

## 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
```

**1** – Écrire une fonction `créer_tableau_trié` qui prend en paramètre deux entiers  $n$  et  $v_{\max}$  et qui génère et renvoie un tableau (liste Python) **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  $x \in \text{tab}$  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  $x \in \text{tab}$  et renvoie  $-1$  sinon. La complexité doit être  $\mathcal{O}(\log_2 n)$ .

**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.

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()
res = fonction_test(tab, x)
perf = perf_counter() - start
```

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

**5** – Écrire une fonction `benchmark_indices_taille_fixée` 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` 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')
plt.scatter(tailles, dic, color='b')
plt.ylim(0, max(lin))
```

et commenter le résultat.