1 Complexité en temps d'un algorithme :

En quel sens peut on dire ce que tel algorithme est plus efficace qu'un autre dans la résolution d'un problème donné ?

On parle d'efficacité d'algorithme pas de programme car un programme est écrit dans un langage donné et est exécuté sur une machine particulière

Il faut trouver une mesure universelle qui ne dépend ni du langage ni de la machine

Prenons par exemple le problème de la recherche d'un élément dans un tableau

Nous avons mesuré expérimentalement le temps d'exécution des fonctions suivantes sur des tableaux de taille variable de taille 1000 jusqu'à 20000, **dans le pire des cas** lorsque l'élément recherché n'est pas dans la liste, ce qui oblige de parcourir tout le tableau

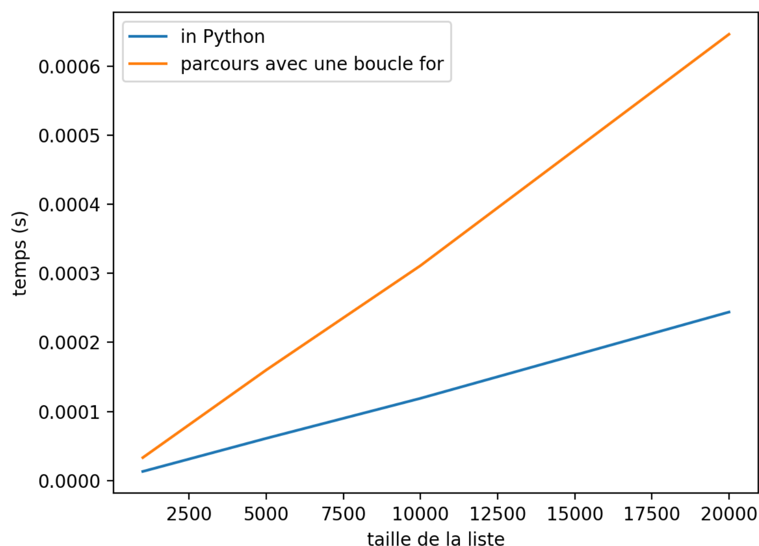
```
def recherche1(T:list,v:int)->bool:
    for element in T:
        if element == v:
            return True
    return False

def recherche2(T:list,v:int)->bool:
    return v in T
```

On observe sur des machines différentes (ici en Python mais cela reste vrai avec d'autres langages) que **le temps d'exécution est proportionnelle à la taille des listes**

On observe une différence entre les deux fonctions, l'une est plus performante que l'autre mais **les deux fonctions sont linéaires et restent dans la même famille de fonctions**

On dira que **la complexité en temps des deux programmes est linéaire en fonction de la taille**

Peut-on prévoir cette observation à partir de l'algorithme de la recherche d'un élément dans une liste ?



Algorithme 1 : Recherche séquentielle

Données : Un tableau T ayant n éléments et une valeur v
Résultat : Vrai si v est dans T, faux sinon

```
1  début
2  |  i ← 0
3  |  tant que (i < n) et (T[i] != v) faire
4  |  |  i ← i + 1
5  |  fin
6  |  si i = n alors
7  |  |  retourner Faux
8  |  sinon
9  |  |  retourner Vrai
10 |  fin
11 fin
```

Dans la boucle `Tant que` deux opérations sont répétées : une *affectation* et deux *comparaisons.*

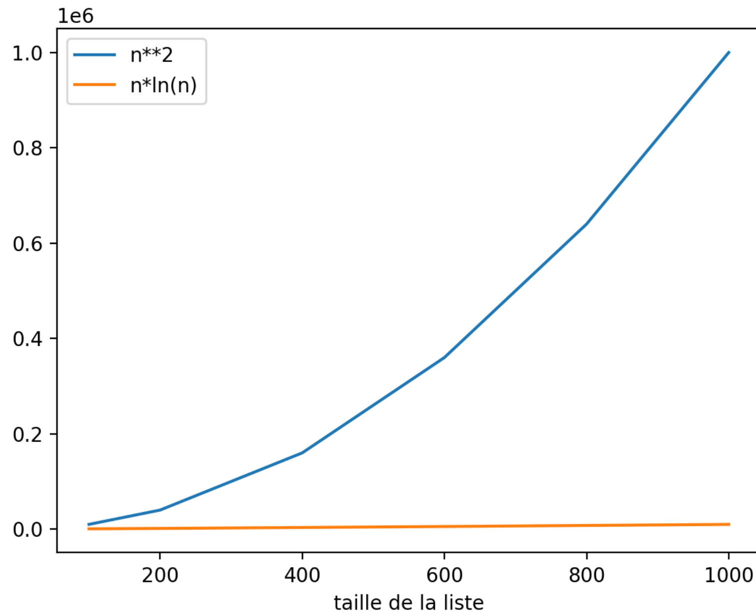
On peut supposer que le temps d'exécution d'une affectation peut varier d'un tour de boucle à l'autre mais ne dépassera pas une constante c_1, de même pour la comparaison le temps d'exécution d'une comparaison est plus petite qu'une constante c_2 à chaque tour de boucle. Combien de fois sera exécuté la boucle ?

On envisage deux cas extrêmes :

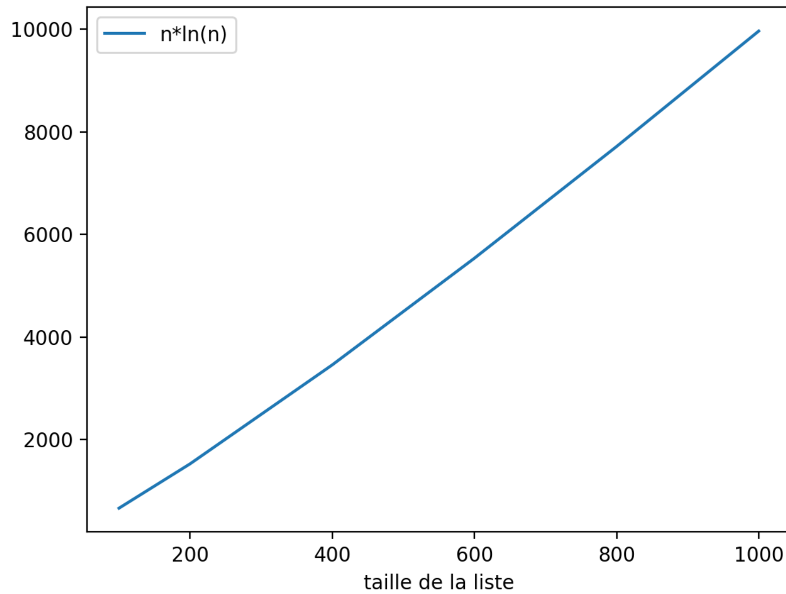
- **Au pire des cas** l'élément cherché est à la fin du tableau ou ne se trouve pas dans le tableau et dans ce cas le temps d'exécution $T(n) \leqslant nc_1 + 2nc_2 = (c_1 + 2c_2)n$. On dit que la complexité est linéaire dans le pire des cas, on dit aussi que la complexité est en O(n) (lire en grand O de n)
- **Au meilleur des cas** l'élément cherché est au début du tableau dans ce cas le temps ne dépend pas de n on dit que la complexité est en O(1)

En conclusion :

1. La complexité en temps d'un algorithme est une fonction de la taille n des données
2. Les principales fonctions vues au Lycée pour les problèmes de complexité en temps sont dans l'ordre croissant de complexité
   (a) $n \to \ln(n)$
   (b) $n \to n$
   (c) $n \to n\ln(n)$
   (d) $n \to n^2$
   (e) $n \to \mathrm{e}^n$
3. Nous verrons par exemple que pour résoudre le même problème P, un algorithme A aura une complexité en temps proportionnelle à $n\ln(n)$ et un autre B une complexité proportionnelle à n^2



Contrairement au graphique du début on voit une différence significative entre les deux courbes à tel point que l'une paraît constante ce qui n'est pas le cas



4. Il arrive que deux algorithmes différents ont la même complexité en temps mais qu'une fois traduit en programme l'un sera plus performant que l'autre (voir exercice)

2 Exercices

Ex 1

Question 1 La complexité en temps de la recherche d'un maximum dans un tableau d'entiers est :

☐ linéaire dans tous les cas
☐ quadratique dans tous les cas
☐ linéaire dans le pire des cas
☐ quadratique dans le pire des cas

Question 2 La complexité en temps de la recherche séquentielle d'un élément dans un tableau est :

☐ linéaire dans tous les cas
☐ quadratique dans tous les cas
☐ linéaire dans le pire des cas
☐ quadratique dans le pire des cas

Ex 2

Si sur une machine particulière il a fallu 1 s pour trouver un élément d'un tableau de taille 100 000 en dernière position, combien de temps à peu près mettra-t-on pour trouver un élément en dernière position dans un tableau de taille 1 000 000 sur cette même machine ?

Ex 3

Il s'agit de comparer deux programmes P_1 et P_2
P_1 est la traduction d'un algorithme de complexité linéaire et P_2 est la traduction d'un algorithme de complexité quadratique (dans tous les cas chacun)
P_1 est exécuté en à peu près 10^{-4} seconde sur un échantillon de taille 1000, alors que P_2 est exécuté en à peu près 10^{-3} seconde sur un échantillon de taille 1000
Donner une estimation du temps d'exécution des deux programmes lorsque la taille des données est un million

Ex 4

Modifier l'algorithme de recherche séquentielle afin d'avoir comme information tous les indices i tel que $T[i] = v$. Quelle est la complexité de cet algorithme dans tous les cas ?

Ex 5

Compléter la fonction `somme (liste)` qui prend en argument une liste non vide d'entiers et renvoie la somme de ses éléments.

Compléter aussi le programme principal

Algorithme 2 : Somme des entiers contenus dans une liste

```
    début
        /* Définition de la fonction somme(liste)                    */
 1      somme (T)
 2      début
            Données : Un tableau T non vide ayant n entiers
            Résultat : la somme des n entiers
 3          somme ← 0
 4          pour i ← 0 jusqu'à n − 1 faire
 5              .............................
 6              i ← i + 1
 7          fin
 8          retourner .................
 9      fin
        /* ----Programme Principal ------------                       */
10      liste ← [1,6,6,4]
11      .....................
    fin
```

1. Quel est le nombre d'additions ? En déduire la complexité
2. Ecrire une fonction `moyenne (liste)` qui prend en argument une liste non vide d'entiers et renvoie la moyenne de ses éléments. On pourra utiliser la fonction `somme(liste)` ci-dessus.

Ex 6

On définit en mathématiques la moyenne pondérée de n nombres x_1, x_2,...,x_n par des coefficients c_1, c_2, ..., c_n par :

$$\frac{c_1 \times x_1 + c_2 \times x_2 + ... + c_n \times x_n}{c_1 + c_2 + ... + c_n}$$

1. Ecrire une fonction `moyennePonderee(valeurs,coefficients)` qui prend en argument, une liste non vide de nombres `valeurs` et une liste non vide d'entiers `coefficients` et qui renvoie la moyenne pondérée .
2. Implémenter cette fonction en Python
3. Calculer la complexité en temps de cette fonction en prenant uniquement en compte les multiplications

Ex 7

On a une liste L de nombres décimaux, les températures maximales relevées à la station météo de Vélizy Villacoublay pour chaque jour du mois de Juin 2021

1. Ecrire une fonction en pseudo-code retournant le jour de la température maximale sur le mois de Juin 2021. Bien préciser les spécifications de la fonction
2. Ecrire une deuxième fonction retournant cette fois ci le jour de la première valeur de la température maximale sur le mois de Juin 2021
3. Quelle est la complexité dans tous les cas de la recherche d'un maximum dans une liste non triée ?

Ex 8

Si on sait qu'une liste est **déjà** triée dans l'ordre **croissant** que suffit-il de faire pour avoir le maximum de la liste ? Quelle est alors la complexité ?

Ex 9

Etant donné un tableau 2D carré d'entiers naturels

1. Définir en pseudo-code une fonction `nb_entiers_pairs1(tab2D)` qui parcourt le tableau ligne par ligne et renvoie le nombre d'entiers pairs contenus dans le tableau
2. Définir en pseudo-code une fonction `nb_entiers_pairs2(tab2D)` qui parcourt le tableau colonne par colonne et renvoie le nombre d'entiers pairs contenus dans le tableau
3. Evaluer la complexité en temps de chaque fonction
4. Implémenter chaque fonction en Python et comparer leur temps d'exécution sur des tableau de tailles différentes

Ex 10

Une liste contient des entiers non distincts mais triés dans l'ordre croissant par exemple [1,1,2,2,2,3]
De cette liste on veut créer une nouvelle liste ne contenant qu'un exemplaire de la précédente liste mais avec un seul exemplaire des entiers ici [1,2,3]
Donner un algorithme et évaluer sa complexité