TP3 : Algorithmes de tri et de recherche

Enseignants	Pr. TAHA
Tags	TP, Tri et recherche
Année universitaire	2024-2025
Section	MIP_GI_S2

Objectifs :

1. Implémenter et comparer différents algorithmes de tri (bubble, insertion, merge, quick).
2. Comprendre la récursion et le « divide & conquer » à travers merge sort et quick sort.
3. Mettre en œuvre la recherche linéaire et dichotomique et mesurer leurs performances.

Introduction :

Le tri et la recherche sont des primitives centrales en algorithmique. Leur maîtrise est essentielle pour tout développement performant.

À savoir pour TP3:

1. `import time` **:** Permet de mesurer la durée d'exécution d'un bloc de code.

```
start = time.time()
# ... appel à votre fonction de tri ...
end = time.time()
print(f"Durée : {end - start:.5f} secondes")
```

2. `import matplotlib.pyplot as plt` **:** permet de tracer graphiquement les résultats (courbes de temps, comparaisons).

```
plt.plot([100, 500, 1000], [t1, t2, t3])
plt.xlabel("Taille de la liste")
plt.ylabel("Temps (s)")
plt.title("Comparaison des tris")
plt.show()
```

3. `import random` **:** permet de générer des données aléatoires pour tester vos algorithmes.

```
L = [i for i in range(10)]
random.shuffle(L)        # mélange la liste en place
```

```
x = random.randint(1, 100)  # entier aléatoire entre 1 et 100
```

Exercices :

1. **Tri à bulles et tri par insertion**
   - Implémenter `bubble_sort(L)` et `insertion_sort(L)`.
   - Tester sur des listes aléatoires de taille 100, 500, 1000 ; mesurer les temps d'exécution.
2. **Merge sort (tri fusion)**
   - Implémenter `merge_sort(L)` de manière récursive.
   - Tracer l'arbre d'appels pour `len(L) = 6`.
3. **Quick sort**
   - Implémenter `quick_sort(L)` en choisissant le pivot au milieu.
   - Expérimenter sur une liste quasi triée et un liste inversée : quel impact sur la complexité ?
4. **Comparaison des tris**
   - Pour `n` variant de 1000 à 10 000, tracer les temps d'exécution des quatre tris sur la **même liste non triée**.
   - Interpréter les résultats par rapport aux complexités théoriques.
5. **Recherche linéaire vs dichotomique**
   - Implémenter `linear_search(L, x)` et `binary_search(L, x)` (sur une liste triée).
   - Pour `n` de 10^3 à 10^5, mesurez le temps moyen de recherche d'un élément présent et d'un élément absent.

Questions de réflexion :

- Dans quels cas Quick Sort peut-il devenir inefficace ? Comment y remédier ?
- Pourquoi Merge Sort garantit-il toujours O(n log n) alors que Quick Sort non ?
- Quelle méthode de recherche choisiriez vous pour une liste extrêmement volumineuse ? Justifiez.