Corrigé – Algorithmes de recherche

Exercice 1

```
1  def recherche(lst, val):
2      indices = []
3      for ind in range(len(lst)):
4          if lst[ind] == val:
5              indices.append(ind)
6      return indices
```

Nous n'utilisons à partir d'ici QUE DES TABLEAUX DE DONNÉES TRIÉES !

/!\ /!\ /!\ Soit le tableau suivant, **valable pour tout ce qui suit** : /!\ /!\ /!\

```
lst = [1, 4, 10, 14, 21, 30, 76, 78, 99]
```

Exercice 2

1. Il faut **4 tours** de boucle pour trouver la valeur 14 dans la liste `lst` ;
2. Il faut **9 tours** de boucle pour constater qu'on ne trouve pas la valeur 208 dans `lst` (soit la taille de la liste en entrée).
3. Concernant la complexité, on est donc dans le **pire des cas** car c'est le maximum d'itérations qu'on puisse faire avant d'avoir une réponse.
4. Il faut chercher le premier élément du tableau. Ici, il vaut 1.

Exercice 3

```
fonction recherche_sequentielle(tableau t, valeur cible):
    debut ← indice du premier element de t
    fin ← indice du dernier element de t
    pour indice variant de debut → fin:
        si t[indice] vaut cible :
            renvoyer indice
    renvoyer −1
```

La fonction `recherche_sequentielle` *prend en argument un tableau et une valeur cible, et renvoie l'indice de la valeur cible dans le tableau, -1 sinon.*

Exercice 4

1. (a)
```
fonction recherche_max(tableau t):
    maxi ← 0
    pour chaque element el de t:
        si el > maxi :
            maxi ← el
    renvoyer maxi
```

   (b) Coût linéaire, en $O(n)$

2. (a)
```
fonction calcule_moyenne(tableau t):
    total ← 0
    nb ← nombre d'elements de t
    pour chaque element el de t:
        total ← total + el
    renvoyer total / nb
```

   (b) Coût linéaire, en $O(n)$

Exercice 5

On cherche la valeur 10 dans `lst` (définie précédemment, en tête de section) en appliquant l'algorithme de recherche dichotomique.

1.

indices	0	1	2	3	4	5	6	7	8
valeurs	1	4	10	14	21	30	76	78	99
tour 1	g				m				d
tour 2	**g**	m		**d**					
tour 3			**m g**	**d**					

2. Il y a 9 éléments dans la liste.
3. On trouve en 3 étapes.
4. 2
5. En 3 étapes aussi.

Exercice 6

On cherche la valeur 45 dans `lst` (définie précédemment, en tête de section) en appliquant l'algorithme de recherche dichotomique.

1. Compléter le tableau du déroulement de l'algorithme :

indices	0	1	2	3	4	5	6	7	8
valeurs	1	4	10	14	21	30	76	78	99
tour 1	**g**				**m**				**d**
tour 2						**g**	**m**		**d**
tour 3						**g m d**			

2. Il y a 9 éléments dans la liste.
3. On trouve en 4 étapes.
4. -1
5. En utilisant l'algorithme de recherche séquentielle, on aurait trouvé en 9 étapes. Comme l'élément n'est pas présent dans la liste, on aurait été obligé de tout regarder avant de pouvoir l'affirmer.

Exercice 7

- `t = [2, 5, 7, 8, 12, 16, 18, 20, 25, 30, 32]` et `cible = 16`.

Tours de boucle	g	d	$g <= d$	m	t$[m]$
1	0	10	True	5	16

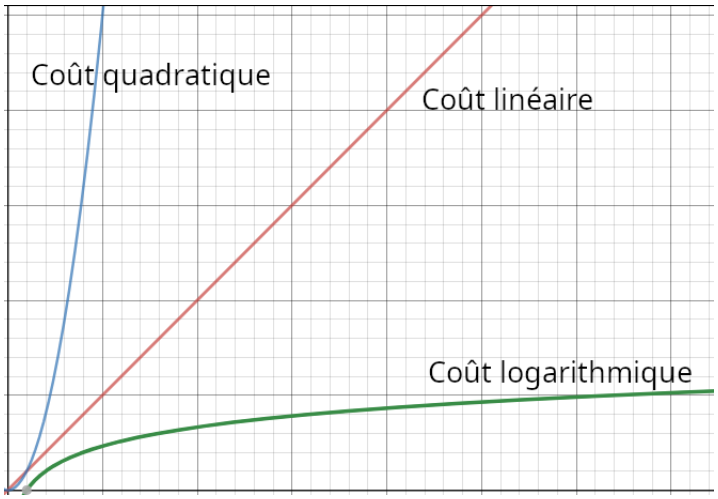
- `t = [2, 5, 7, 8, 12, 16, 18, 20, 25, 30, 32]` et `cible = 12`.

Tours de boucle	g	d	$g <= d$	m	t$[m]$
1	0	10	True	5	16
2	0	4	True	2	7
3	3	4	True	3	8
4	4	4	True	4	12

- `t = [2, 5, 7, 8, 12, 16, 18, 20, 25, 30, 32]` et `cible = 19`.

Tours de boucle	g	d	$g <= d$	m	t$[m]$
1	0	10	True	5	16
2	6	10	True	8	25
3	6	7	True	6	18
4	7	7	True	7	20
5	7	6	False		

Exercice 8



Notons que **plus la liste en entrée est grande, plus l'algorithme de recherche dichotomique est efficace par rapport à la recherche séquentielle**.

Exercice 9

1. Télécharger le fichier `dicho_seq_display.py` sur notre site.
2. Programmer la fonction `recherche_sequentielle` en Python. **Les tests ne doivent pas êtres modifiés et doivent fonctionner !**
3. Programmer la fonction `recherche_dichotomique` en Python. **Les tests ne doivent pas êtres modifiés et doivent fonctionner !**
4. Lorsque les tests passent, décommenter la dernière ligne `plt.show()`, exécuter à nouveau le programme et observer le résultat. Que peut-on conclure ?