**Session d'exercices - Recherche dichotomique**

Maintenant que nous avons vu comment trier nos listes, exploitons les propriétés d'une liste triée pour vérifier efficacement si elle contient un élément donné et, si oui, pour en trouver la position.

La recherche dichotomique, ou *binary search*, fonctionne sur une liste triée (ici en ordre croissant) et un élément cible, noté $e$. On commence par examiner l'élément central de la liste, noté $m$ :

- Si $m = e$, nous avons trouvé $e$ avec succès à la position de $m$.
- Sinon, il y a deux cas :
  - Si $e < m$, on continue la recherche dans la moitié inférieure de la liste, soit de l'index de départ jusqu'à l'élément précédant $m$.
  - Si $e > m$, on continue la recherche dans la moitié supérieure de la liste, de l'élément suivant $m$ jusqu'à la fin de la liste.

La recherche continue de manière identique dans la moitié pertinente, en prenant à chaque étape l'élément central de la sous-liste. Ce processus se répète jusqu'à ce que l'élément soit trouvé, ou jusqu'à ce que la recherche se limite à une sous-liste vide, indiquant alors que $e$ n'est pas présent dans la liste.

**À noter** : Cette méthode ne fonctionne que si la liste est triée en ordre croissant.

## Exercice : Coder une fonction `binary_search()`

[Difficulté: **]

1. Créez une liste `numbers`, triez-la, et affichez-la à l'écran.
2. Définissez et codez la fonction `binary_search()`, qui prend en paramètres :
   - une liste d'entiers `values`
   - un entier `item`

   La fonction retourne l'index de `item` si celui-ci est présent dans `values` (par recherche dichotomique) ou `-1` si ce n'est pas le cas.
3. Vérifiez la fonction en recherchant chaque élément de `numbers` pour confirmer son bon fonctionnement :

```
for v in numbers:
    print(binary_search(numbers, v))
```

4. Testez votre fonction avec des éléments absents de la liste ; elle doit retourner `-1`.