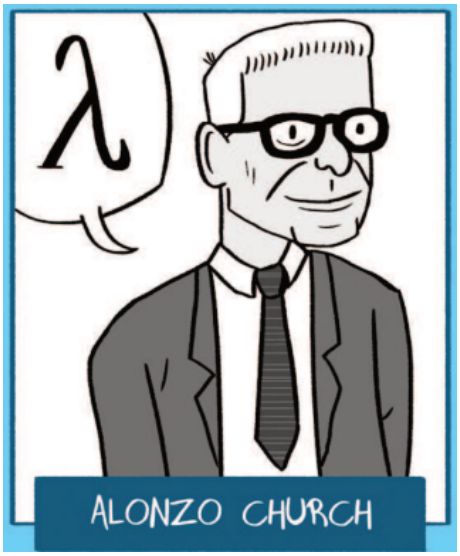

Alonzo Church Il est connu principalement pour le développement du lambda-calcul, son application à la notion de fonction récursive, pour la première démonstration de l'indécidabilité de l'arrêt.

Les travaux de son équipe (Kleene et Rosser) précèdent, sur le problème de l'arrêt, le travail d'Alan Turing, qui va d'ailleurs les rejoindre. C'est Church qui, le premier, a l'idée que l'on peut définir le concept de fonction calculable dans un sens très large. Church en a eu l'idée par le lambda-calcul. **Church démontre en 1936 l'existence d'un problème insoluble par algorithme, autrement dit qui ne peut pas être résolu par un calcul mécanisable**. (`https://fr.wikipedia.org/wiki/Alonzo_Church`)

Le but de ce TP est de tracer un graphique représentant l'évolution du temps d'exécution des algorithmes de tris en fonction de la taille de la liste.

0.1 Implémentation

1. Taper le code des fonctions `tris_insertion` et `tri_selection`. Inclure la **docstring**.

```
def tri_insertion(tab):
    n = len(tab)
    for i in range(1, n):
        elem = tab[i]
        k = i
        while k>= 1 and tab[k-1]>elem:
            tab[k] = tab[k-1]
            k = k-1
        tab[k] = elem
```

```
def tri_selection(tab):
    n = len(tab)
    for j in range(0, n-1):
        ind_mini= j
        for i in range(j+1, n):
            if tab[i] < tab[ind_mini]:
                ind_mini = i
        tmp = tab[j]
        tab[j] = tab[ind_mini]
        tab[ind_mini] = tmp
```

2. Effectuer un jeu de test avec des listes générées aléatoirement.

0.2 Mesure en temps de la complexité

Pour mesurer et représenter le temps d'exécution des algorithmes étudiés, nous aurons besoin des modules suivants:

```
import matplotlib.pyplot as plt
import time
import random
```

0.2.1 Afficher un graphique

Il suffit d'utiliser deux listes: une pour les abscisses et une autre pour les ordonnées. En voici un exemple d'utilisation minimaliste qui affiche une représentation de la fonction *cube*:

```
import matplotlib.pyplot as plt
abscisse = list(range(0, 10))
cube = [t**3 for t in abscisse]
```

```
plt.plot(abscisse, cube)
plt.show()
plt.close()
```

0.2.2 Chronométrer

Une méthode du module `time` pour récupérer le temps en seconde est `perf_counter()` ou `time()` qui donne un temps dont le point de référence de la valeur renvoyée est indéfini, de sorte que *seule la différence entre les résultats d'appels consécutifs est valable.*

1. Effectuer un chronométrage du tri par sélection pour une liste de 1000 entiers.
2. Effectuer une série de mesure pour des listes dont la taille varie de 100 à 2000 avec un pas de 100 et récupérer:
   (a) Les mesures du temps d'execution de chaque tri par sélection dans une liste `chrono`.
   (b) Les tailles de liste correspondantes dans une liste `taille_liste`.
3. A l'aide du module `pyplot`, afficher le graphique des temps d'éxecution en fonction de la taille de la liste.
4. Effectuer la même opérations pour le tri par insertion et la méthode *built-in* `sorted()`.

Indication Pour représenter le graphique associé avec la taille de la liste en abscisse et le temps d'execution en ordonnée, utiliser les méthodes `plot(liste_x,liste_y)` pour tracer le graphique et `show()` pour afficher, ce qui donne avec l'*alias* créé plus haut:

```
plt.plot(taille_liste, chrono)
plt.show()
plt.close()
```

Exemple Compléter le code suivant:

```
def liste_alea(n):
    L = list(range(0, n)
    random.shuffle(L)
    return L

chrono = []
taille_liste = []
for taille in range(100, 2000, ..........):
    taille_liste.append(..........)
    L = liste_alea(taille)
    debut = ..........
    sorted(L)
    fin = ..........
    temps = fin - debut
    chrono.append(..........)


plt.plot(taille_liste, chrono)
plt.show()
plt.close()
```