# Les tris

Trier des données est une opération de base présente dans tout traitement. Disposer d'algorithmes de tri efficaces à l'heure du Big Data est devenu crucial. Le cours de NSI commence par introduire les tris classiques. Ce ne sont pas les plus performants mais ils sont simples et comprendre leur mécanisme permet d'apprendre à construire des algorithmes plus complexes. Par ordre d'efficacité : tri par sélection (selection sort), tri à bulles (bubble sort), tri par insertion (insertion sort), tri fusion (merge sort), tri par tas (heapsort), tri rapide (quicksort). Insertion est très rapide sur des données de petite taille. Rapide est considéré comme le meilleur algorithme sur des données de grande taille.

## I. Tri par sélection

C'est un tri sur place, c'est-à-dire que la liste non triée est modifiée au fur et à mesure du tri lors de la mise en ordre. Il n'y a donc qu'une seule liste en mémoire de l'ordinateur.

Un algorithme possible est le suivant.

```
Pour i allant de 0 à n-2
    indice_min = i

    Pour j allant de i+1 à n
        Si liste[j] < liste[indice_min]
            indice_min = j

    échanger liste[i] et liste[indice_min]
```

1. Coder l'algorithme en Python.
   Puis le tester sur une liste contenant 30 nombres aléatoires qu'on pourra créer de cette manière:
   `l = [random.randint(0,100) for _ in range(20)]` grâce au module `random`.
2. Dans votre fonction, ajouter un compteur en début d'éxecution et incrémenter le après chaque opération de comparaison effectuée.
3. Tester votre algorithme plusieurs fois pour différentes longueurs de liste. Par exemple `len(l) == 20`, `40`, `60`, `80`, `100`, `200`et `500`.
4. Tracer sur votre feuille les points correspondants au nombres obtenus puis tracer une courbe suivant l'allure donnée par ces points.

## II. Tri par insertion

C'est aussi un tri sur place.&

Un algorithme possible est le suivant.

```
Pour i allant de 1 à n-1
    clé = liste[i]
    j = i - 1

    Tant que j >= 0 et liste[j] > clé
        liste[j+1] = liste[j]
        j = j - 1

    liste[j+1] = clé
```

1. Coder l'algorithme en Python.
   Puis le tester sur une liste contenant 30 nombres aléatoires qu'on pourra créer de cette manière:
   `l = [random.randint(0,100) for _ in range(20)]` grâce au module `random`.
2. Dans votre fonction, ajouter un compteur en début d'éxecution et incrémenter le après chaque opération de comparaison effectuée.
3. Tester votre algorithme plusieurs fois pour différentes longueurs de liste. Par exemple `len(l) == 20`, `40`, `60`, `80`, `100`, `200`et `500`.
4. Tracer sur votre feuille les points correspondants au nombres obtenus puis tracer une courbe suivant l'allure donnée par ces points.