnée d'entré donc sera en $O(1)$. (une seule variable utilisée pour le stockage : `valeurs`).

L'outil de $O(N)$

- On rappelle qu'un suite $N \mapsto u(N)$ est dite "en grand O de v_N"qu'on note :

$$u_N = O(v_N)$$

lorsqu'il existe $K \in \mathbb{R}$:

$$\forall N \in \mathbb{N}, |u_N| \leqslant K|v_N|$$

L'outil de $O(N)$

- On rappelle qu'un suite $N \mapsto u(N)$ est dite "en grand O de v_N"qu'on note :

$$u_N = O(v_N)$$

lorsqu'il existe $K \in \mathbb{R}$:

$$\forall N \in \mathbb{N}, |u_N| \leqslant K|v_N|$$

- On rappelle alors qu'on a en se montrant TRES indulgent sur les écritures :

$$Cte \times O(v_n) = O(v_n) \quad , \quad v_n O(u_n) = O(u_n v_n) \quad , \quad o(u_n) + O(u_n) = O(u_n)$$

Complexité du tri sélection I

- On rappelle ci-dessous la première méthode de tri que nous avons programmé cette année et qui était basé sur une fonction `max_list` permettant de déterminer la valeur maximale dans une liste de nombres.

```
def tri_naif(L) :
    longueur = len(L)
    for k in range(longueur) :
        dernier = longueur-k-1
        m = max_liste(L[0:dernier+1])
        tmp = L[m[1]]
        L[m[1]] = L[dernier]
        L[dernier] = tmp
```

Complexité du tri sélection II

- La taille des données d'entrée est uniquement définie par la longueur N de la liste L
- Dans la suite, on notera $C_{max}(N)$ le nombre d'opérations en temps constants effectuées par cette fonction dans le pire des cas.
- Dans la série d'instructions qui compose cette fonction on trouve :
  - une boucle qui sera exécutée N fois et qui fait que les nombres d'instructions en temps constants effectués à chaque tour de boucle devront être sommés dans le décompte, à savoir :
    - en temps constant : K opérations (accès à une variable, calcul, affectation)
    - en temps non constant : l'appel à `max_liste` qui va générer dans le pire des cas $K'(N-k-1)+K''$ opérations en temps constant.

Complexité du tri sélection III

- On en déduit que dans le pire des cas le nombre d'opérations est :

$$C_{max}(N) = \sum_{k=0}^{N-1} K + K'(N-k-1) + K'' = \sum_{k=0}^{N-1} (K - K' + K'') + K'(N-k)$$

Ce qui donne par linéarité de la somme :

$$C_{max}(N) = K_3 \times N + K' \sum_{k=0}^{N-1} N - k = K_3 N + K' \times \frac{N(N-1)}{2}$$

Complexité du tri sélection IV

- Ce qui donne une complexité dans le pire des cas quadratique :

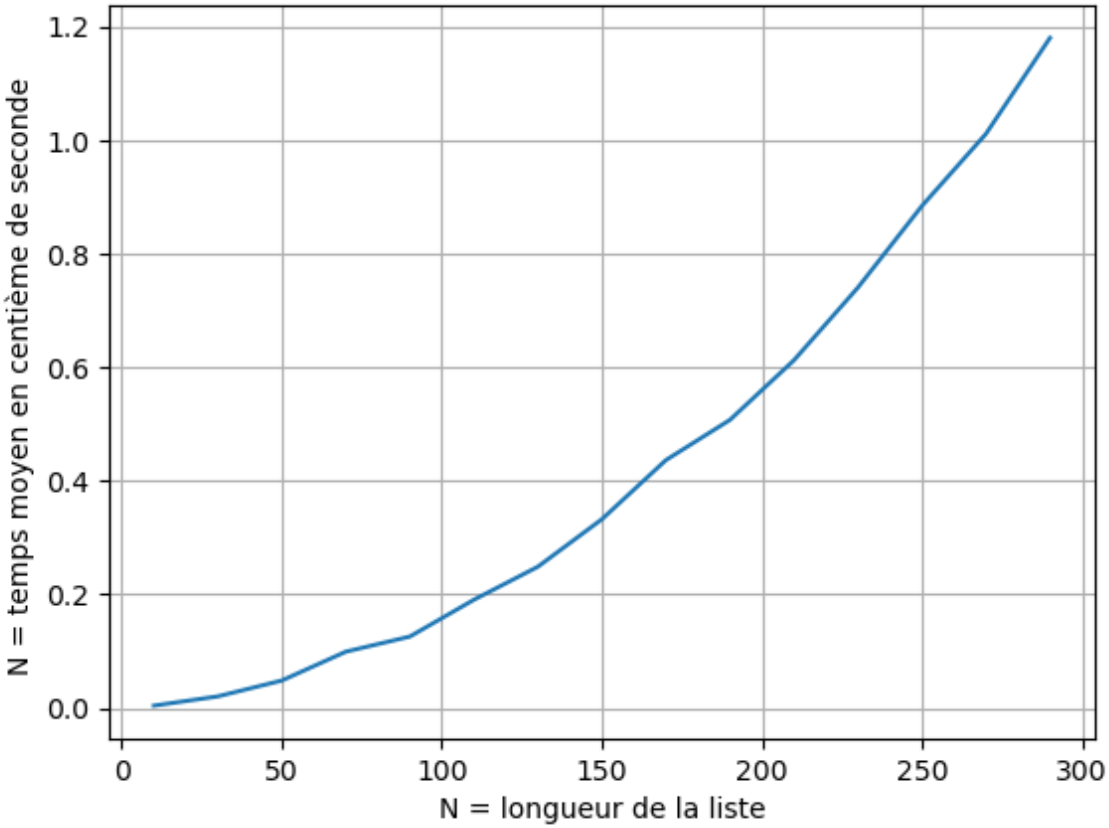
$$C_{max}(N) \sim K' N^2$$

ou plus simplement MAIS moins précisément

$$C_{max}(N) = O(N^2).$$

Complexité du tri sélection V

- En faisant des tests de temps on trouve le graphique suivant :



Estimation rapide de la complexité à l'aide de O

Pour l'estimation rapide de la complexité de la fonction précédente on peut raisonner de la manière suivante en notant N la longueur de la liste d'entrée :

1. Les instruction du corps de la boucle contiennent un appel à `max_list` qui génèrent un nombre d'opération en temps constant qui est en $O(N)$.
2. Ce corps de boucle est appelé N fois, ce qui donne une complexitée en :

$$C(N) = N \times O(N) = O(N^2)$$

3. On a donc une complexité maximale de type quadratique.

- La réponse est oui...

- La réponse est oui... . Par exemple le tri dit rapide qu'on avait programmé en TP donne les mesures de temps suivantes :

- La réponse est oui... . Par exemple le tri dit rapide qu'on avait programmé en TP donne les mesures de temps suivantes :
- Et cela ne croit pas en N^2.

- La réponse est oui... . Par exemple le tri dit rapide qu'on avait programmé en TP donne les mesures de temps suivantes :
- Et cela ne croit pas en N^2.
- En fait on peut démontrer par contre qu'on ne fera jamais mieux que $O(N \log(N)$ où N est la longueur de la liste.