



UNIVERSITÉ
CÔTE D'AZUR

Programmation impérative en Python

Cours 9. Animations et algorithmes

Olivier Baldellon

Courriel : `prenom.nom@univ-cotedazur.fr`

Page professionnelle : `https://upinfo.univ-cotedazur.fr/~obaldellon/`

LICENCE I — FACULTÉ DES SCIENCES ET INGÉNIERIE DE NICE — UNIVERSITÉ CÔTE D'AZUR

 Partie I. Programmation graphique

 Partie II. Algorithmes

 Partie III. Algorithmes de tri

 Partie IV. Complexité

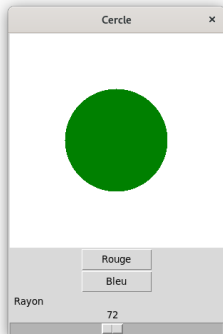
 Partie V. Algorithmes de recherche

 Partie VI. Crible d'Ératosthène

 Partie VII. Bilan

 Partie VIII. Table des matières

- ▶ On souhaite faire une petite animation qui affiche un cercle



avec

- ▶ un bouton pour changer la couleur en rouge
 - ▶ un autre pour la changer en bleu
 - ▶ un curseur pour choisir le taille
-
- ▶ Comment programmer une telle interaction avec l'utilisateur ?

- ▶ On ne va pas chercher à transformer une image en une autre.

- ▶ On ne va pas chercher à transformer une image en une autre.
- ▶ On va introduire une variable pour définir l'état du système
 - ▶ l'état correspond aux paramètres de l'image
 - ▶ à partir de l'état, on peut dessiner l'image correspondante

- ▶ On ne va pas chercher à transformer une image en une autre.
- ▶ On va introduire une variable pour définir l'état du système
 - ▶ l'état correspond aux paramètres de l'image
 - ▶ à partir de l'état, on peut dessiner l'image correspondante
- ▶ Lorsque l'utilisateur déclenche un évènement (ex : clic sur bouton)
 - ▶ on modifie l'état
 - ▶ on efface l'ancienne image
 - ▶ on retrace l'image correspondante au nouvel état.

- ▶ Ici, l'image ne dépend que de deux **paramètres** :
 - ▶ La couleur du disque
 - ▶ La taille du disque
- ▶ Il y a trois types d'**évènements** possibles
 - ▶ clic sur le bouton « rouge »
 - ▶ clic sur le bouton « bleu »
 - ▶ déplacement du curseur

- ▶ Ici, l'image ne dépend que de deux **paramètres** :
 - ▶ La couleur du disque
 - ▶ La taille du disque
- ▶ Il y a trois types d'**évènements** possibles
 - ▶ clic sur le bouton « rouge »
 - ▶ clic sur le bouton « bleu »
 - ▶ déplacement du curseur
- ▶ Au début on initialise l'état

Évènements :

États : E_0

Affichage :

- ▶ Ici, l'image ne dépend que de deux **paramètres** :
 - ▶ La couleur du disque
 - ▶ La taille du disque
- ▶ Il y a trois types d'**évènements** possibles
 - ▶ clic sur le bouton « rouge »
 - ▶ clic sur le bouton « bleu »
 - ▶ déplacement du curseur
- ▶ Au début on initialise l'état et on l'affiche

Évènements :

États :

E_0



Affichage :



- ▶ Ici, l'image ne dépend que de deux **paramètres** :
 - ▶ La couleur du disque
 - ▶ La taille du disque
- ▶ Il y a trois types d'**évènements** possibles
 - ▶ clic sur le bouton « rouge »
 - ▶ clic sur le bouton « bleu »
 - ▶ déplacement du curseur
- ▶ Au début on initialise l'état et on l'affiche
 - ▶ L'utilisateur change le rayon sur le curseur

Évènements :

États :

E_0



Affichage :



- ▶ Ici, l'image ne dépend que de deux **paramètres** :
 - ▶ La couleur du disque
 - ▶ La taille du disque
- ▶ Il y a trois types d'**évènements** possibles
 - ▶ clic sur le bouton « rouge »
 - ▶ clic sur le bouton « bleu »
 - ▶ déplacement du curseur
- ▶ Au début on initialise l'état et on l'affiche
 - ▶ L'utilisateur change le rayon sur le curseur : l'état change

Évènements :

États :

E_0 → E_1

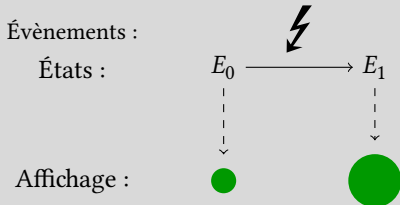


⋮

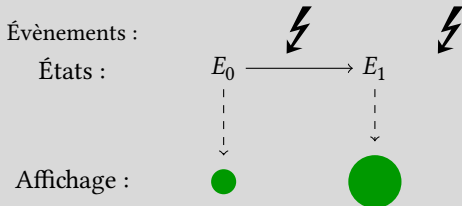
Affichage :



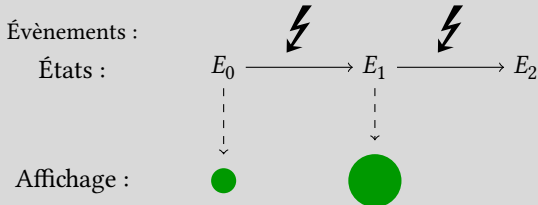
- ▶ Ici, l'image ne dépend que de deux **paramètres** :
 - ▶ La couleur du disque
 - ▶ La taille du disque
- ▶ Il y a trois types d'**évènements** possibles
 - ▶ clic sur le bouton « rouge »
 - ▶ clic sur le bouton « bleu »
 - ▶ déplacement du curseur
- ▶ Au début on initialise l'état et on l'affiche
 - ▶ L'utilisateur change le rayon sur le curseur : l'état change et on l'affiche



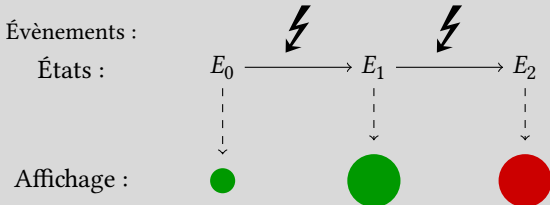
- ▶ Ici, l'image ne dépend que de deux **paramètres** :
 - ▶ La couleur du disque
 - ▶ La taille du disque
- ▶ Il y a trois types d'**évènements** possibles
 - ▶ clic sur le bouton « rouge »
 - ▶ clic sur le bouton « bleu »
 - ▶ déplacement du curseur
- ▶ Au début on initialise l'état et on l'affiche
 - ▶ L'utilisateur change le rayon sur le curseur : l'état change et on l'affiche
 - ▶ L'utilisateur clique sur le bouton « rouge »



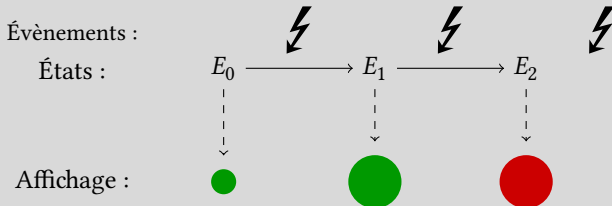
- ▶ Ici, l'image ne dépend que de deux **paramètres** :
 - ▶ La couleur du disque
 - ▶ La taille du disque
- ▶ Il y a trois types d'**évènements** possibles
 - ▶ clic sur le bouton « rouge »
 - ▶ clic sur le bouton « bleu »
 - ▶ déplacement du curseur
- ▶ Au début on initialise l'état et on l'affiche
 - ▶ L'utilisateur change le rayon sur le curseur : l'état change et on l'affiche
 - ▶ L'utilisateur clique sur le bouton « rouge » : l'état change



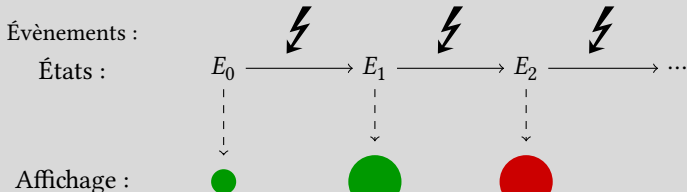
- ▶ Ici, l'image ne dépend que de deux **paramètres** :
 - ▶ La couleur du disque
 - ▶ La taille du disque
- ▶ Il y a trois types d'**évènements** possibles
 - ▶ clic sur le bouton « rouge »
 - ▶ clic sur le bouton « bleu »
 - ▶ déplacement du curseur
- ▶ Au début on initialise l'état et on l'affiche
 - ▶ L'utilisateur change le rayon sur le curseur : l'état change et on l'affiche
 - ▶ L'utilisateur clique sur le bouton « rouge » : l'état change et on l'affiche



- ▶ Ici, l'image ne dépend que de deux **paramètres** :
 - ▶ La couleur du disque
 - ▶ La taille du disque
- ▶ Il y a trois types d'**évènements** possibles
 - ▶ clic sur le bouton « rouge »
 - ▶ clic sur le bouton « bleu »
 - ▶ déplacement du curseur
- ▶ Au début on initialise l'état et on l'affiche
 - ▶ L'utilisateur change le rayon sur le curseur : l'état change et on l'affiche
 - ▶ L'utilisateur clique sur le bouton « rouge » : l'état change et on l'affiche
 - ▶ etc.



- ▶ Ici, l'image ne dépend que de deux **paramètres** :
 - ▶ La couleur du disque
 - ▶ La taille du disque
- ▶ Il y a trois types d'**évènements** possibles
 - ▶ clic sur le bouton « rouge »
 - ▶ clic sur le bouton « bleu »
 - ▶ déplacement du curseur
- ▶ Au début on initialise l'état et on l'affiche
 - ▶ L'utilisateur change le rayon sur le curseur : l'état change et on l'affiche
 - ▶ L'utilisateur clique sur le bouton « rouge » : l'état change et on l'affiche
 - ▶ etc.



- ▶ On stocke l'état dans un objet
 - ▶ Avec deux attributs (couleur et rayon) et une méthode (**affichage**)

SCRIPT

```
class État():  
  
    def __init__(self):  
        self.couleur='green'  
        self.rayon=Largeur/4  
        self.affichage() # On fait le premier affichage  
  
    def affichage(self):  
        Dessin.delete('all') # On efface tout  
        (x0,y0)=(Largeur//2,Hauteur//2)  
        disque(x0,y0,self.rayon,self.couleur)  
  
état=État() # On définit l'état de manière globale
```

- ▶ On stocke l'état dans un objet
 - ▶ Avec deux attributs (couleur et rayon) et une méthode (**affichage**)

SCRIPT

```
class État():  
  
    def __init__(self):  
        self.couleur='green'  
        self.rayon=Largeur/4  
        self.affichage() # On fait le premier affichage  
  
    def affichage(self):  
        Dessin.delete('all') # On efface tout  
        (x0,y0)=(Largeur//2,Hauteur//2)  
        disque(x0,y0,self.rayon,self.couleur)  
  
état=État() # On définit l'état de manière globale
```

- ▶ Remarquez que l'on fait appel à la méthode `affichage` dès l'initialisation

- ▶ On stocke l'état dans un objet
 - ▶ Avec deux attributs (couleur et rayon) et une méthode (**affichage**)

SCRIPT

```
class État():  
  
    def __init__(self):  
        self.couleur='green'  
        self.rayon=Largeur/4  
        self.affichage() # On fait le premier affichage  
  
    def affichage(self):  
        Dessin.delete('all') # On efface tout  
        (x0,y0)=(Largeur//2,Hauteur//2)  
        disque(x0,y0,self.rayon,self.couleur)  
  
état=État() # On définit l'état de manière globale
```

- ▶ Remarquez que l'on fait appel à la méthode `affichage` dès l'initialisation
- ▶ À chaque fois que l'on fait l'affichage :
 - ▶ on efface les objets précédents pour ne pas saturer la mémoire

- ▶ On crée maintenant des boutons et un curseur
- ▶ Ainsi que les fonctions correspondantes aux évènements associés

SCRIPT

```
def rouge(): # Appelée lorsqu'on clique sur Bouton 1
    état.couleur='red'
    état.affichage()

def bleu(): # Appelée lorsqu'on clique sur Bouton 2
    état.couleur='blue'
    état.affichage()

def change_taille(x): # Appelée lorsqu'on bouge le curseur
    état.rayon=int(x)
    état.affichage()

bouton1 = tk.Button(root,text="Rouge",command=rouge,width=9)
bouton2 = tk.Button(root,text="Bleu",command=bleu,width=9)
curseur = tk.Scale(root, orient="horizontal", length=Largeur,
                  label='Rayon',command=change_taille,
                  from_=1, to=Hauteur//2)

curseur.set(état.rayon) # Placement initial du curseur
```

SCRIPT

```
import tkinter as tk
Hauteur,Largeur = 800,800
root = tk.Tk()
root.title("Cercle")
Dessin = tk.Canvas(root,height=Hauteur,width=Largeur,bg='white')
Dessin.pack()

def disque(x,y,r,c): # (cf cours 4)

class État(): # Définit l'état et la méthode affichage
    état=État() # On initialise l'état

def rouge():          # Les actions associées
def bleu():           # aux évènements
def change_taille(x):#

bouton1 = # Les éléments de l'interface
bouton2 = # graphiques qui déclencheront
curseur = # les évènements







bouton1.pack() # On les positionne
bouton2.pack() # sur la fenêtre
curseur.pack()

root.mainloop() # À mettre à la fin de chaque programme Tk
```

► Souris

<Button-1>	Clic gauche	<Motion>	La souris bouge
<Button-2>	Clic central	<Button>	Clic sur un bouton
<Button-3>	Clic droit	<ButtonRelease>	Fin d'un clic

► Clavier



<Up>	touche 	<Key-a>	touche  (minuscule)
<Down>	touche 	<Key-B>	touche  (majuscule)
<Left>	touche 	<KeyPress>	on appuie sur une touche
<Right>	touche 	<KeyRelease>	on relâche une touche

► Pour associer l'évènement à une commande :

SCRIPT

```
def ça_bouge(event): # l'argument est obligatoire
    # Coordonnée et bouton de la souris (Bouton 1, 2 ou 3)
    print(event.x, event.y, event.num)
    # Symbole clavier (a, B, Enter, space)
    print(event.keysym)

root.bind('<Motion>', ça_bouge)
```

- ▶ On ajoute trois nouveaux évènements
 - ▶  ou  : le rayon augmente ou diminue
 - ▶ Mouvement souris : si on est sur le disque il devient rouge (bleu sinon)

SCRIPT

```
def sur_cercle(x1,y1):  
    (x0,y0)=(Largeur/2,Hauteur/2)  
    return (x0-x1)**2 + (y0-y1)**2 < état.rayon**2  
  
def ça_bouge(event):  
    if sur_cercle(event.x,event.y): état.couleur='red'  
    else: état.couleur='blue'  
    état.affichage()  
  
def plus_grand(event):  
    état.rayon = état.rayon+1  
    état.affichage()  
  
def plus_petit(event):  
    état.rayon = état.rayon-1  
    état.affichage()  
  
root.bind('<Motion>', ça_bouge)  
root.bind('<Down>', plus_petit)  
root.bind('<Up>', plus_grand)
```


- ▶ Pour l'instant, l'état évoluait suite à une action de l'utilisateur
 - ▶ Les évènements appelaient des fonctions qui modifiaient l'état.

- ▶ Pour l'instant, l'état évoluait suite à une action de l'utilisateur
 - ▶ Les évènements appelaient des fonctions qui modifiaient l'état.
- ▶ On veut maintenant que l'état puisse évoluer au cours du temps.
 - ▶ On va créer une nouvelle fonction
 - ▶ On va appeler cette fonction à intervalle régulier


- ▶ Pour l'instant, l'état évoluait suite à une action de l'utilisateur
 - ▶ Les évènements appelaient des fonctions qui modifiaient l'état.
- ▶ On veut maintenant que l'état puisse évoluer au cours du temps.
 - ▶ On va créer une nouvelle fonction
 - ▶ On va appeler cette fonction à intervalle régulier



Horloge :

États : E_0



Affichage : 

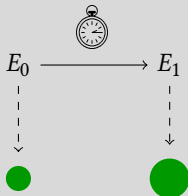
- ▶ Pour l'instant, l'état évoluait suite à une action de l'utilisateur
 - ▶ Les évènements appelaient des fonctions qui modifiaient l'état.
- ▶ On veut maintenant que l'état puisse évoluer au cours du temps.
 - ▶ On va créer une nouvelle fonction
 - ▶ On va appeler cette fonction à intervalle régulier

- ▶  Tic

Horloge :

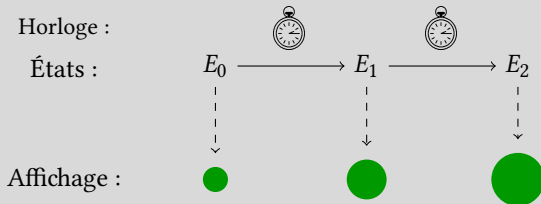
États :

Affichage :



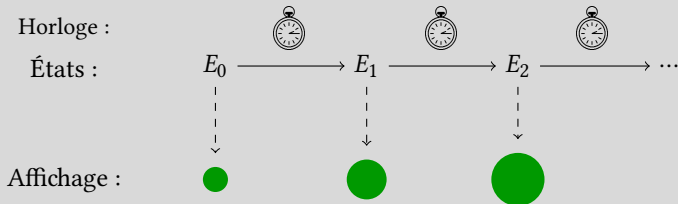
- ▶ Pour l'instant, l'état évoluait suite à une action de l'utilisateur
 - ▶ Les évènements appelaient des fonctions qui modifiaient l'état.
- ▶ On veut maintenant que l'état puisse évoluer au cours du temps.
 - ▶ On va créer une nouvelle fonction
 - ▶ On va appeler cette fonction à intervalle régulier

- ▶  Tic Tac



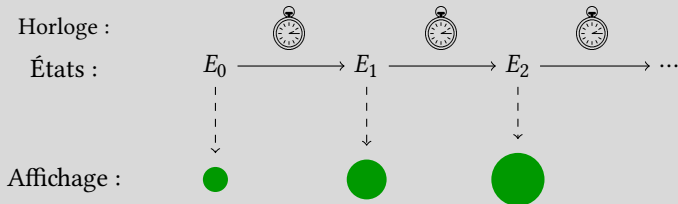
- ▶ Pour l'instant, l'état évoluait suite à une action de l'utilisateur
 - ▶ Les évènements appelaient des fonctions qui modifiaient l'état.
- ▶ On veut maintenant que l'état puisse évoluer au cours du temps.
 - ▶ On va créer une nouvelle fonction
 - ▶ On va appeler cette fonction à intervalle régulier

- ▶  Tic Tac **Tic**



- ▶ Pour l'instant, l'état évoluait suite à une action de l'utilisateur
 - ▶ Les évènements appelaient des fonctions qui modifiaient l'état.
- ▶ On veut maintenant que l'état puisse évoluer au cours du temps.
 - ▶ On va créer une nouvelle fonction
 - ▶ On va appeler cette fonction à intervalle régulier

- ▶  Tic Tac Tic



- ▶ Le disque donne l'impression de grossir au cours du temps
 - ▶ 24 images par seconde donnent l'illusion d'un mouvement continu

- ▶ On va rajouter un attribut `self.temps` à notre état.
- ▶ Il suffit que notre affichage dépende de `état.temps`.

```
def tictac():  
    état.temps = état.temps+1  
    état.affichage()  
    Dessin.after(10,tictac)
```

SCRIPT

- ▶ Pour utiliser `tictac`
 - ▶ Il faut le lancer une première fois

- ▶ On va rajouter un attribut `self.temps` à notre état.
- ▶ Il suffit que notre affichage dépende de `état.temps`.

```
def tictac():  
    état.temps = état.temps+1  
    état.affichage()  
    Dessin.after(10,tictac)
```

SCRIPT

- ▶ Pour utiliser `tictac`
 - ▶ Il faut le lancer une première fois
 - ▶ À chaque étape la fonction incrémente l'attribut `temps` de `état` ...

- ▶ On va rajouter un attribut `self.temps` à notre état.
- ▶ Il suffit que notre affichage dépende de `état.temps`.

```
def tictac():  
    état.temps = état.temps+1  
    état.affichage()  
    Dessin.after(10,tictac)
```

SCRIPT

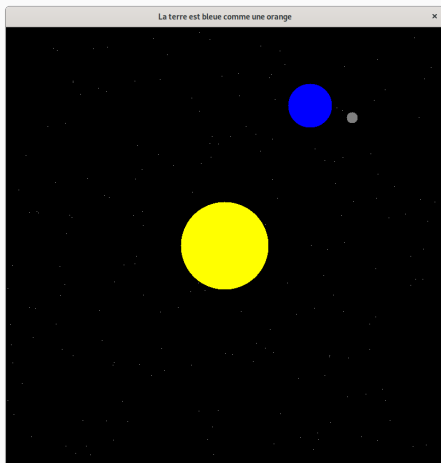
- ▶ Pour utiliser `tictac`
 - ▶ Il faut le lancer une première fois
 - ▶ À chaque étape la fonction incrémente l'attribut `temps` de `état` ...
 - ▶ ... et relance l'affichage()

- ▶ On va rajouter un attribut `self.temps` à notre état.
- ▶ Il suffit que notre affichage dépende de `état.temps`.

```
def tictac():  
    état.temps = état.temps+1  
    état.affichage()  
    Dessin.after(10,tictac)
```

SCRIPT

- ▶ Pour utiliser `tictac`
 - ▶ Il faut le lancer une première fois
 - ▶ À chaque étape la fonction incrémente l'attribut `temps` de `état` ...
 - ▶ ... et relance l'affichage()
 - ▶ Enfin, la fonction demande a être rappelée dans 10 ms.



On cherche à représenter le Soleil,
la Terre et la Lune en rotation

- ▶ Le tout sur un ciel étoilé
- ▶ on rappelle la formule d'un point sur un cercle (cours 4)

$$\begin{cases} x(t) = x_0 + R \cos(t), \\ y(t) = y_0 + R \sin(t). \end{cases}$$

```
def rotation(x,y,r,w,t)
  # position d'un point à l'instant t, tournant sur
  # un cercle de centre (x,y) et de rayon r avec une
  # vitesse angulaire w
  return (x + r*cos(-t*w) , y + r*sin(-t*w))
```

SCRIPT

SCRIPT

```
class État():
    def __init__(self): # Deux attributs : le temps et une liste d'étoiles
        (L,H) = (Largeur,Hauteur)
        self.étoiles = [(randint(0,L),randint(0,H)) for e in range(200)]
        self.temps=0
        self.affichage()

    def affichage(self):
        Dessin.delete('all') # On efface tout
        for (x,y) in self.étoiles: # Les étoiles
            Dessin.create_rectangle((x-1,y-1),(x+1,y+1),fill='white')

        (x0,y0) = (Largeur//2,Hauteur//2) # Le Soleil
        (x1,y1) = rotation(x0,y0,300,1/100,self.temps) # La Terre
        (x2,y2) = rotation(x1,y1, 80,12/100,self.temps) # La Lune
        disque(x0,y0,80,'yellow')
        disque(x1,y1,40,'blue')
        disque(x2,y2,10,'gray')

def tictac(): # Toutes les 20 ms, on change l'état et on l'affiche
    état.temps = état.temps+1
    état.affichage()
    Dessin.after(20,tictac)

état=État()
tictac() # On lance l'horloge
root.mainloop() # À mettre à la fin de chaque programme Tk
```

- ▶ Ce cours n'a pas pour objectif de faire de vous des experts en Tk.
 - ▶ le but était de vous montrer les possibilités,
 - ▶ de vous expliquer les grands principes (programmation événementielle)
 - ▶ et de vous permettre de continuer seul.

- ▶ Ce cours n'a pas pour objectif de faire de vous des experts en Tk.
 - ▶ le but était de vous montrer les possibilités,
 - ▶ de vous expliquer les grands principes (programmation événementielle)
 - ▶ et de vous permettre de continuer seul.
 - ▶ Écrire des programmes est un excellent moyen de progresser
 - ▶ C'est en codant qu'on apprend à programmer
 - ▶ Les jeux et animations sont un bon prétexte pour se motiver
 - ▶ Pour les curieux, il y a le site très complet en français :
- <http://pascal.ortiz.free.fr/contents/tkinter/tkinter/>
- ▶ Tk est très pratique pour faire des interfaces graphiques complètes
 - ▶ Thonny est fait en Tk.
 - ▶ Si vous êtes intéressés par les jeux, vous pouvez aussi regarder pygame.

 Partie I. Programmation graphique

 Partie II. Algorithmes

 Partie III. Algorithmes de tri

 Partie IV. Complexité

 Partie V. Algorithmes de recherche

 Partie VI. Crible d'Ératosthène

 Partie VII. Bilan

 Partie VIII. Table des matières

- ▶ On souhaite confier à l'ordinateur la **résolution de tâches complexes**

- ▶ On souhaite confier à l'ordinateur la **résolution de tâches complexes**
 - ▶ Recherche de plus courts chemins,

- ▶ On souhaite confier à l'ordinateur la **résolution de tâches complexes**
 - ▶ Recherche de plus courts chemins,
 - ▶ Recherche d'un mot dans le dictionnaire,

- ▶ On souhaite confier à l'ordinateur la **résolution de tâches complexes**
 - ▶ Recherche de plus courts chemins,
 - ▶ Recherche d'un mot dans le dictionnaire,
 - ▶ Affichage d'une figure, etc.

- ▶ On souhaite confier à l'ordinateur la **résolution de tâches complexes**
 - ▶ Recherche de plus courts chemins,
 - ▶ Recherche d'un mot dans le dictionnaire,
 - ▶ Affichage d'une figure, etc.

- ▶ Comment résoudre ce type de tâches ?

- ▶ On souhaite confier à l'ordinateur la **résolution de tâches complexes**
 - ▶ Recherche de plus courts chemins,
 - ▶ Recherche d'un mot dans le dictionnaire,
 - ▶ Affichage d'une figure, etc.
- ▶ Comment résoudre ce type de tâches ?
 - ▶ On peut savoir résoudre spontanément des cas particuliers à la main

- ▶ On souhaite confier à l'ordinateur la **résolution de tâches complexes**
 - ▶ Recherche de plus courts chemins,
 - ▶ Recherche d'un mot dans le dictionnaire,
 - ▶ Affichage d'une figure, etc.
- ▶ Comment résoudre ce type de tâches ?
 - ▶ On peut savoir résoudre spontanément des cas particuliers à la main
 - ▶ Mais on cherche des **méthodes générales**. On parle alors d'**algorithme**

- ▶ On souhaite confier à l'ordinateur la **résolution de tâches complexes**
 - ▶ Recherche de plus courts chemins,
 - ▶ Recherche d'un mot dans le dictionnaire,
 - ▶ Affichage d'une figure, etc.
- ▶ Comment résoudre ce type de tâches ?
 - ▶ On peut savoir résoudre spontanément des cas particuliers à la main
 - ▶ Mais on cherche des **méthodes générales**. On parle alors d'**algorithme**
 - ▶ De manière générale, réfléchir avant de coder, est souvent utile.

- ▶ On souhaite confier à l'ordinateur la **résolution de tâches complexes**
 - ▶ Recherche de plus courts chemins,
 - ▶ Recherche d'un mot dans le dictionnaire,
 - ▶ Affichage d'une figure, etc.

- ▶ Comment résoudre ce type de tâches ?
 - ▶ On peut savoir résoudre spontanément des cas particuliers à la main
 - ▶ Mais on cherche des **méthodes générales**. On parle alors d'**algorithme**
 - ▶ De manière générale, réfléchir avant de coder, est souvent utile.
 - ▶ De manière générale, réfléchir est souvent utile.

- ▶ On souhaite confier à l'ordinateur la **résolution de tâches complexes**
 - ▶ Recherche de plus courts chemins,
 - ▶ Recherche d'un mot dans le dictionnaire,
 - ▶ Affichage d'une figure, etc.
- ▶ Comment résoudre ce type de tâches ?
 - ▶ On peut savoir résoudre spontanément des cas particuliers à la main
 - ▶ Mais on cherche des **méthodes générales**. On parle alors d'**algorithme**
 - ▶ De manière générale, réfléchir avant de coder, est souvent utile.
 - ▶ De manière générale, réfléchir est souvent utile.
- ▶ Un algorithme est indépendant du langage dans lequel on code :

- ▶ On souhaite confier à l'ordinateur la **résolution de tâches complexes**
 - ▶ Recherche de plus courts chemins,
 - ▶ Recherche d'un mot dans le dictionnaire,
 - ▶ Affichage d'une figure, etc.
- ▶ Comment résoudre ce type de tâches ?
 - ▶ On peut savoir résoudre spontanément des cas particuliers à la main
 - ▶ Mais on cherche des **méthodes générales**. On parle alors d'**algorithme**
 - ▶ De manière générale, réfléchir avant de coder, est souvent utile.
 - ▶ De manière générale, réfléchir est souvent utile.
- ▶ Un algorithme est indépendant du langage dans lequel on code :
 - ▶ Il ne sera donc pas forcément écrit en Python.

- ▶ On souhaite confier à l'ordinateur la **résolution de tâches complexes**
 - ▶ Recherche de plus courts chemins,
 - ▶ Recherche d'un mot dans le dictionnaire,
 - ▶ Affichage d'une figure, etc.
- ▶ Comment résoudre ce type de tâches ?
 - ▶ On peut savoir résoudre spontanément des cas particuliers à la main
 - ▶ Mais on cherche des **méthodes générales**. On parle alors d'**algorithme**
 - ▶ De manière générale, réfléchir avant de coder, est souvent utile.
 - ▶ De manière générale, réfléchir est souvent utile.
- ▶ Un algorithme est indépendant du langage dans lequel on code :
 - ▶ Il ne sera donc pas forcément écrit en Python.
 - ▶ Il pourra même être appliqué à la main.

- ▶ On souhaite confier à l'ordinateur la **résolution de tâches complexes**
 - ▶ Recherche de plus courts chemins,
 - ▶ Recherche d'un mot dans le dictionnaire,
 - ▶ Affichage d'une figure, etc.
- ▶ Comment résoudre ce type de tâches ?
 - ▶ On peut savoir résoudre spontanément des cas particuliers à la main
 - ▶ Mais on cherche des **méthodes générales**. On parle alors d'**algorithme**
 - ▶ De manière générale, réfléchir avant de coder, est souvent utile.
 - ▶ De manière générale, réfléchir est souvent utile.
- ▶ Un algorithme est indépendant du langage dans lequel on code :
 - ▶ Il ne sera donc pas forcément écrit en Python.
 - ▶ Il pourra même être appliqué à la main.
- ▶ Il doit être constitué :

- ▶ On souhaite confier à l'ordinateur la **résolution de tâches complexes**
 - ▶ Recherche de plus courts chemins,
 - ▶ Recherche d'un mot dans le dictionnaire,
 - ▶ Affichage d'une figure, etc.
- ▶ Comment résoudre ce type de tâches ?
 - ▶ On peut savoir résoudre spontanément des cas particuliers à la main
 - ▶ Mais on cherche des **méthodes générales**. On parle alors d'**algorithme**
 - ▶ De manière générale, réfléchir avant de coder, est souvent utile.
 - ▶ De manière générale, réfléchir est souvent utile.
- ▶ Un algorithme est indépendant du langage dans lequel on code :
 - ▶ Il ne sera donc pas forcément écrit en Python.
 - ▶ Il pourra même être appliqué à la main.
- ▶ Il doit être constitué :
 - ▶ d'opérations élémentaires non ambiguës à suivre pas à pas ;

- ▶ On souhaite confier à l'ordinateur la **résolution de tâches complexes**
 - ▶ Recherche de plus courts chemins,
 - ▶ Recherche d'un mot dans le dictionnaire,
 - ▶ Affichage d'une figure, etc.
- ▶ Comment résoudre ce type de tâches ?
 - ▶ On peut savoir résoudre spontanément des cas particuliers à la main
 - ▶ Mais on cherche des **méthodes générales**. On parle alors d'**algorithme**
 - ▶ De manière générale, réfléchir avant de coder, est souvent utile.
 - ▶ De manière générale, réfléchir est souvent utile.
- ▶ Un algorithme est indépendant du langage dans lequel on code :
 - ▶ Il ne sera donc pas forcément écrit en Python.
 - ▶ Il pourra même être appliqué à la main.
- ▶ Il doit être constitué :
 - ▶ d'opérations élémentaires non ambiguës à suivre pas à pas ;
 - ▶ de tests et de prises de décisions.

- ▶ En mathématiques, on cherche :

- ▶ En mathématiques, on cherche :
 - ▶ À connaître des vérités générales : les théorèmes

- ▶ En mathématiques, on cherche :
 - ▶ À connaître des vérités générales : les théorèmes
 - ▶ À trouver des méthodes de calculs, constructions : les algorithmes

- ▶ En mathématiques, on cherche :
 - ▶ À connaître des vérités générales : les théorèmes
 - ▶ À trouver des méthodes de calculs, constructions : les algorithmes
- ▶ Dès l'Antiquité Euclide faisait déjà la distinction entre :
 - ▶ Ce qu'il fallait démontrer
 - ▶ Ce qu'il fallait construire (constructions à la règle et au compas)

- ▶ En mathématiques, on cherche :
 - ▶ À connaître des vérités générales : les théorèmes
 - ▶ À trouver des méthodes de calculs, constructions : les algorithmes
- ▶ Dès l'Antiquité Euclide faisait déjà la distinction entre :
 - ▶ Ce qu'il fallait démontrer
 - ▶ Ce qu'il fallait construire (constructions à la règle et au compas)
- ▶ Le mot « algorithme » dérive du nom de Al-Khwarizmi.
 - ▶ mathématicien perse du IX^e siècle.
 - ▶ Son *kitab al-mukhtasar fi hisab al-jabr wa-l-muqabala* (abrégé de calcul par réduction et comparaison) a donné le mot algèbre

- ▶ En mathématiques, on cherche :
 - ▶ À connaître des vérités générales : les théorèmes
 - ▶ À trouver des méthodes de calculs, constructions : les algorithmes
- ▶ Dès l'Antiquité Euclide faisait déjà la distinction entre :
 - ▶ Ce qu'il fallait démontrer
 - ▶ Ce qu'il fallait construire (constructions à la règle et au compas)
- ▶ Le mot « algorithme » dérive du nom de Al-Khwarizmi.
 - ▶ mathématicien perse du IX^e siècle.
 - ▶ Son *kitab al-mukhtasar fi hisab al-jabr wa-l-muqabala* (abrégé de calcul par réduction et comparaison) a donné le mot algèbre
- ▶ Depuis l'avènement des ordinateurs, nous avons les outils idéaux pour appliquer les algorithmes.

- ▶ La vitesse d'exécution d'un programme dépend du nombre d'opérations.

- ▶ La vitesse d'exécution d'un programme dépend du nombre d'opérations.
 - ▶ Nombre d'additions

- ▶ La vitesse d'exécution d'un programme dépend du nombre d'opérations.
 - ▶ Nombre d'additions
 - ▶ Nombre de comparaisons

- ▶ La vitesse d'exécution d'un programme dépend du nombre d'opérations.
 - ▶ Nombre d'additions
 - ▶ Nombre de comparaisons
 - ▶ Nombre d'écritures en mémoire

- ▶ La vitesse d'exécution d'un programme dépend du nombre d'opérations.
 - ▶ Nombre d'additions
 - ▶ Nombre de comparaisons
 - ▶ Nombre d'écritures en mémoire
- ▶ **Complexité** : nombre d'opérations nécessaires au calcul d'une tâche

- ▶ La vitesse d'exécution d'un programme dépend du nombre d'opérations.
 - ▶ Nombre d'additions
 - ▶ Nombre de comparaisons
 - ▶ Nombre d'écritures en mémoire
- ▶ **Complexité** : nombre d'opérations nécessaires au calcul d'une tâche
 - ▶ Elle est exprimée comme une fonction de la taille du problème n

- ▶ La vitesse d'exécution d'un programme dépend du nombre d'opérations.
 - ▶ Nombre d'additions
 - ▶ Nombre de comparaisons
 - ▶ Nombre d'écritures en mémoire
- ▶ **Complexité** : nombre d'opérations nécessaires au calcul d'une tâche
 - ▶ Elle est exprimée comme une fonction de la taille du problème n
 - ▶ n peut être la taille d'une liste

- ▶ La vitesse d'exécution d'un programme dépend du nombre d'opérations.
 - ▶ Nombre d'additions
 - ▶ Nombre de comparaisons
 - ▶ Nombre d'écritures en mémoire
- ▶ **Complexité** : nombre d'opérations nécessaires au calcul d'une tâche
 - ▶ Elle est exprimée comme une fonction de la taille du problème n
 - ▶ n peut être la taille d'une liste
 - ▶ n peut être la taille d'une image

- ▶ La vitesse d'exécution d'un programme dépend du nombre d'opérations.
 - ▶ Nombre d'additions
 - ▶ Nombre de comparaisons
 - ▶ Nombre d'écritures en mémoire
- ▶ **Complexité** : nombre d'opérations nécessaires au calcul d'une tâche
 - ▶ Elle est exprimée comme une fonction de la taille du problème n
 - ▶ n peut être la taille d'une liste
 - ▶ n peut être la taille d'une image
 - ▶ etc.

- ▶ La vitesse d'exécution d'un programme dépend du nombre d'opérations.
 - ▶ Nombre d'additions
 - ▶ Nombre de comparaisons
 - ▶ Nombre d'écritures en mémoire
- ▶ **Complexité** : nombre d'opérations nécessaires au calcul d'une tâche
 - ▶ Elle est exprimée comme une fonction de la taille du problème n
 - ▶ n peut être la taille d'une liste
 - ▶ n peut être la taille d'une image
 - ▶ etc.
- ▶ En pratique on ne s'intéresse pas à une formule exacte

- ▶ La vitesse d'exécution d'un programme dépend du nombre d'opérations.
 - ▶ Nombre d'additions
 - ▶ Nombre de comparaisons
 - ▶ Nombre d'écritures en mémoire
- ▶ **Complexité** : nombre d'opérations nécessaires au calcul d'une tâche
 - ▶ Elle est exprimée comme une fonction de la taille du problème n
 - ▶ n peut être la taille d'une liste
 - ▶ n peut être la taille d'une image
 - ▶ etc.
- ▶ En pratique on ne s'intéresse pas à une formule exacte
 - ▶ Si le nombre d'opérations est de la forme : $3 \cdot n^2 + 2 \cdot n + 125$

- ▶ La vitesse d'exécution d'un programme dépend du nombre d'opérations.
 - ▶ Nombre d'additions
 - ▶ Nombre de comparaisons
 - ▶ Nombre d'écritures en mémoire
- ▶ **Complexité** : nombre d'opérations nécessaires au calcul d'une tâche
 - ▶ Elle est exprimée comme une fonction de la taille du problème n
 - ▶ n peut être la taille d'une liste
 - ▶ n peut être la taille d'une image
 - ▶ etc.
- ▶ En pratique on ne s'intéresse pas à une formule exacte
 - ▶ Si le nombre d'opérations est de la forme : $3 \cdot n^2 + 2 \cdot n + 125$
 - ▶ on notera simplement $O(n^2)$: notation de Landau

- ▶ La vitesse d'exécution d'un programme dépend du nombre d'opérations.
 - ▶ Nombre d'additions
 - ▶ Nombre de comparaisons
 - ▶ Nombre d'écritures en mémoire
- ▶ **Complexité** : nombre d'opérations nécessaires au calcul d'une tâche
 - ▶ Elle est exprimée comme une fonction de la taille du problème n
 - ▶ n peut être la taille d'une liste
 - ▶ n peut être la taille d'une image
 - ▶ etc.
- ▶ En pratique on ne s'intéresse pas à une formule exacte
 - ▶ Si le nombre d'opérations est de la forme : $3 \cdot n^2 + 2 \cdot n + 125$
 - ▶ on notera simplement $O(n^2)$: notation de Landau
 - ▶ car ce qui nous intéresse est que ce soit un polynôme du second degré.

- ▶ La complexité d'un programme se note $O(f(n))$ « Grand Ô de f de n ».

- ▶ La complexité d'un programme se note $O(f(n))$ « Grand Ô de f de n ».
- ▶ On pose $C(n) = O(f(n))$ s'il existe un nombre $k > 0$ tel que :

$$C(n) \leq k \cdot f(n) \quad \text{pour } n \text{ suffisamment grand}$$

- ▶ La complexité d'un programme se note $O(f(n))$ « Grand Ô de f de n ».
- ▶ On pose $C(n) = O(f(n))$ s'il existe un nombre $k > 0$ tel que :

$$C(n) \leq k \cdot f(n) \quad \text{pour } n \text{ suffisamment grand}$$

- ▶ Par exemple, si C est un polynôme du second degré : $C(n) = 3n^2 - 5n + 4$

- ▶ La complexité d'un programme se note $O(f(n))$ « Grand Ô de f de n ».
- ▶ On pose $C(n) = O(f(n))$ s'il existe un nombre $k > 0$ tel que :

$$C(n) \leq k \cdot f(n) \quad \text{pour } n \text{ suffisamment grand}$$

- ▶ Par exemple, si C est un polynôme du second degré : $C(n) = 3n^2 - 5n + 4$
 - ▶ Lorsque n est grand $C(n) \approx 3n^2$ et des poussières

- ▶ La complexité d'un programme se note $O(f(n))$ « Grand Ô de f de n ».
- ▶ On pose $C(n) = O(f(n))$ s'il existe un nombre $k > 0$ tel que :

$$C(n) \leq k \cdot f(n) \quad \text{pour } n \text{ suffisamment grand}$$

- ▶ Par exemple, si C est un polynôme du second degré : $C(n) = 3n^2 - 5n + 4$
 - ▶ Lorsque n est grand $C(n) \approx 3n^2$ et des poussières $\leq 4n^2$

- ▶ La complexité d'un programme se note $O(f(n))$ « Grand Ô de f de n ».
- ▶ On pose $C(n) = O(f(n))$ s'il existe un nombre $k > 0$ tel que :

$$C(n) \leq k \cdot f(n) \quad \text{pour } n \text{ suffisamment grand}$$

- ▶ Par exemple, si C est un polynôme du second degré : $C(n) = 3n^2 - 5n + 4$
 - ▶ Lorsque n est grand $C(n) \approx 3n^2$ et des poussières $\leq 4n^2$
 - ▶ On a donc $C(n) = O(n^2)$

- ▶ La complexité d'un programme se note $O(f(n))$ « Grand Ô de f de n ».
- ▶ On pose $C(n) = O(f(n))$ s'il existe un nombre $k > 0$ tel que :

$$C(n) \leq k \cdot f(n) \quad \text{pour } n \text{ suffisamment grand}$$

- ▶ Par exemple, si C est un polynôme du second degré : $C(n) = 3n^2 - 5n + 4$
 - ▶ Lorsque n est grand $C(n) \approx 3n^2$ et des poussières $\leq 4n^2$
 - ▶ On a donc $C(n) = O(n^2)$
- ▶ Voici les complexités souvent rencontrées :

- ▶ La complexité d'un programme se note $O(f(n))$ « Grand Ô de f de n ».
- ▶ On pose $C(n) = O(f(n))$ s'il existe un nombre $k > 0$ tel que :

$$C(n) \leq k \cdot f(n) \quad \text{pour } n \text{ suffisamment grand}$$

- ▶ Par exemple, si C est un polynôme du second degré : $C(n) = 3n^2 - 5n + 4$
 - ▶ Lorsque n est grand $C(n) \approx 3n^2$ et des poussières $\leq 4n^2$
 - ▶ On a donc $C(n) = O(n^2)$
- ▶ Voici les complexités souvent rencontrées :
 - ▶ $O(2^n)$ pour une complexité exponentielle

- ▶ La complexité d'un programme se note $O(f(n))$ « Grand Ô de f de n ».
- ▶ On pose $C(n) = O(f(n))$ s'il existe un nombre $k > 0$ tel que :

$$C(n) \leq k \cdot f(n) \quad \text{pour } n \text{ suffisamment grand}$$

- ▶ Par exemple, si C est un polynôme du second degré : $C(n) = 3n^2 - 5n + 4$
 - ▶ Lorsque n est grand $C(n) \approx 3n^2$ et des poussières $\leq 4n^2$
 - ▶ On a donc $C(n) = O(n^2)$
- ▶ Voici les complexités souvent rencontrées :
 - ▶ $O(2^n)$ pour une complexité exponentielle
 - ▶ $O(n^p)$ pour une complexité polynomiale (un polynôme de degré p)

- ▶ La complexité d'un programme se note $O(f(n))$ « Grand Ô de f de n ».
- ▶ On pose $C(n) = O(f(n))$ s'il existe un nombre $k > 0$ tel que :

$$C(n) \leq k \cdot f(n) \quad \text{pour } n \text{ suffisamment grand}$$

- ▶ Par exemple, si C est un polynôme du second degré : $C(n) = 3n^2 - 5n + 4$
 - ▶ Lorsque n est grand $C(n) \approx 3n^2$ et des poussières $\leq 4n^2$
 - ▶ On a donc $C(n) = O(n^2)$
- ▶ Voici les complexités souvent rencontrées :
 - ▶ $O(2^n)$ pour une complexité exponentielle
 - ▶ $O(n^p)$ pour une complexité polynomiale (un polynôme de degré p)
 - ▶ $O(n^2)$ pour une complexité quadratique (qui est aussi polynomiale $p = 2$)

- ▶ La complexité d'un programme se note $O(f(n))$ « Grand Ô de f de n ».
- ▶ On pose $C(n) = O(f(n))$ s'il existe un nombre $k > 0$ tel que :

$$C(n) \leq k \cdot f(n) \quad \text{pour } n \text{ suffisamment grand}$$

- ▶ Par exemple, si C est un polynôme du second degré : $C(n) = 3n^2 - 5n + 4$
 - ▶ Lorsque n est grand $C(n) \approx 3n^2$ et des poussières $\leq 4n^2$
 - ▶ On a donc $C(n) = O(n^2)$
- ▶ Voici les complexités souvent rencontrées :
 - ▶ $O(2^n)$ pour une complexité exponentielle
 - ▶ $O(n^p)$ pour une complexité polynomiale (un polynôme de degré p)
 - ▶ $O(n^2)$ pour une complexité quadratique (qui est aussi polynomiale $p = 2$)
 - ▶ $O(n \log n)$

- ▶ La complexité d'un programme se note $O(f(n))$ « Grand Ô de f de n ».
- ▶ On pose $C(n) = O(f(n))$ s'il existe un nombre $k > 0$ tel que :

$$C(n) \leq k \cdot f(n) \quad \text{pour } n \text{ suffisamment grand}$$

- ▶ Par exemple, si C est un polynôme du second degré : $C(n) = 3n^2 - 5n + 4$
 - ▶ Lorsque n est grand $C(n) \approx 3n^2$ et des poussières $\leq 4n^2$
 - ▶ On a donc $C(n) = O(n^2)$
- ▶ Voici les complexités souvent rencontrées :
 - ▶ $O(2^n)$ pour une complexité exponentielle
 - ▶ $O(n^p)$ pour une complexité polynomiale (un polynôme de degré p)
 - ▶ $O(n^2)$ pour une complexité quadratique (qui est aussi polynomiale $p = 2$)
 - ▶ $O(n \log n)$
 - ▶ $O(n)$ pour une complexité linéaire

- ▶ La complexité d'un programme se note $O(f(n))$ « Grand Ô de f de n ».
- ▶ On pose $C(n) = O(f(n))$ s'il existe un nombre $k > 0$ tel que :

$$C(n) \leq k \cdot f(n) \quad \text{pour } n \text{ suffisamment grand}$$


- ▶ Par exemple, si C est un polynôme du second degré : $C(n) = 3n^2 - 5n + 4$
 - ▶ Lorsque n est grand $C(n) \approx 3n^2$ et des poussières $\leq 4n^2$
 - ▶ On a donc $C(n) = O(n^2)$
- ▶ Voici les complexités souvent rencontrées :
 - ▶ $O(2^n)$ pour une complexité exponentielle
 - ▶ $O(n^p)$ pour une complexité polynomiale (un polynôme de degré p)
 - ▶ $O(n^2)$ pour une complexité quadratique (qui est aussi polynomiale $p = 2$)
 - ▶ $O(n \log n)$
 - ▶ $O(n)$ pour une complexité linéaire
 - ▶ $O(\log n)$ pour une complexité logarithmique

- ▶ La complexité d'un programme se note $O(f(n))$ « Grand Ô de f de n ».
- ▶ On pose $C(n) = O(f(n))$ s'il existe un nombre $k > 0$ tel que :

$$C(n) \leq k \cdot f(n) \quad \text{pour } n \text{ suffisamment grand}$$

- ▶ Par exemple, si C est un polynôme du second degré : $C(n) = 3n^2 - 5n + 4$
 - ▶ Lorsque n est grand $C(n) \approx 3n^2$ et des poussières $\leq 4n^2$
 - ▶ On a donc $C(n) = O(n^2)$
- ▶ Voici les complexités souvent rencontrées :
 - ▶ $O(2^n)$ pour une complexité exponentielle
 - ▶ $O(n^p)$ pour une complexité polynomiale (un polynôme de degré p)
 - ▶ $O(n^2)$ pour une complexité quadratique (qui est aussi polynomiale $p = 2$)
 - ▶ $O(n \log n)$
 - ▶ $O(n)$ pour une complexité linéaire
 - ▶ $O(\log n)$ pour une complexité logarithmique
 - ▶ $O(1)$ pour une complexité constante

 Partie I. Programmation graphique

 Partie II. Algorithmes

 Partie III. Algorithmes de tri

 Partie IV. Complexité

 Partie V. Algorithmes de recherche

 Partie VI. Crible d'Ératosthène

 Partie VII. Bilan

 Partie VIII. Table des matières

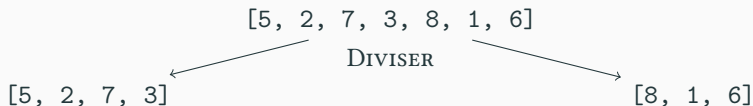
- ▶ Concevons un algorithme de tri efficace en $O(n \cdot \log n)$.

- ▶ Concevons un algorithme de tri efficace en $O(n \cdot \log n)$.
- ▶ On utilisera une méthode de type DIVISER POUR RÉGNER

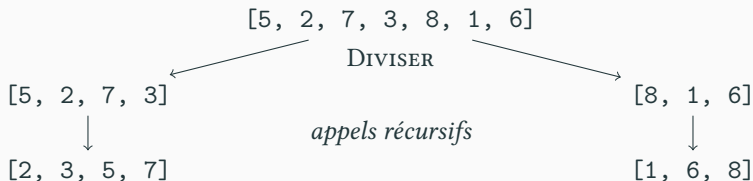
- ▶ Concevons un algorithme de tri efficace en $O(n \cdot \log n)$.
- ▶ On utilisera une méthode de type DIVISER POUR RÉGNER
- ▶ Pour trier une liste L :

[5, 2, 7, 3, 8, 1, 6]

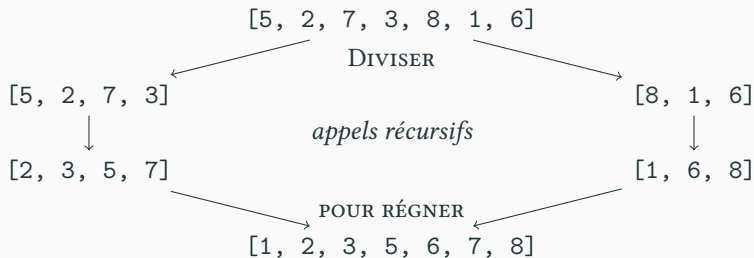
- ▶ Concevons un algorithme de tri efficace en $O(n \cdot \log n)$.
- ▶ On utilisera une méthode de type DIVISER POUR RÉGNER
- ▶ Pour trier une liste L :
 - ▶ on sépare L en deux listes de même taille L1 et L2;



- ▶ Concevons un algorithme de tri efficace en $O(n \cdot \log n)$.
- ▶ On utilisera une méthode de type DIVISER POUR RÉGNER
- ▶ Pour trier une liste L :
 - ▶ on sépare L en deux listes de même taille L1 et L2;
 - ▶ on trie ces deux listes récursivement;



- ▶ Concevons un algorithme de tri efficace en $O(n \cdot \log n)$.
- ▶ On utilisera une méthode de type DIVISER POUR RÉGNER
- ▶ Pour trier une liste L :
 - ▶ on sépare L en deux listes de même taille L1 et L2;
 - ▶ on trie ces deux listes récursivement;
 - ▶ on fusionne les deux listes.



SCRIPT

```
def tri_fusion(L):  
    if len(L) < 2:  
        return L  
    else:  
        moitié = len(L) // 2  
        L1 = tri_fusion(L[:moitié])  
        L2 = tri_fusion(L[moitié:])  
        return fusion(L1,L2)
```


SCRIPT

```
def tri_fusion(L):  
    if len(L) < 2:  
        return L  
    else:  
        moitié = len(L) // 2  
        L1 = tri_fusion(L[:moitié])  
        L2 = tri_fusion(L[moitié:])  
        return fusion(L1,L2)
```

- ▶ il reste à écrire l'opération de fusion

SCRIPT

```
def tri_fusion(L):  
    if len(L) < 2:  
        return L  
    else:  
        moitié = len(L) // 2  
        L1 = tri_fusion(L[:moitié])  
        L2 = tri_fusion(L[moitié:])  
        return fusion(L1,L2)
```

- ▶ il reste à écrire l'opération de fusion
 - ▶ On crée une liste vide R

SCRIPT

```
def tri_fusion(L):  
    if len(L) < 2:  
        return L  
    else:  
        moitié = len(L) // 2  
        L1 = tri_fusion(L[:moitié])  
        L2 = tri_fusion(L[moitié:])  
        return fusion(L1,L2)
```

- ▶ il reste à écrire l'opération de fusion
 - ▶ On crée une liste vide R
 - ▶ on regarde le premier élément de L1 et de L2.

SCRIPT

```
def tri_fusion(L):  
    if len(L) < 2:  
        return L  
    else:  
        moitié = len(L) // 2  
        L1 = tri_fusion(L[:moitié])  
        L2 = tri_fusion(L[moitié:])  
        return fusion(L1,L2)
```

- ▶ il reste à écrire l'opération de fusion
 - ▶ On crée une liste vide R
 - ▶ on regarde le premier élément de L1 et de L2.
 - ▶ on retire le plus petit des deux et on l'ajoute à R

SCRIPT

```
def tri_fusion(L):  
    if len(L) < 2:  
        return L  
    else:  
        moitié = len(L) // 2  
        L1 = tri_fusion(L[:moitié])  
        L2 = tri_fusion(L[moitié:])  
        return fusion(L1,L2)
```

- ▶ il reste à écrire l'opération de fusion
 - ▶ On crée une liste vide R
 - ▶ on regarde le premier élément de L1 et de L2.
 - ▶ on retire le plus petit des deux et on l'ajoute à R
 - ▶ Et on recommence tant que L1 et L2 sont non vides.

- ▶ cf. TD 9 (exercice 4)

SCRIPT

```
def tri_fusion(L):  
    if len(L) < 2:  
        return L  
    else:  
        moitié = len(L) // 2  
        L1 = tri_fusion(L[:moitié])  
        L2 = tri_fusion(L[moitié:])  
        return fusion(L1,L2)
```

- ▶ il reste à écrire l'opération de fusion
 - ▶ On crée une liste vide R
 - ▶ on regarde le premier élément de L1 et de L2.
 - ▶ on retire le plus petit des deux et on l'ajoute à R
 - ▶ Et on recommence tant que L1 et L2 sont non vides.

L1 = [3, 6] L2 = [1, 12] R = []

- ▶ cf. TD 9 (exercice 4)

SCRIPT

```
def tri_fusion(L):  
    if len(L) < 2:  
        return L  
    else:  
        moitié = len(L) // 2  
        L1 = tri_fusion(L[:moitié])  
        L2 = tri_fusion(L[moitié:])  
        return fusion(L1,L2)
```

- ▶ il reste à écrire l'opération de fusion
 - ▶ On crée une liste vide R
 - ▶ on regarde le premier élément de L1 et de L2.
 - ▶ on retire le plus petit des deux et on l'ajoute à R
 - ▶ Et on recommence tant que L1 et L2 sont non vides.

L1 = [3, 6] L2 = [1, 12] R = []

L1 = [3, 6] L2 = [12] R = [1]

- ▶ cf. TD 9 (exercice 4)

SCRIPT

```
def tri_fusion(L):  
    if len(L) < 2:  
        return L  
    else:  
        moitié = len(L) // 2  
        L1 = tri_fusion(L[:moitié])  
        L2 = tri_fusion(L[moitié:])  
        return fusion(L1,L2)
```

- ▶ il reste à écrire l'opération de fusion
 - ▶ On crée une liste vide R
 - ▶ on regarde le premier élément de L1 et de L2.
 - ▶ on retire le plus petit des deux et on l'ajoute à R
 - ▶ Et on recommence tant que L1 et L2 sont non vides.

L1 = [3, 6] L2 = [1, 12] R = []

L1 = [3, 6] L2 = [12] R = [1]

L1 = [6] L2 = [12] R = [1, 3]

- ▶ cf. TD 9 (exercice 4)

SCRIPT

```
def tri_fusion(L):  
    if len(L) < 2:  
        return L  
    else:  
        moitié = len(L) // 2  
        L1 = tri_fusion(L[:moitié])  
        L2 = tri_fusion(L[moitié:])  
        return fusion(L1,L2)
```

- ▶ il reste à écrire l'opération de fusion
 - ▶ On crée une liste vide R
 - ▶ on regarde le premier élément de L1 et de L2.
 - ▶ on retire le plus petit des deux et on l'ajoute à R
 - ▶ Et on recommence tant que L1 et L2 sont non vides.

L1 = [3, 6] L2 = [1, 12] R = []

L1 = [3, 6] L2 = [12] R = [1]

L1 = [6] L2 = [12] R = [1, 3]

L1 = [] L2 = [12] R = [1, 3, 6]

- ▶ cf. TD 9 (exercice 4)

SCRIPT

```
def tri_fusion(L):
    if len(L) < 2:
        return L
    else:
        moitié = len(L) // 2
        L1 = tri_fusion(L[:moitié])
        L2 = tri_fusion(L[moitié:])
        return fusion(L1,L2)
```

- ▶ il reste à écrire l'opération de fusion
 - ▶ On crée une liste vide R
 - ▶ on regarde le premier élément de L1 et de L2.
 - ▶ on retire le plus petit des deux et on l'ajoute à R
 - ▶ Et on recommence tant que L1 et L2 sont non vides.

L1 = [3, 6] L2 = [1, 12] R = []

L1 = [3, 6] L2 = [12] R = [1]

L1 = [6] L2 = [12] R = [1, 3]

L1 = [] L2 = [12] R = [1, 3, 6]

L1 = [] L2 = [] R = [1, 3, 6, 12]

- ▶ cf. TD 9 (exercice 4)

- ▶ Il existe un algorithme souvent plus rapide que le tri fusion.

- ▶ Il existe un algorithme souvent plus rapide que le tri fusion.
 - ▶ Du moins en pratique

- ▶ Il existe un algorithme souvent plus rapide que le tri fusion.
 - ▶ Du moins en pratique
- ▶ On utilise aussi le principe DIVISER POUR RÉGNER

- ▶ Il existe un algorithme souvent plus rapide que le tri fusion.
 - ▶ Du moins en pratique
- ▶ On utilise aussi le principe DIVISER POUR RÉGNER
- ▶ L'algorithme est le suivant

- ▶ Il existe un algorithme souvent plus rapide que le tri fusion.
 - ▶ Du moins en pratique
- ▶ On utilise aussi le principe DIVISER POUR RÉGNER
- ▶ L'algorithme est le suivant
 - **Étape 1** : on choisit un élément pivot p de L
 - ▶ on peut prendre le premier élément de la liste

- ▶ Il existe un algorithme souvent plus rapide que le tri fusion.
 - ▶ Du moins en pratique
- ▶ On utilise aussi le principe DIVISER POUR RÉGNER
- ▶ L'algorithme est le suivant
 - **Étape 1** : on choisit un élément pivot p de L
 - ▶ on peut prendre le premier élément de la liste
 - **Étape 2** : On définit deux listes $L1$ et $L2$ de la manière suivante
 - ▶ $L1$ contient tous les éléments de L plus petits que p
 - ▶ $L2$ contient tous les éléments de L plus grands que p

- ▶ Il existe un algorithme souvent plus rapide que le tri fusion.
 - ▶ Du moins en pratique
- ▶ On utilise aussi le principe DIVISER POUR RÉGNER
- ▶ L'algorithme est le suivant
 - **Étape 1** : on choisit un élément pivot p de L
 - ▶ on peut prendre le premier élément de la liste
 - **Étape 2** : On définit deux listes $L1$ et $L2$ de la manière suivante
 - ▶ $L1$ contient tous les éléments de L plus petits que p
 - ▶ $L2$ contient tous les éléments de L plus grands que p
 - **Étape 3** : On trie $L1$ et $L2$ de manière récursive.

- ▶ Il existe un algorithme souvent plus rapide que le tri fusion.
 - ▶ Du moins en pratique
- ▶ On utilise aussi le principe DIVISER POUR RÉGNER
- ▶ L'algorithme est le suivant
 - **Étape 1** : on choisit un élément pivot p de L
 - ▶ on peut prendre le premier élément de la liste
 - **Étape 2** : On définit deux listes $L1$ et $L2$ de la manière suivante
 - ▶ $L1$ contient tous les éléments de L plus petits que p
 - ▶ $L2$ contient tous les éléments de L plus grands que p
 - **Étape 3** : On trie $L1$ et $L2$ de manière récursive.
 - **Étape 4** : On recolle les morceaux $L1 + [p] + L2$

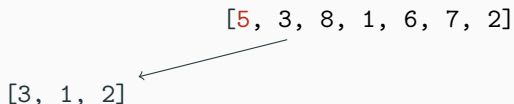
► Posons $L = [5, 3, 8, 1, 7, 2]$

$[5, 3, 8, 1, 6, 7, 2]$

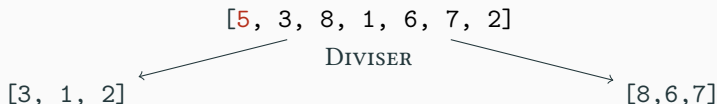
- ▶ Posons $L = [5, 3, 8, 1, 7, 2]$
- ▶ On choisit $p=L[0]=5$ comme pivot. L devient $[3, 8, 1, 7, 2]$

$[5, 3, 8, 1, 6, 7, 2]$

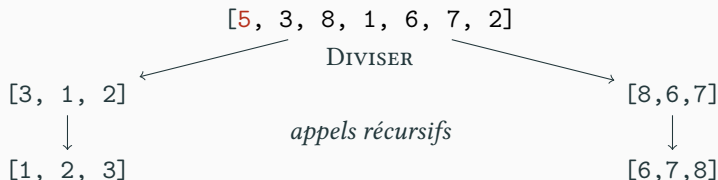
- ▶ Posons $L = [5, 3, 8, 1, 7, 2]$
- ▶ On choisit $p=L[0]=5$ comme pivot. L devient $[3, 8, 1, 7, 2]$
 - ▶ On prend les éléments plus petits que p pour créer $L1 = [3,1,2]$



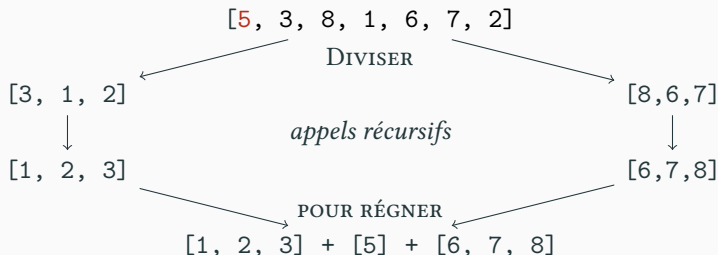
- ▶ Posons $L = [5, 3, 8, 1, 7, 2]$
- ▶ On choisit $p=L[0]=5$ comme pivot. L devient $[3, 8, 1, 7, 2]$
 - ▶ On prend les éléments plus petits que p pour créer $L1 = [3,1,2]$
 - ▶ On prend les éléments plus grands que p pour créer $L2 = [7,8,6]$



- ▶ Posons $L = [5, 3, 8, 1, 7, 2]$
- ▶ On choisit $p=L[0]=5$ comme pivot. L devient $[3, 8, 1, 7, 2]$
 - ▶ On prend les éléments plus petits que p pour créer $L1 = [3,1,2]$
 - ▶ On prend les éléments plus grands que p pour créer $L2 = [7,8,6]$
- ▶ On trie $L1$ et $L2$ de manière récursive



- ▶ Posons $L = [5, 3, 8, 1, 7, 2]$
- ▶ On choisit $p=L[0]=5$ comme pivot. L devient $[3, 8, 1, 7, 2]$
 - ▶ On prend les éléments plus petits que p pour créer $L1 = [3,1,2]$
 - ▶ On prend les éléments plus grands que p pour créer $L2 = [7,8,6]$
- ▶ On trie $L1$ et $L2$ de manière récursive
- ▶ On concatène les trois listes : $[1,2,3] + [5] + [6,7,8]$




```
def quicksort(L): # Tri rapide
```

SCRIPT

SCRIPT

```
def quicksort(L): # Tri rapide
    if len(L) <= 1: # Cas d'arrêt de notre fonction récursive
        return L
    else:
```

SCRIPT

```
def quicksort(L): # Tri rapide
    if len(L) <= 1: # Cas d'arrêt de notre fonction récursive
        return L
    else:
        # On divise
        p = L[0] # le pivot
        L1 = [ x for x in L[1:] if x < p ]
        L2 = [ x for x in L[1:] if x >= p ]
```

- ▶ Une compréhension de listes facilite l'écriture de la scission.

SCRIPT

```
def quicksort(L): # Tri rapide
    if len(L) <= 1: # Cas d'arrêt de notre fonction récursive
        return L
    else:
        # On divise
        p = L[0] # le pivot
        L1 = [ x for x in L[1:] if x < p ]
        L2 = [ x for x in L[1:] if x >= p ]
        # On trie
        L1 = quicksort(L1) # tri récursif
        L2 = quicksort(L2) # tri récursif
```

- ▶ Une compréhension de listes facilite l'écriture de la scission.

SCRIPT

```
def quicksort(L): # Tri rapide
    if len(L) <= 1: # Cas d'arrêt de notre fonction récursive
        return L
    else:
        # On divise
        p = L[0] # le pivot
        L1 = [ x for x in L[1:] if x < p ]
        L2 = [ x for x in L[1:] if x >= p ]
        # On trie
        L1 = quicksort(L1) # tri récursif
        L2 = quicksort(L2) # tri récursif
        # on règne !
        return L1 + [p] + L2
```

- ▶ Une compréhension de listes facilite l'écriture de la scission.

SCRIPT

```
def quicksort(L): # Tri rapide
    if len(L) <= 1: # Cas d'arrêt de notre fonction récursive
        return L
    else:
        # On divise
        p = L[0] # le pivot
        L1 = [ x for x in L[1:] if x < p ]
        L2 = [ x for x in L[1:] if x >= p ]
        # On trie
        L1 = quicksort(L1) # tri récursif
        L2 = quicksort(L2) # tri récursif
        # on règne !
        return L1 + [p] + L2
```

- ▶ Une compréhension de listes facilite l'écriture de la scission.
- ▶ Quel élément choisir pour le pivot ?

SCRIPT

```
def quicksort(L): # Tri rapide
    if len(L) <= 1: # Cas d'arrêt de notre fonction récursive
        return L
    else:
        # On divise
        p = L[0] # le pivot
        L1 = [ x for x in L[1:] if x < p ]
        L2 = [ x for x in L[1:] if x >= p ]
        # On trie
        L1 = quicksort(L1) # tri récursif
        L2 = quicksort(L2) # tri récursif
        # on règne !
        return L1 + [p] + L2
```

- ▶ Une compréhension de listes facilite l'écriture de la scission.
- ▶ Quel élément choisir pour le pivot?
 - ▶ Pour une liste dans le désordre, `L[0]` est un excellent choix.

SCRIPT

```
def quicksort(L): # Tri rapide
    if len(L) <= 1: # Cas d'arrêt de notre fonction récursive
        return L
    else:
        # On divise
        p = L[0] # le pivot
        L1 = [ x for x in L[1:] if x < p ]
        L2 = [ x for x in L[1:] if x >= p ]
        # On trie
        L1 = quicksort(L1) # tri récursif
        L2 = quicksort(L2) # tri récursif
        # on règne !
        return L1 + [p] + L2
```

- ▶ Une compréhension de listes facilite l'écriture de la scission.
- ▶ Quel élément choisir pour le pivot?
 - ▶ Pour une liste dans le désordre, $L[0]$ est un excellent choix.
 - ▶ Si la liste est presque triée, on peut prendre $L[m]$ où m est le milieu de L .

SCRIPT

```
def quicksort(L): # Tri rapide
    if len(L) <= 1: # Cas d'arrêt de notre fonction récursive
        return L
    else:
        # On divise
        p = L[0] # le pivot
        L1 = [ x for x in L[1:] if x < p ]
        L2 = [ x for x in L[1:] if x >= p ]
        # On trie
        L1 = quicksort(L1) # tri récursif
        L2 = quicksort(L2) # tri récursif
        # on règne !
        return L1 + [p] + L2
```

- ▶ Une compréhension de listes facilite l'écriture de la scission.
- ▶ Quel élément choisir pour le pivot?
 - ▶ Pour une liste dans le désordre, $L[0]$ est un excellent choix.
 - ▶ Si la liste est presque triée, on peut prendre $L[m]$ où m est le milieu de L .
 - ▶ On peut aussi choisir au hasard dans la liste (cela fonctionne bien).

 Partie I. Programmation graphique

 Partie II. Algorithmes

 Partie III. Algorithmes de tri

 Partie IV. Complexité

 Partie V. Algorithmes de recherche

 Partie VI. Crible d'Ératosthène

 Partie VII. Bilan

 Partie VIII. Table des matières

- ▶ Quel est l'algorithme le plus efficace ?
- ▶ Une première approche consiste à mesurer le temps de calcul

SCRIPT

```

from time import time
from random import randint

def chrono(f,x):
    début = time()
    f(x)
    fin = time()
    durée = fin - début
    return durée

def liste_aléatoire(n):
    return [ randint(1,100) for i in range(n) ]

def test():
    L = liste_aléatoire(500)
    for f in [sorted, quicksort, tri_fusion, tri_select]:
        s = f '{f.__name__:<10} -> {chrono(f,L):.6f}'
        print(s)
    
```

- ▶ Rappels :
 - ▶ `sorted` est l'algorithme de base utilisé par Python
 - ▶ `tri_select` est l'algorithme vu dans le cours 5 sur les listes

```
>>>
```

SHELL

- ▶ Rappels :
 - ▶ `sorted` est l'algorithme de base utilisé par Python
 - ▶ `tri_select` est l'algorithme vu dans le cours 5 sur les listes

```
>>> test()
```

SHELL

▶ Rappels :

- ▶ `sorted` est l'algorithme de base utilisé par Python
- ▶ `tri_select` est l'algorithme vu dans le cours 5 sur les listes

```
>>> test()  
sorted      -> 0.000046  
quicksort  -> 0.000637  
tri_fusion  -> 0.002608  
tri_select  -> 0.003999  
>>>
```

SHELL

▶ Rappels :

- ▶ `sorted` est l'algorithme de base utilisé par Python
- ▶ `tri_select` est l'algorithme vu dans le cours 5 sur les listes

```
>>> test()
sorted      -> 0.000046
quicksort  -> 0.000637
tri_fusion  -> 0.002608
tri_select  -> 0.003999
>>> test()
```

SHELL

► Rappels :

- `sorted` est l'algorithme de base utilisé par Python
- `tri_select` est l'algorithme vu dans le cours 5 sur les listes

```

>>> test()
sorted      -> 0.000046
quicksort   -> 0.000637
tri_fusion  -> 0.002608
tri_select  -> 0.003999
>>> test()
sorted      -> 0.000046
quicksort   -> 0.000604
tri_fusion  -> 0.002290
tri_select  -> 0.003985
    
```

SHELL

- ▶ Rappels :
 - ▶ `sorted` est l'algorithme de base utilisé par Python
 - ▶ `tri_select` est l'algorithme vu dans le cours 5 sur les listes

```
>>> test()
sorted      -> 0.000046
quicksort  -> 0.000637
tri_fusion -> 0.002608
tri_select -> 0.003999
>>> test()
sorted      -> 0.000046
quicksort  -> 0.000604
tri_fusion -> 0.002290
tri_select -> 0.003985
```

SHELL

- ▶ Ceci n'est pas une preuve rigoureuse.
 - ▶ Peut-être que sur certaines listes, `sorted` est plus lent que `quicksort` ?

- ▶ Rappels :
 - ▶ `sorted` est l'algorithme de base utilisé par Python
 - ▶ `tri_select` est l'algorithme vu dans le cours 5 sur les listes

```

>>> test()
sorted      -> 0.000046
quicksort   -> 0.000637
tri_fusion  -> 0.002608
tri_select  -> 0.003999
>>> test()
sorted      -> 0.000046
quicksort   -> 0.000604
tri_fusion  -> 0.002290
tri_select  -> 0.003985
    
```

SHELL

- ▶ Ceci n'est pas une preuve rigoureuse.
 - ▶ Peut-être que sur certaines listes, `sorted` est plus lent que `quicksort` ?
- ▶ Cela ne permet pas de comprendre pourquoi un algorithme est efficace.

- ▶ Nous allons calculer la complexité du tri fusion.

- ▶ Nous allons calculer la complexité du tri fusion.
 - Nous allons nous contenter du calcul du nombre de comparaisons,
 - ▶ Nous aurions pu aussi compter le nombre de lecture et d'écriture.
 - ▶ Le résultat final aurait été le même.
 - ▶ Et ceci car nous ne nous intéressons qu'à l'ordre de grandeur.

- ▶ Nous allons calculer la complexité du tri fusion.
 - Nous allons nous contenter du calcul du nombre de comparaisons,
 - ▶ Nous aurions pu aussi compter le nombre de lecture et d'écriture.
 - ▶ Le résultat final aurait été le même.
 - ▶ Et ceci car nous ne nous intéressons qu'à l'ordre de grandeur.
- ▶ Notons C_n le nombre de comparaisons.

- ▶ Nous allons calculer la complexité du tri fusion.
 - Nous allons nous contenter du calcul du nombre de comparaisons,
 - ▶ Nous aurions pu aussi compter le nombre de lecture et d'écriture.
 - ▶ Le résultat final aurait été le même.
 - ▶ Et ceci car nous ne nous intéressons qu'à l'ordre de grandeur.
- ▶ Notons C_n le nombre de comparaisons.
 - Appels récursifs
 - ▶ Les appels récursifs nécessitent $\frac{C_n}{2}$ comparaisons chacun.

- ▶ Nous allons calculer la complexité du tri fusion.
 - Nous allons nous contenter du calcul du nombre de comparaisons,
 - ▶ Nous aurions pu aussi compter le nombre de lecture et d'écriture.
 - ▶ Le résultat final aurait été le même.
 - ▶ Et ceci car nous ne nous intéressons qu'à l'ordre de grandeur.

- ▶ Notons C_n le nombre de comparaisons.
 - Appels récursifs
 - ▶ Les appels récursifs nécessitent $\frac{C_n}{2}$ comparaisons chacun.
 - Diviser et régner
 - ▶ DIVISER se fait en 0 comparaison donc $O(1)$ (mais $O(n)$ opérations).
 - ▶ La fusion peut être faite en $O(n)$ (cf. TD)
 - ▶ on montre aisément que $O(1) + O(n) = O(n)$

- ▶ Nous allons calculer la complexité du tri fusion.
 - Nous allons nous contenter du calcul du nombre de comparaisons,
 - ▶ Nous aurions pu aussi compter le nombre de lecture et d'écriture.
 - ▶ Le résultat final aurait été le même.
 - ▶ Et ceci car nous ne nous intéressons qu'à l'ordre de grandeur.
- ▶ Notons C_n le nombre de comparaisons.
 - Appels récursifs
 - ▶ Les appels récursifs nécessitent $C_{\frac{n}{2}}$ comparaisons chacun.
 - Diviser et régner
 - ▶ DIVISER se fait en 0 comparaison donc $O(1)$ (mais $O(n)$ opérations).
 - ▶ La fusion peut être faite en $O(n)$ (cf. TD)
 - ▶ on montre aisément que $O(1) + O(n) = O(n)$
- ▶ Nous avons donc la relation de récurrence suivante : $C_n = 2 \cdot C_{\frac{n}{2}} + O(n)$

▶ Comment résoudre : $C_n = 2 \cdot C_{\frac{n}{2}} + O(n)$?

▶ Dans le cas où n est une puissance de 2 c'est à dire $n = 2^{\log_2(n)}$

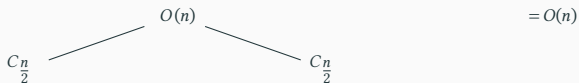
$$C_n =$$

$$C_n$$

► Comment résoudre : $C_n = 2 \cdot C_{\frac{n}{2}} + O(n)$?

► Dans le cas où n est une puissance de 2 c'est à dire $n = 2^{\log_2(n)}$

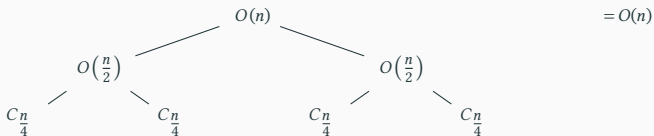
$$C_n = 2 \cdot C_{\frac{n}{2}} + O(n)$$



► Comment résoudre : $C_n = 2 \cdot C_{\frac{n}{2}} + O(n)$?

► Dans le cas où n est une puissance de 2 c'est à dire $n = 2^{\log_2(n)}$

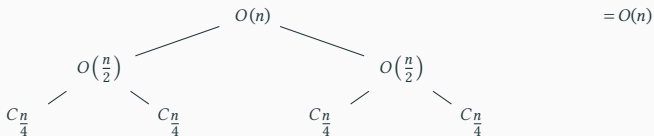
$$C_n = 2 \cdot C_{\frac{n}{2}} + O(n) = 2 \cdot \left(2 \cdot C_{\frac{n}{4}} + O\left(\frac{n}{2}\right) \right) + O(n)$$



► Comment résoudre : $C_n = 2 \cdot C_{\frac{n}{2}} + O(n)$?

► Dans le cas où n est une puissance de 2 c'est à dire $n = 2^{\log_2(n)}$

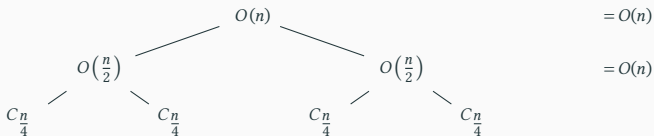
$$\begin{aligned} C_n &= 2 \cdot C_{\frac{n}{2}} + O(n) = 2 \cdot \left(2 \cdot C_{\frac{n}{4}} + O\left(\frac{n}{2}\right) \right) + O(n) \\ &= 4 \cdot C_{\frac{n}{4}} + 2 \cdot O\left(\frac{n}{2}\right) + O(n) \end{aligned}$$



► Comment résoudre : $C_n = 2 \cdot C_{\frac{n}{2}} + O(n)$?

► Dans le cas où n est une puissance de 2 c'est à dire $n = 2^{\log_2(n)}$

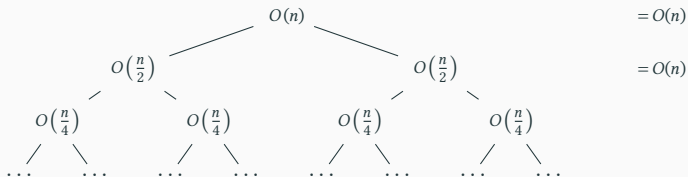
$$\begin{aligned} C_n &= 2 \cdot C_{\frac{n}{2}} + O(n) = 2 \cdot \left(2 \cdot C_{\frac{n}{4}} + O\left(\frac{n}{2}\right) \right) + O(n) \\ &= 4 \cdot C_{\frac{n}{4}} + 2 \cdot O\left(\frac{n}{2}\right) + O(n) = 4 \cdot C_{\frac{n}{4}} + O(n) + O(n) \end{aligned}$$



► Comment résoudre : $C_n = 2 \cdot C_{\frac{n}{2}} + O(n)$?

► Dans le cas où n est une puissance de 2 c'est à dire $n = 2^{\log_2(n)}$

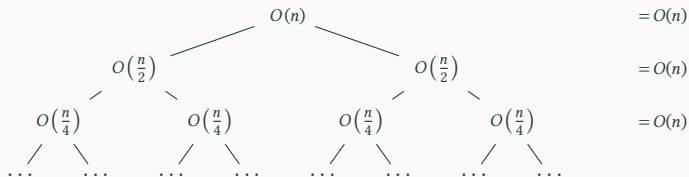
$$\begin{aligned} C_n &= 2 \cdot C_{\frac{n}{2}} + O(n) = 2 \cdot \left(2 \cdot C_{\frac{n}{4}} + O\left(\frac{n}{2}\right) \right) + O(n) \\ &= 4 \cdot C_{\frac{n}{4}} + 2 \cdot O\left(\frac{n}{2}\right) + O(n) = 4 \cdot C_{\frac{n}{4}} + O(n) + O(n) \end{aligned}$$



► Comment résoudre : $C_n = 2 \cdot C_{\frac{n}{2}} + O(n)$?

► Dans le cas où n est une puissance de 2 c'est à dire $n = 2^{\log_2(n)}$

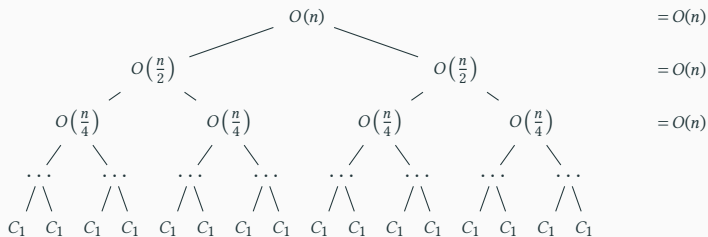
$$\begin{aligned} C_n &= 2 \cdot C_{\frac{n}{2}} + O(n) = 2 \cdot \left(2 \cdot C_{\frac{n}{4}} + O\left(\frac{n}{2}\right) \right) + O(n) \\ &= 4 \cdot C_{\frac{n}{4}} + 2 \cdot O\left(\frac{n}{2}\right) + O(n) = 4 \cdot C_{\frac{n}{4}} + O(n) + O(n) \end{aligned}$$



► Comment résoudre : $C_n = 2 \cdot C_{\frac{n}{2}} + O(n)$?

► Dans le cas où n est une puissance de 2 c'est à dire $n = 2^{\log_2(n)}$

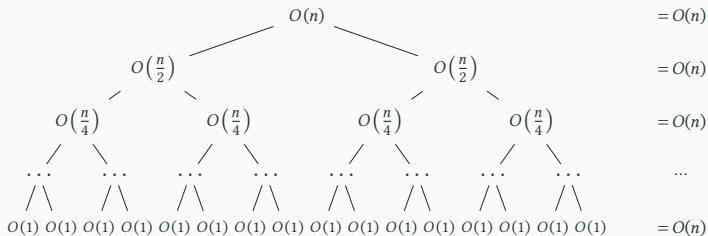
$$\begin{aligned} C_n &= 2 \cdot C_{\frac{n}{2}} + O(n) = 2 \cdot \left(2 \cdot C_{\frac{n}{4}} + O\left(\frac{n}{2}\right) \right) + O(n) \\ &= 4 \cdot C_{\frac{n}{4}} + 2 \cdot O\left(\frac{n}{2}\right) + O(n) = 4 \cdot C_{\frac{n}{4}} + O(n) + O(n) \end{aligned}$$



► Comment résoudre : $C_n = 2 \cdot C_{\frac{n}{2}} + O(n)$?

► Dans le cas où n est une puissance de 2 c'est à dire $n = 2^{\log_2(n)}$

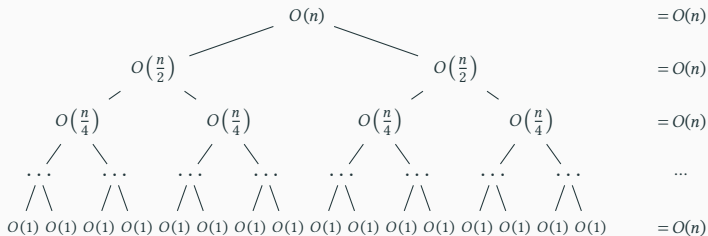
$$\begin{aligned} C_n &= 2 \cdot C_{\frac{n}{2}} + O(n) = 2 \cdot \left(2 \cdot C_{\frac{n}{4}} + O\left(\frac{n}{2}\right) \right) + O(n) \\ &= 4 \cdot C_{\frac{n}{4}} + 2 \cdot O\left(\frac{n}{2}\right) + O(n) = 4 \cdot C_{\frac{n}{4}} + O(n) + O(n) \end{aligned}$$



► Comment résoudre : $C_n = 2 \cdot C_{\frac{n}{2}} + O(n)$?

► Dans le cas où n est une puissance de 2 c'est à dire $n = 2^{\log_2(n)}$

$$\begin{aligned} C_n &= 2 \cdot C_{\frac{n}{2}} + O(n) = 2 \cdot \left(2 \cdot C_{\frac{n}{4}} + O\left(\frac{n}{2}\right) \right) + O(n) \\ &= 4 \cdot C_{\frac{n}{4}} + 2 \cdot O\left(\frac{n}{2}\right) + O(n) = 4 \cdot C_{\frac{n}{4}} + O(n) + O(n) \end{aligned}$$



► Au bout du calcul on aura (la hauteur de l'arbre étant $\log_2(n)$) :

$$C_n = O(n) + \dots + O(n) = (\log_2(n) + 1) \cdot O(n) = O(n \cdot \log n)$$

- ▶ Le calcul précédent est correct car tous les $O(\dots)$ ont la même constante

- ▶ Le calcul précédent est correct car tous les $O(\dots)$ ont la même constante
 - ▶ Rappel : $f(n) = O(n)$ si et seulement il existe $k > 0$ tel que $f(n) \leq k \cdot n$

- ▶ Le calcul précédent est correct car tous les $O(\dots)$ ont la même constante
 - ▶ Rappel : $f(n) = O(n)$ si et seulement il existe $k > 0$ tel que $f(n) \leq k \cdot n$
- ▶ Pour une démonstration rigoureuse voir l'**excellent** «Cormen» (chap. 4) :
 - ▶ Introduction à l'algorithmique, *T. Cormen, C. Leiserson, R. Rivest, C. Stein*
 - ▶ Disponible en BU et dans toutes les bonnes librairies.

- ▶ Le calcul précédent est correct car tous les $O(\dots)$ ont la même constante
 - ▶ Rappel : $f(n) = O(n)$ si et seulement il existe $k > 0$ tel que $f(n) \leq k \cdot n$
- ▶ Pour une démonstration rigoureuse voir l'**excellent** «Cormen» (chap. 4) :
 - ▶ Introduction à l'algorithmique, *T. Cormen, C. Leiserson, R. Rivest, C. Stein*
 - ▶ Disponible en BU et dans toutes les bonnes librairies.
- ▶ Pour du tri, on ne peut pas faire mieux que $O(n \cdot \log n)$

- ▶ Le calcul précédent est correct car tous les $O(\dots)$ ont la même constante
 - ▶ Rappel : $f(n) = O(n)$ si et seulement il existe $k > 0$ tel que $f(n) \leq k \cdot n$
- ▶ Pour une démonstration rigoureuse voir l'**excellent** «Cormen» (chap. 4) :
 - ▶ Introduction à l'algorithmique, *T. Cormen, C. Leiserson, R. Rivest, C. Stein*
 - ▶ Disponible en BU et dans toutes les bonnes librairies.
- ▶ Pour du tri, on ne peut pas faire mieux que $O(n \cdot \log n)$
 - En ce sens le tri fusion est optimal.

- ▶ Le calcul précédent est correct car tous les $O(\dots)$ ont la même constante
 - ▶ Rappel : $f(n) = O(n)$ si et seulement il existe $k > 0$ tel que $f(n) \leq k \cdot n$
- ▶ Pour une démonstration rigoureuse voir l'**excellent** «Cormen» (chap. 4) :
 - ▶ Introduction à l'algorithmique, *T. Cormen, C. Leiserson, R. Rivest, C. Stein*
 - ▶ Disponible en BU et dans toutes les bonnes librairies.
- ▶ Pour du tri, on ne peut pas faire mieux que $O(n \cdot \log n)$
 - En ce sens le tri fusion est optimal.
 - Le tri rapide n'est pas en $O(n \cdot \log n)$.

- ▶ Le calcul précédent est correct car tous les $O(\dots)$ ont la même constante
 - ▶ Rappel : $f(n) = O(n)$ si et seulement il existe $k > 0$ tel que $f(n) \leq k \cdot n$
- ▶ Pour une démonstration rigoureuse voir l'**excellent** «Cormen» (chap. 4) :
 - ▶ Introduction à l'algorithmique, *T. Cormen, C. Leiserson, R. Rivest, C. Stein*
 - ▶ Disponible en BU et dans toutes les bonnes librairies.
- ▶ Pour du tri, on ne peut pas faire mieux que $O(n \cdot \log n)$
 - En ce sens le tri fusion est optimal.
 - Le tri rapide n'est pas en $O(n \cdot \log n)$.
 - ▶ Dans le pire cas il est en $O(n^2)$

- ▶ Le calcul précédent est correct car tous les $O(\dots)$ ont la même constante
 - ▶ Rappel : $f(n) = O(n)$ si et seulement il existe $k > 0$ tel que $f(n) \leq k \cdot n$
- ▶ Pour une démonstration rigoureuse voir l'**excellent** «Cormen» (chap. 4) :
 - ▶ Introduction à l'algorithmique, *T. Cormen, C. Leiserson, R. Rivest, C. Stein*
 - ▶ Disponible en BU et dans toutes les bonnes librairies.
- ▶ Pour du tri, on ne peut pas faire mieux que $O(n \cdot \log n)$
 - En ce sens le tri fusion est optimal.
 - Le tri rapide n'est pas en $O(n \cdot \log n)$.
 - ▶ Dans le pire cas il est en $O(n^2)$
 - ▶ Mais en moyenne et dans la plupart des cas il est bien en $O(n \cdot \log n)$.

- ▶ Le calcul précédent est correct car tous les $O(\dots)$ ont la même constante
 - ▶ Rappel : $f(n) = O(n)$ si et seulement il existe $k > 0$ tel que $f(n) \leq k \cdot n$
- ▶ Pour une démonstration rigoureuse voir l'**excellent** «Cormen» (chap. 4) :
 - ▶ Introduction à l'algorithmique, *T. Cormen, C. Leiserson, R. Rivest, C. Stein*
 - ▶ Disponible en BU et dans toutes les bonnes librairies.
- ▶ Pour du tri, on ne peut pas faire mieux que $O(n \cdot \log n)$
 - En ce sens le tri fusion est optimal.
 - Le tri rapide n'est pas en $O(n \cdot \log n)$.
 - ▶ Dans le pire cas il est en $O(n^2)$
 - ▶ Mais en moyenne et dans la plupart des cas il est bien en $O(n \cdot \log n)$.
 - ▶ En pratique et avec une bonne implémentation, il est même plus rapide que le tri fusion !

- ▶ Le calcul précédent est correct car tous les $O(\dots)$ ont la même constante
 - ▶ Rappel : $f(n) = O(n)$ si et seulement il existe $k > 0$ tel que $f(n) \leq k \cdot n$
- ▶ Pour une démonstration rigoureuse voir l'**excellent** «Cormen» (chap. 4) :
 - ▶ Introduction à l'algorithmique, *T. Cormen, C. Leiserson, R. Rivest, C. Stein*
 - ▶ Disponible en BU et dans toutes les bonnes librairies.
- ▶ Pour du tri, on ne peut pas faire mieux que $O(n \cdot \log n)$
 - En ce sens le tri fusion est optimal.
 - Le tri rapide n'est pas en $O(n \cdot \log n)$.
 - ▶ Dans le pire cas il est en $O(n^2)$
 - ▶ Mais en moyenne et dans la plupart des cas il est bien en $O(n \cdot \log n)$.
 - ▶ En pratique et avec une bonne implémentation, il est même plus rapide que le tri fusion !
 - ▶ En effet on peut le programmer « sur place », c'est-à-dire sans créer de nouvelles listes en mémoire.

 Partie I. Programmation graphique

 Partie II. Algorithmes

 Partie III. Algorithmes de tri

 Partie IV. Complexité

 **Partie V. Algorithmes de recherche**

 Partie VI. Crible d'Ératosthène

 Partie VII. Bilan

 Partie VIII. Table des matières

- ▶ Problème fréquent en programmation :
 - rechercher la position d'un élément dans une séquence (ici une liste).

- ▶ Problème fréquent en programmation :
 - rechercher la position d'un élément dans une séquence (ici une liste).
 - Soit L une liste de nombres non triée.
 - ▶ Par exemple, $L = [3, 2, 8, 6, 2, 4, 7, 6, 5]$

- ▶ Problème fréquent en programmation :
 - rechercher la position d'un élément dans une séquence (ici une liste).
 - Soit L une liste de nombres non triée.
 - ▶ Par exemple, $L = [3, 2, 8, 6, 2, 4, 7, 6, 5]$
 - On cherche l'élément x (par exemple $x = 6$)
 - ▶ S'il apparaît plusieurs fois, on renvoie le premier indice
 - ▶ S'il n'est pas dans la liste, on renvoie -1 .

- ▶ Problème fréquent en programmation :
 - rechercher la position d'un élément dans une séquence (ici une liste).
 - Soit L une liste de nombres non triée.
 - ▶ Par exemple, $L = [3, 2, 8, 6, 2, 4, 7, 6, 5]$
 - On cherche l'élément x (par exemple $x = 6$)
 - ▶ S'il apparaît plusieurs fois, on renvoie le premier indice
 - ▶ S'il n'est pas dans la liste, on renvoie -1 .

```
def chercher(x, L):  
    for i in range(len(L)):  
        if L[i] == x:  
            return i  
    return -1
```

SCRIPT

>>>

SHELL

- ▶ Problème fréquent en programmation :
 - rechercher la position d'un élément dans une séquence (ici une liste).
 - Soit L une liste de nombres non triée.
 - ▶ Par exemple, L = [3, 2, 8, 6, 2, 4, 7, 6, 5]
 - On cherche l'élément x (par exemple x = 6)
 - ▶ S'il apparaît plusieurs fois, on renvoie le premier indice
 - ▶ S'il n'est pas dans la liste, on renvoie -1.

```
def chercher(x, L):  
    for i in range(len(L)):  
        if L[i] == x:  
            return i  
    return -1
```

SCRIPT

```
>>> chercher(6,L)
```

SHELL

- ▶ Problème fréquent en programmation :
 - rechercher la position d'un élément dans une séquence (ici une liste).
 - Soit L une liste de nombres non triée.
 - ▶ Par exemple, L = [3, 2, 8, 6, 2, 4, 7, 6, 5]
 - On cherche l'élément x (par exemple x = 6)
 - ▶ S'il apparaît plusieurs fois, on renvoie le premier indice
 - ▶ S'il n'est pas dans la liste, on renvoie -1.

```
def chercher(x, L):  
    for i in range(len(L)):  
        if L[i] == x:  
            return i  
    return -1
```

SCRIPT

```
>>> chercher(6,L)  
3  
>>>
```

SHELL

- ▶ Problème fréquent en programmation :
 - rechercher la position d'un élément dans une séquence (ici une liste).
 - Soit L une liste de nombres non triée.
 - ▶ Par exemple, $L = [3, 2, 8, 6, 2, 4, 7, 6, 5]$
 - On cherche l'élément x (par exemple $x = 6$)
 - ▶ S'il apparaît plusieurs fois, on renvoie le premier indice
 - ▶ S'il n'est pas dans la liste, on renvoie -1 .

```
def chercher(x, L):  
    for i in range(len(L)):  
        if L[i] == x:  
            return i  
    return -1
```

SCRIPT

```
>>> chercher(6,L)  
3  
>>> chercher(9,L)
```

SHELL

- ▶ Problème fréquent en programmation :
 - rechercher la position d'un élément dans une séquence (ici une liste).
 - Soit L une liste de nombres non triée.
 - ▶ Par exemple, L = [3, 2, 8, 6, 2, 4, 7, 6, 5]
 - On cherche l'élément x (par exemple x = 6)
 - ▶ S'il apparaît plusieurs fois, on renvoie le premier indice
 - ▶ S'il n'est pas dans la liste, on renvoie -1.

```
def chercher(x, L):  
    for i in range(len(L)):  
        if L[i] == x:  
            return i  
    return -1
```

SCRIPT

```
>>> chercher(6,L)  
3  
>>> chercher(9,L)  
-1
```

SHELL

- ▶ Problème fréquent en programmation :
 - rechercher la position d'un élément dans une séquence (ici une liste).
 - Soit L une liste de nombres non triée.
 - ▶ Par exemple, $L = [3, 2, 8, 6, 2, 4, 7, 6, 5]$
 - On cherche l'élément x (par exemple $x = 6$)
 - ▶ S'il apparaît plusieurs fois, on renvoie le premier indice
 - ▶ S'il n'est pas dans la liste, on renvoie -1 .

```
def chercher(x, L):  
    for i in range(len(L)):  
        if L[i] == x:  
            return i  
    return -1
```

SCRIPT

```
>>> chercher(6,L)  
3  
>>> chercher(9,L)  
-1
```

SHELL

- ▶ Coût en $O(n)$ avec $n=\text{len}(L)$
 - ▶ Le temps de recherche est proportionnel à la taille de la liste.
 - ▶ Avec une liste 2 fois plus longue, la recherche prend 2 fois plus de temps

- ▶ Imaginez un dictionnaire dans le désordre,

- ▶ Imaginez un dictionnaire dans le désordre,
 - ▶ chercher un mot serait un enfer...
 - ▶ il faudrait lire tous les mots du dictionnaire pour trouver le bon !

- ▶ Imaginez un dictionnaire dans le désordre,
 - ▶ chercher un mot serait un enfer...
 - ▶ il faudrait lire tous les mots du dictionnaire pour trouver le bon !
- ▶ Un dictionnaire rangé par ordre alphabétique est beaucoup plus utile.
 - ▶ La taille du dictionnaire a alors peu d'influence sur temps de recherche.
 - ▶ On ouvre au milieu et on sait dans quelle moitié se trouve le mot cherché.

- ▶ Imaginez un dictionnaire dans le désordre,
 - ▶ chercher un mot serait un enfer...
 - ▶ il faudrait lire tous les mots du dictionnaire pour trouver le bon !
- ▶ Un dictionnaire rangé par ordre alphabétique est beaucoup plus utile.
 - ▶ La taille du dictionnaire a alors peu d'influence sur temps de recherche.
 - ▶ On ouvre au milieu et on sait dans quelle moitié se trouve le mot cherché.
- ▶ Cet algorithme a un nom technique : **la recherche dichotomique**

- ▶ Imaginez un dictionnaire dans le désordre,
 - ▶ chercher un mot serait un enfer...
 - ▶ il faudrait lire tous les mots du dictionnaire pour trouver le bon !
- ▶ Un dictionnaire rangé par ordre alphabétique est beaucoup plus utile.
 - ▶ La taille du dictionnaire a alors peu d'influence sur temps de recherche.
 - ▶ On ouvre au milieu et on sait dans quelle moitié se trouve le mot cherché.
- ▶ Cet algorithme a un nom technique : **la recherche dichotomique**
 - ▶ On regarde au centre de l'intervalle de recherche

- ▶ Imaginez un dictionnaire dans le désordre,
 - ▶ chercher un mot serait un enfer...
 - ▶ il faudrait lire tous les mots du dictionnaire pour trouver le bon !
- ▶ Un dictionnaire rangé par ordre alphabétique est beaucoup plus utile.
 - ▶ La taille du dictionnaire a alors peu d'influence sur temps de recherche.
 - ▶ On ouvre au milieu et on sait dans quelle moitié se trouve le mot cherché.
- ▶ Cet algorithme a un nom technique : **la recherche dichotomique**
 - ▶ On regarde au centre de l'intervalle de recherche
 - ▶ On compare le mot cherché au mot du milieu

- ▶ Imaginez un dictionnaire dans le désordre,
 - ▶ chercher un mot serait un enfer...
 - ▶ il faudrait lire tous les mots du dictionnaire pour trouver le bon !
- ▶ Un dictionnaire rangé par ordre alphabétique est beaucoup plus utile.
 - ▶ La taille du dictionnaire a alors peu d'influence sur temps de recherche.
 - ▶ On ouvre au milieu et on sait dans quelle moitié se trouve le mot cherché.
- ▶ Cet algorithme a un nom technique : **la recherche dichotomique**
 - ▶ On regarde au centre de l'intervalle de recherche
 - ▶ On compare le mot cherché au mot du milieu
 - ▶ On prend comme nouvel intervalle la moitié contenant notre mot.

- ▶ Imaginez un dictionnaire dans le désordre,
 - ▶ chercher un mot serait un enfer...
 - ▶ il faudrait lire tous les mots du dictionnaire pour trouver le bon !
- ▶ Un dictionnaire rangé par ordre alphabétique est beaucoup plus utile.
 - ▶ La taille du dictionnaire a alors peu d'influence sur temps de recherche.
 - ▶ On ouvre au milieu et on sait dans quelle moitié se trouve le mot cherché.
- ▶ Cet algorithme a un nom technique : **la recherche dichotomique**
 - ▶ On regarde au centre de l'intervalle de recherche
 - ▶ On compare le mot cherché au mot du milieu
 - ▶ On prend comme nouvel intervalle la moitié contenant notre mot.
 - ▶ Et on recommence avec le nouvel intervalle.

SCRIPT

```
# On cherche x entre L[début] et L[fin], L étant triée
def dicho(x,L,début,fin):
    m = (début+fin)//2
    if début > fin:
        return -1
    elif x<L[m]:
        return dicho(x,L,début,m-1)
    elif x>L[m]:
        return dicho(x,L,m+1,fin)
    else:
        return m

# Initialise la recherche dichotomique avec début et fin
def est_présent(x,L):
    """ L doit être une liste triée """
    dicho(x,L,0,len(L)-1)
```

SCRIPT

```
# On cherche x entre L[début] et L[fin], L étant triée
def dichotom(x,L,début,fin):
    m = (début+fin)//2
    if début > fin:
        return -1
    elif x<L[m]:
        return dichotom(x,L,début,m-1)
    elif x>L[m]:
        return dichotom(x,L,m+1,fin)
    else:
        return m

# Initialise la recherche dichotomique avec début et fin
def est_présent(x,L):
    """ L doit être une liste triée """
    return dichotom(x,L,0,len(L)-1)
```

- Combien d'intervalles allons-nous visiter ?

SCRIPT

```
# On cherche x entre L[début] et L[fin], L étant triée
def dichotom(x,L,début,fin):
    m = (début+fin)//2
    if début > fin:
        return -1
    elif x<L[m]:
        return dichotom(x,L,début,m-1)
    elif x>L[m]:
        return dichotom(x,L,m+1,fin)
    else:
        return m

# Initialise la recherche dichotomique avec début et fin
def est_présent(x,L):
    """ L doit être une liste triée """
    return dichotom(x,L,0,len(L)-1)
```

- ▶ Combien d'intervalles allons-nous visiter?
 - ▶ À chaque étape on divise l'intervalle de recherche par 2.

SCRIPT

```
# On cherche x entre L[début] et L[fin], L étant triée
def dichotom(x,L,début,fin):
    m = (début+fin)//2
    if début > fin:
        return -1
    elif x<L[m]:
        return dichotom(x,L,début,m-1)
    elif x>L[m]:
        return dichotom(x,L,m+1,fin)
    else:
        return m

# Initialise la recherche dichotomique avec début et fin
def est_présent(x,L):
    """ L doit être une liste triée """
    return dichotom(x,L,0,len(L)-1)
```

- ▶ Combien d'intervalles allons-nous visiter?
 - ▶ À chaque étape on divise l'intervalle de recherche par 2.
 - ▶ si $n = 2^k$, on ne peut le faire que k fois

SCRIPT

```
# On cherche x entre L[début] et L[fin], L étant triée
def dichotom(x,L,début,fin):
    m = (début+fin)//2
    if début > fin:
        return -1
    elif x<L[m]:
        return dichotom(x,L,début,m-1)
    elif x>L[m]:
        return dichotom(x,L,m+1,fin)
    else:
        return m

# Initialise la recherche dichotomique avec début et fin
def est_présent(x,L):
    """ L doit être une liste triée """
    return dichotom(x,L,0,len(L)-1)
```

- ▶ Combien d'intervalles allons-nous visiter ?
 - ▶ À chaque étape on divise l'intervalle de recherche par 2.
 - ▶ si $n = 2^k$, on ne peut le faire que k fois
 - ▶ Plus généralement on ne peut le faire que $\log_2(n)$ fois.

SCRIPT

```
# On cherche x entre L[début] et L[fin], L étant triée
def dichotom(x,L,début,fin):
    m = (début+fin)//2
    if début > fin:
        return -1
    elif x<L[m]:
        return dichotom(x,L,début,m-1)
    elif x>L[m]:
        return dichotom(x,L,m+1,fin)
    else:
        return m

# Initialise la recherche dichotomique avec début et fin
def est_présent(x,L):
    """ L doit être une liste triée """
    return dichotom(x,L,0,len(L)-1)
```

- ▶ Combien d'intervalles allons-nous visiter ?
 - ▶ À chaque étape on divise l'intervalle de recherche par 2.
 - ▶ si $n = 2^k$, on ne peut le faire que k fois (remarque $k = \log_2(n)$)
 - ▶ Plus généralement on ne peut le faire que $\log_2(n)$ fois.

SCRIPT

```
# On cherche x entre L[début] et L[fin], L étant triée
def dichotom(x,L,début,fin):
    m = (début+fin)//2
    if début > fin:
        return -1
    elif x<L[m]:
        return dichotom(x,L,début,m-1)
    elif x>L[m]:
        return dichotom(x,L,m+1,fin)
    else:
        return m

# Initialise la recherche dichotomique avec début et fin
def est_présent(x,L):
    """ L doit être une liste triée """
    return dichotom(x,L,0,len(L)-1)
```

- ▶ Combien d'intervalles allons-nous visiter ?
 - ▶ À chaque étape on divise l'intervalle de recherche par 2.
 - ▶ si $n = 2^k$, on ne peut le faire que k fois (remarque $k = \log_2(n)$)
 - ▶ Plus généralement on ne peut le faire que $\log_2(n)$ fois.
- ▶ La recherche dichotomique est ainsi en $O(\log n)$
 - ▶ C'est particulièrement efficace !

- ▶ Prenons une liste triée de $16 = 2^4$ éléments

- ▶ Prenons une liste triée de $16 = 2^4$ éléments
 - ▶ La recherche se terminera donc en moins de 4 étapes

- ▶ Prenons une liste triée de $16 = 2^4$ éléments
 - ▶ La recherche se terminera donc en moins de 4 étapes
- ▶ On cherche le nombre 22

- ▶ Prenons une liste triée de $16 = 2^4$ éléments
 - ▶ La recherche se terminera donc en moins de 4 étapes
- ▶ On cherche le nombre 22
 - ▶ L'intervalle de recherche est en gris
 - ▶ Le milieu est en rouge

2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----

- ▶ Prenons une liste triée de $16 = 2^4$ éléments
 - ▶ La recherche se terminera donc en moins de 4 étapes
- ▶ On cherche le nombre 22
 - ▶ L'intervalle de recherche est en gris
 - ▶ Le milieu est en rouge

2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----

- ▶ On cherche à droite car $22 > 16$

2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----

- ▶ Prenons une liste triée de $16 = 2^4$ éléments
 - ▶ La recherche se terminera donc en moins de 4 étapes
- ▶ On cherche le nombre 22
 - ▶ L'intervalle de recherche est en gris
 - ▶ Le milieu est en rouge

2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----

- ▶ On cherche à droite car $22 > 16$

2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----

- ▶ On cherche à gauche car $22 < 24$

2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----

- ▶ Prenons une liste triée de $16 = 2^4$ éléments
 - ▶ La recherche se terminera donc en moins de 4 étapes
- ▶ On cherche le nombre 22
 - ▶ L'intervalle de recherche est en gris
 - ▶ Le milieu est en rouge

2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----

- ▶ On cherche à droite car $22 > 16$

2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----

- ▶ On cherche à gauche car $22 < 24$

2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----

- ▶ On cherche à droite car $22 > 20$

2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----

- ▶ Prenons une liste triée de $16 = 2^4$ éléments
 - ▶ La recherche se terminera donc en moins de 4 étapes
- ▶ On cherche le nombre 22
 - ▶ L'intervalle de recherche est en gris
 - ▶ Le milieu est en rouge

2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----

- ▶ On cherche à droite car $22 > 16$

2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----

- ▶ On cherche à gauche car $22 < 24$

2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----

- ▶ On cherche à droite car $22 > 20$

2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----

- ▶ Trouvé : il ne reste qu'un élément et c'est 22

- ▶ La recherche dichotomique est plus rapide que la recherche séquentielle.

Recherche dichotomique

Complexité de $O(\log n)$: rapide

Nécessite une liste triée

Recherche séquentielle

Complexité $O(n)$: lent

Fonctionne avec toutes les listes

- ▶ La recherche dichotomique est plus rapide que la recherche séquentielle.

Recherche dichotomique

Complexité de $O(\log n)$: rapide
Nécessite une liste triée

Recherche séquentielle

Complexité $O(n)$: lent
Fonctionne avec toutes les listes

- ▶ Et si on trie la liste pour faire la recherche dichotomique ?

- ▶ La recherche dichotomique est plus rapide que la recherche séquentielle.

Recherche dichotomique

Complexité de $O(\log n)$: rapide
Nécessite une liste triée

Recherche séquentielle

Complexité $O(n)$: lent
Fonctionne avec toutes les listes

- ▶ Et si on trie la liste pour faire la recherche dichotomique ?
 - ▶ Le tri est très coûteux $O(n \cdot \log n)$

- ▶ La recherche dichotomique est plus rapide que la recherche séquentielle.

Recherche dichotomique

Complexité de $O(\log n)$: rapide
Nécessite une liste triée

Recherche séquentielle

Complexité $O(n)$: lent
Fonctionne avec toutes les listes

- ▶ Et si on trie la liste pour faire la recherche dichotomique ?
 - ▶ Le tri est très coûteux $O(n \cdot \log n)$
 - ▶ bien plus que la recherche !
 - ▶ ce n'est donc pas rentable...

- ▶ La recherche dichotomique est plus rapide que la recherche séquentielle.

Recherche dichotomique

Complexité de $O(\log n)$: rapide
Nécessite une liste triée

Recherche séquentielle

Complexité $O(n)$: lent
Fonctionne avec toutes les listes

- ▶ Et si on trie la liste pour faire la recherche dichotomique ?
 - ▶ Le tri est très coûteux $O(n \cdot \log n)$
 - ▶ bien plus que la recherche !
 - ▶ ce n'est donc pas rentable...
- ▶ Sauf si on est amené à faire de nombreuses recherches.
 - ▶ Soit R le nombre de recherche.
 - ▶ recherche séquentielle : $O(R \cdot n)$: R recherches qui coûtent n
 - ▶ dichotomique (avec tri) = $O(R \cdot \log n + n \cdot \log n)$

- ▶ La recherche dichotomique est plus rapide que la recherche séquentielle.

Recherche dichotomique

Complexité de $O(\log n)$: rapide
Nécessite une liste triée

Recherche séquentielle

Complexité $O(n)$: lent
Fonctionne avec toutes les listes

- ▶ Et si on trie la liste pour faire la recherche dichotomique ?
 - ▶ Le tri est très coûteux $O(n \cdot \log n)$
 - ▶ bien plus que la recherche !
 - ▶ ce n'est donc pas rentable...
- ▶ Sauf si on est amené à faire de nombreuses recherches.
 - ▶ Soit R le nombre de recherche.
 - ▶ recherche séquentielle : $O(R \cdot n)$: R recherches qui coûtent n
 - ▶ dichotomique (avec tri) = $O(R \cdot \log n + n \cdot \log n)$
 - ▶ dichotomique (avec tri) $\approx O(R \cdot \log n)$ (pour R très grand devant n)

- ▶ La recherche dichotomique est plus rapide que la recherche séquentielle.

Recherche dichotomique

Complexité de $O(\log n)$: rapide
Nécessite une liste triée

Recherche séquentielle

Complexité $O(n)$: lent
Fonctionne avec toutes les listes

- ▶ Et si on trie la liste pour faire la recherche dichotomique ?
 - ▶ Le tri est très coûteux $O(n \cdot \log n)$
 - ▶ bien plus que la recherche !
 - ▶ ce n'est donc pas rentable...
- ▶ Sauf si on est amené à faire de nombreuses recherches.
 - ▶ Soit R le nombre de recherche.
 - ▶ recherche séquentielle : $O(R \cdot n)$: R recherches qui coûtent n
 - ▶ dichotomique (avec tri) = $O(R \cdot \log n + n \cdot \log n)$
 - ▶ dichotomique (avec tri) $\approx O(R \cdot \log n)$ (pour R très grand devant n)
- ▶ Il est rentable de trier une liste **si on doit y faire de nombreuses recherches**

 Partie I. Programmation graphique

 Partie II. Algorithmes

 Partie III. Algorithmes de tri

 Partie IV. Complexité

 Partie V. Algorithmes de recherche

 Partie VI. Crible d'Ératosthène

 Partie VII. Bilan

 Partie VIII. Table des matières

- ▶ Comment savoir si un nombre est premier ?

- ▶ Comment savoir si un nombre est premier ?
 - ▶ Nous avons vu dans le cours 2 un algorithme en $O(\sqrt{N})$

- ▶ Comment savoir si un nombre est premier ?
 - ▶ Nous avons vu dans le cours 2 un algorithme en $O(\sqrt{N})$
 - ▶ Nous pourrions aller plus vite si nous connaissions tous les nombres premiers plus petits que N .

- ▶ Comment savoir si un nombre est premier ?
 - ▶ Nous avons vu dans le cours 2 un algorithme en $O(\sqrt{N})$
 - ▶ Nous pourrions aller plus vite si nous connaissions tous les nombres premiers plus petits que N .
 - ▶ Mais faire ce pré-calcul est très coûteux.

- ▶ Comment savoir si un nombre est premier ?
 - ▶ Nous avons vu dans le cours 2 un algorithme en $O(\sqrt{N})$
 - ▶ Nous pourrions aller plus vite si nous connaissions tous les nombres premiers plus petits que N .
 - ▶ Mais faire ce pré-calcul est très coûteux.
- ▶ Sauf si on cherche à calculer tous les nombres premiers inférieurs à N .

- ▶ Comment savoir si un nombre est premier ?
 - ▶ Nous avons vu dans le cours 2 un algorithme en $O(\sqrt{N})$
 - ▶ Nous pourrions aller plus vite si nous connaissions tous les nombres premiers plus petits que N .
 - ▶ Mais faire ce pré-calcul est très coûteux.
- ▶ Sauf si on cherche à calculer tous les nombres premiers inférieurs à N .
 - ▶ Il n'est alors pas nécessaire de faire N tests de primalité.

- ▶ Comment savoir si un nombre est premier ?
 - ▶ Nous avons vu dans le cours 2 un algorithme en $O(\sqrt{N})$
 - ▶ Nous pourrions aller plus vite si nous connaissions tous les nombres premiers plus petits que N .
 - ▶ Mais faire ce pré-calcul est très coûteux.
- ▶ Sauf si on cherche à calculer tous les nombres premiers inférieurs à N .
 - ▶ Il n'est alors pas nécessaire de faire N tests de primalité.
 - ▶ Il existe un algorithme connu depuis la Grèce antique !
 - ▶ Le crible d'Ératosthène

- ▶ Comment savoir si un nombre est premier ?
 - ▶ Nous avons vu dans le cours 2 un algorithme en $O(\sqrt{N})$
 - ▶ Nous pourrions aller plus vite si nous connaissions tous les nombres premiers plus petits que N .
 - ▶ Mais faire ce pré-calcul est très coûteux.
- ▶ Sauf si on cherche à calculer tous les nombres premiers inférieurs à N .
 - ▶ Il n'est alors pas nécessaire de faire N tests de primalité.
 - ▶ Il existe un algorithme connu depuis la Grèce antique !
 - ▶ Le crible d'Ératosthène
- ▶ On considère la liste de tous les nombres inférieurs à N

- ▶ Comment savoir si un nombre est premier ?
 - ▶ Nous avons vu dans le cours 2 un algorithme en $O(\sqrt{N})$
 - ▶ Nous pourrions aller plus vite si nous connaissions tous les nombres premiers plus petits que N .
 - ▶ Mais faire ce pré-calcul est très coûteux.
- ▶ Sauf si on cherche à calculer tous les nombres premiers inférieurs à N .
 - ▶ Il n'est alors pas nécessaire de faire N tests de primalité.
 - ▶ Il existe un algorithme connu depuis la Grèce antique !
 - ▶ Le crible d'Ératosthène
- ▶ On considère la liste de tous les nombres inférieurs à N
 - ▶ On entoure 2 et on barre tous ses multiples : 4, 6, etc.

- ▶ Comment savoir si un nombre est premier ?
 - ▶ Nous avons vu dans le cours 2 un algorithme en $O(\sqrt{N})$
 - ▶ Nous pourrions aller plus vite si nous connaissions tous les nombres premiers plus petits que N .
 - ▶ Mais faire ce pré-calcul est très coûteux.
- ▶ Sauf si on cherche à calculer tous les nombres premiers inférieurs à N .
 - ▶ Il n'est alors pas nécessaire de faire N tests de primalité.
 - ▶ Il existe un algorithme connu depuis la Grèce antique !
 - ▶ Le crible d'Ératosthène
- ▶ On considère la liste de tous les nombres inférieurs à N
 - ▶ On entoure 2 et on barre tous ses multiples : 4, 6, etc.
 - ▶ Le premier nombre non barré est un nombre premier (3)
 - ▶ On entoure 3 et on barre tous ses multiples : 3, (6 déjà barré), 9, etc.

- ▶ Comment savoir si un nombre est premier ?
 - ▶ Nous avons vu dans le cours 2 un algorithme en $O(\sqrt{N})$
 - ▶ Nous pourrions aller plus vite si nous connaissions tous les nombres premiers plus petits que N .
 - ▶ Mais faire ce pré-calcul est très coûteux.
- ▶ Sauf si on cherche à calculer tous les nombres premiers inférieurs à N .
 - ▶ Il n'est alors pas nécessaire de faire N tests de primalité.
 - ▶ Il existe un algorithme connu depuis la Grèce antique !
 - ▶ Le crible d'Ératosthène
- ▶ On considère la liste de tous les nombres inférieurs à N
 - ▶ On entoure 2 et on barre tous ses multiples : 4, 6, etc.
 - ▶ Le premier nombre non barré est un nombre premier (3)
 - ▶ On entoure 3 et on barre tous ses multiples : 3, (6 déjà barré), 9, etc.
 - ▶ Le premier nombre non barré est un nombre premier (5)
 - ▶ On entoure 5 et on barre tous ses multiples : ...

SCRIPT

```
# renvoie les premiers compris entre 2 et N
def crible(N):
    premier = (N+1) * [True]
    premier[0]=False
    premier[1]=False
    k=2
    while k*k <= N:
        for m in range(2*k, N+1, k):# m = 2k 3k 4k...
            premier[m]=False
        k = k+1
    return [k for k in range(n+1) if premier[k]]
```

SHELL

>>>

SCRIPT

```
# renvoie les premiers compris entre 2 et N
def crible(N):
    premier = (N+1) * [True]
    premier[0]=False
    premier[1]=False
    k=2
    while k*k <= N:
        for m in range(2*k, N+1, k):# m = 2k 3k 4k...
            premier[m]=False
        k = k+1
    return [k for k in range(n+1) if premier[k]]
```

SHELL

```
>>> crible(50)
```

SCRIPT

```

# renvoie les premiers compris entre 2 et N
def crible(N):
    premier = (N+1) * [True]
    premier[0]=False
    premier[1]=False
    k=2
    while k*k <= N:
        for m in range(2*k, N+1, k):# m = 2k 3k 4k...
            premier[m]=False
        k = k+1
    return [k for k in range(n+1) if premier[k]]
    
```

SHELL

```

>>> crible(50)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
>>>
    
```

SCRIPT

```

# renvoie les premiers compris entre 2 et N
def crible(N):
    premier = (N+1) * [True]
    premier[0]=False
    premier[1]=False
    k=2
    while k*k <= N:
        for m in range(2*k, N+1, k):# m = 2k 3k 4k...
            premier[m]=False
        k = k+1
    return [k for k in range(n+1) if premier[k]]
    
```

SHELL

```

>>> crible(50)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
>>> len(crible(1000))
    
```

SCRIPT

```

# renvoie les premiers compris entre 2 et N
def crible(N):
    premier = (N+1) * [True]
    premier[0]=False
    premier[1]=False
    k=2
    while k*k <= N:
        for m in range(2*k, N+1, k):# m = 2k 3k 4k...
            premier[m]=False
        k = k+1
    return [k for k in range(n+1) if premier[k]]
    
```

SHELL

```

>>> crible(50)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
>>> len(crible(1000))
168
    
```

SCRIPT

```

# renvoie les premiers compris entre 2 et N
def crible(N):
    premier = (N+1) * [True]
    premier[0]=False
    premier[1]=False
    k=2
    while k*k <= N:
        for m in range(2*k, N+1, k):# m = 2k 3k 4k...
            premier[m]=False
        k = k+1
    return [k for k in range(n+1) if premier[k]]
    
```

SHELL

```

>>> crible(50)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
>>> len(crible(1000))
168
    
```

- Pour trouver les nombres premiers entre 2 et N, on peut

SCRIPT

```

# renvoie les premiers compris entre 2 et N
def crible(N):
    premier = (N+1) * [True]
    premier[0]=False
    premier[1]=False
    k=2
    while k*k <= N:
        for m in range(2*k, N+1, k):# m = 2k 3k 4k...
            premier[m]=False
        k = k+1
    return [k for k in range(n+1) if premier[k]]
    
```

SHELL

```

>>> crible(50)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
>>> len(crible(1000))
168
    
```

- ▶ Pour trouver les nombres premiers entre 2 et N, on peut
 - ▶ tester si chacun de ces nombres est premier (cours 2) : coût en $O(N\sqrt{N})$;

SCRIPT

```

# renvoie les premiers compris entre 2 et N
def crible(N):
    premier = (N+1) * [True]
    premier[0]=False
    premier[1]=False
    k=2
    while k*k <= N:
        for m in range(2*k, N+1, k):# m = 2k 3k 4k...
            premier[m]=False
        k = k+1
    return [k for k in range(n+1) if premier[k]]
    
```

SHELL

```

>>> crible(50)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
>>> len(crible(1000))
168
    
```

- ▶ Pour trouver les nombres premiers entre 2 et N , on peut
 - ▶ tester si chacun de ces nombres est premier (cours 2) : coût en $O(N\sqrt{N})$;
 - ▶ utiliser le crible d'Ératosthène : coût en $O(N \log \log N)$.

SCRIPT

```
# renvoie les premiers compris entre 2 et N
def crible(N):
    premier = (N+1) * [True]
    premier[0]=False
    premier[1]=False
    k=2
    while k*k <= N:
        for m in range(2*k, N+1, k):# m = 2k 3k 4k...
            premier[m]=False
        k = k+1
    return [k for k in range(n+1) if premier[k]]
```

SHELL

```
>>> crible(50)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
>>> len(crible(1000))
168
```

- ▶ Pour trouver les nombres premiers entre 2 et N , on peut
 - ▶ tester si chacun de ces nombres est premier (cours 2) : coût en $O(N\sqrt{N})$;
 - ▶ utiliser le crible d'Ératosthène : coût en $O(N \log \log N)$.
 - ▶ Mais ce n'est pas si facile de prouver cette complexité

 Partie I. Programmation graphique

 Partie II. Algorithmes

 Partie III. Algorithmes de tri

 Partie IV. Complexité

 Partie V. Algorithmes de recherche

 Partie VI. Crible d'Ératosthène

 **Partie VII. Bilan**

 Partie VIII. Table des matières

- ▶ Les algorithmes sont des mélanges d'astuces élégantes et de grands principes universels
 - ▶ diviser pour régner, programmation dynamique, algorithmes gloutons,...

- ▶ Les algorithmes sont des mélanges d'astuces élégantes et de grands principes universels
 - ▶ diviser pour régner, programmation dynamique, algorithmes gloutons,...
- ▶ Même des algorithmes relativement simples (ex : Quicksort) font l'objet de recherches mathématiques poussées encore aujourd'hui, en particulier en combinatoire.

- ▶ Les algorithmes sont des mélanges d'astuces élégantes et de grands principes universels
 - ▶ diviser pour régner, programmation dynamique, algorithmes gloutons,...
- ▶ Même des algorithmes relativement simples (ex : Quicksort) font l'objet de recherches mathématiques poussées encore aujourd'hui, en particulier en combinatoire.
- ▶ C'est un des nombreux exemples où l'informatique théorique et les mathématiques se rejoignent.

- ▶ Les algorithmes sont des mélanges d'astuces élégantes et de grands principes universels
 - ▶ diviser pour régner, programmation dynamique, algorithmes gloutons,...
- ▶ Même des algorithmes relativement simples (ex : Quicksort) font l'objet de recherches mathématiques poussées encore aujourd'hui, en particulier en combinatoire.
- ▶ C'est un des nombreux exemples où l'informatique théorique et les mathématiques se rejoignent.
- ▶ En TD, vous verrez d'autres algorithmes proches de ceux de ce cours,

- ▶ Les algorithmes sont des mélanges d'astuces élégantes et de grands principes universels
 - ▶ diviser pour régner, programmation dynamique, algorithmes gloutons,...
- ▶ Même des algorithmes relativement simples (ex : Quicksort) font l'objet de recherches mathématiques poussées encore aujourd'hui, en particulier en combinatoire.
- ▶ C'est un des nombreux exemples où l'informatique théorique et les mathématiques se rejoignent.
- ▶ En TD, vous verrez d'autres algorithmes proches de ceux de ce cours,
- ▶ Mais pour vraiment approfondir le domaine, **vous pourrez suivre les cours d'algorithmique de L2 et L3.**

- ▶ Les algorithmes sont des mélanges d'astuces élégantes et de grands principes universels
 - ▶ diviser pour régner, programmation dynamique, algorithmes gloutons,...
- ▶ Même des algorithmes relativement simples (ex : Quicksort) font l'objet de recherches mathématiques poussées encore aujourd'hui, en particulier en combinatoire.
- ▶ C'est un des nombreux exemples où l'informatique théorique et les mathématiques se rejoignent.
- ▶ En TD, vous verrez d'autres algorithmes proches de ceux de ce cours,
- ▶ Mais pour vraiment approfondir le domaine, **vous pourrez suivre les cours d'algorithmique de L2 et L3.**
- ▶ En attendant il y a toujours l'**excellent** «Cormen» :
 - ▶ Introduction à l'algorithmique, *T. Cormen, C. Leiserson, R. Rivest, C. Stein*
 - ▶ Disponible en BU et dans toutes les bonnes librairies.

Merci pour votre attention

Questions



Cours 9 — Animations et algorithmes

🍃 Partie I. Programmation graphique

Objectif

Programmation évènementielle

Évolution de l'état

Définir l'état

Ajouter des boutons

Le code (presque) complet

Autres évènements

Exemple

Faire une animation

Gérer le temps

Révolutions!

Révolutions : le code

Conclusion

🍃 Partie II. Algorithmes

Qu'est-ce qu'un algorithme?

Historique

Complexité des algorithmes

Notations de Landau

🍃 Partie III. Algorithmes de tri

Tri fusion, principe

Tri fusion, implémentation

Tri rapide, principe

Tri rapide, exemple

Tri rapide, implémentation

🍃 Partie IV. Complexité

Comparaison empirique des algorithmes

Comparaison empirique des algorithmes

Comparaison théorique des algorithmes

Analyse du tri fusion

Complexité des algorithmes de tri

🍃 Partie V. Algorithmes de recherche

Recherche séquentielle dans une liste

Recherche dichotomique, principe

Recherche dichotomique, code

Recherche dichotomique, exemple

Notion de coût amorti, principe

🍃 Partie VI. Crible d'Ératosthène

Principe

Implémentation

🍃 Partie VII. Bilan

Algorithmes

Dernier TP : créez votre propre python

🍃 Partie VIII. Table des matières