ITC – TD n°7

Benchmark 1 – recherche dichotomique

On souhaite dans ce TD tester et comparer la recherche d'un élément dans un tableau par la méthode dichotomique ou par la méthode séquentielle.

On démarrera le code par les imports suivants :

```
from random import randint
from matplotlib import pyplot as plt
from time import perf_counter
from numpy import array  # explication dans l'encadré ci-dessous
```

> Pour des raisons que nous expliquerons dans le prochain cours, si vous souhaitez faire une version récursive de la recherche dichotomique, alors les listes Python ont un comportement peu pratique. Nous pouvons contourner cette difficulté en utilisant des tableaux Numpy à la place. Dans ce cas, pour **tous** vos tableaux, créez des listes et convertissez-les : avec `tab = array([0, 1, 4, 5, 7, 8])`, la liste fournie est convertie en array par la fonction array.

1 – Écrire une fonction `créer_tableau_trié` qui prend en paramètre deux entiers n et $v_{\max}$ et qui génère et renvoie un tableau (NumPy) **trié** de taille n contenant des entiers tirés aléatoirement entre 0 et $v_{\max}$. On utilisera pour tirer un nombre aléatoire `randint(0, v_max)` et on pourra utiliser la fonction `sort` de Python en regardant sa documentation.

2 – Écrire une fonction `indice_séquentiel` qui prend en paramètre un tableau `tab` et une valeur x, et renvoie l'indice de la première occurrence de x dans le tableau si $x \in$ tab et renvoie -1 sinon. La complexité doit être $\mathcal{O}(n)$.

3 – Écrire une fonction `indice_dichotomie` qui prend en paramètre un tableau **trié** `tab` et une valeur x, et renvoie l'indice de la première occurrence de x dans le tableau si $x \in$ tab et renvoie -1 sinon. La complexité doit être $\mathcal{O}(\log_2 n)$.
Remarque : vous pouvez écrire cette fonction de façon impérative ou récursive, ou idéalement, les deux.

4 – Proposer quelques tests pour ces fonctions. En écrire au moins deux dans une procédure `tests_recherche_indice` qui prend la fonction de recherche d'élément en paramètre. Vérifier que vos fonctions passent les tests.
Remarque : un jeu de tests vous sera fourni sur cahier de prépa après quelques jours.

On cherche à présent à comparer la performance des deux algorithmes. On procède comme suit :

- pour une taille n donnée, on génère un tableau aléatoire trié de taille n, avec des entiers compris entre 0 et $2n$;
- on choisit un nombre aléatoire x entre 0 et `int(2.2*n)` ;
- on mesure le temps d'exécution avec les instructions suivantes :

```
start = perf_counter()          # lance le chrono
res = fonction_test(tab, x)     # la fonction s'exécute
perf = perf_counter() - start   # arrête le chrono
```

après ces instructions, **le temps d'exécution (en secondes) de la ligne 2 est stockée dans la variable** `perf`

- on fait `nb_essais` fois ce processus, et on effectue la moyenne des performances.

5 – Écrire une fonction

```
benchmark_indices_taille_fixée(n:int, nb_essais: int, ftest: 'fonction') -> float
```

qui prend en paramètres une taille n, un nombre d'essais et une fonction à benchmarker, et applique la procédure ci-dessus pour renvoyer son temps d'exécution moyen.

6 – Écrire une fonction

```
benchmark_indices(tailles: list, nb_essais: int, ftest: 'fonction') -> list
```

qui prend en paramètres une liste de tailles, un nombre d'essais et une fonction à benchmarker, et renvoie une liste contenant les temps moyens pour chaque taille demandée en paramètre.

7 – Exécuter le code suivant :

```
tailles = [2*i for i in range(1, 50)]
lin = benchmark_indice(tailles, 1000, indice_séquentiel)
dic = benchmark_indice(tailles, 1000, indice_dichotomie)

plt.scatter(tailles, lin, color='r', label='Linéaire')
plt.scatter(tailles, dic, color='b', label='Dichotomique')
plt.ylim(0, max(lin))
plt.xlabel('n')
plt.ylabel('temps')
plt.legend()
```

et commenter le résultat.