Exercice 1 *Carrés*

Correction. Il y a deux erreurs.

Première erreur : il faudrait écrire `range(N+1)` pour réelleement parcour les entiers de 0 à N.

Seconde erreur : l'identation de la commande `return(L)` est mauvaise, un tel algorithme ferait sortir de la boucle dès le premier passage pour $i = 0$ et renverrait $[0]$. Il suffit de le mettre au niveau du for pour avoir un algorithme fonctionnnel.

Exercice 2 *Algorithmes de contrôle*

Correction.

1. (a) Si la liste est triée, tous les tests vont échouer, et à la sortie de la boucle for, le résultat `res=True` sera renvoyé, ce qui est correct.

   Si la liste n'est pas triée, au moins un test va échouer, et à la sortie de la boucle for, le résultat `res=False` sera renvoyé, ce qui est correct.

   On considère l'algorithme suivant :

```
def testi(X):
    n=len(X)
    res=True
    for i in range(n-1):
        if X[i]>X[i+1]:
            res=False
    return(res)
```

   (b) La complexité $\boxed{A_N \text{ est égale à } N-1}$, car pour une liste de longueur N on effectue exactement $N-1$ comparaisons, une par chaque tour dans la boucle for.

2. (a) `M=N//2` est le quotient dans al division euclidienne de N par 2, et vaut $N/2$ si N est pair, $(N-1)/2$ sinon.

   `X[0:N//2]` est la sous-liste $[x_0, \ldots, x_{M-1}]$, `X[N//2:]` est la sous-liste $[x_M, \ldots, x_{N-1}]$.

   (b)
```
def testr(X):
    if len(X)<=1:
        return(True)
    else:
        M=N//2
        if testr(X[0:M]) and testr(X[N//2:]):
            return(X[M]<=X[M+1])
        else:
            return(False)
```

   Si la liste X est de longueur 0 ou 1, elle est triée.

   Sinon, on teste si les sous-listes `X[0:N//2]` et `X[N//2:]` sont triées, et si tel est le cas, on vérifie si `X[M]<=X[M+1]`.

   Sinon, le résultat est Faux, car soit l'une des sous-listes n'est pas triée, soit les éléments $X[M]$ et $X[M+1]$ ne sont pas dans le bon ordre.

   (c) Il va y avoir p séries d'appels récursifs pour atteindre des cas de terminaison (avec des listes de longueur 1), soit au total (sans compter l'appel initial et els cas terminaux) $2^1 + 2^2 + \cdots + 2^{p-1} = \dfrac{2^p - 2}{2-1} = \boxed{2^p - 2}$ appels récursifs avant d'attendre des cas terminaux. Une nouvelle compraison étant effectuée à chaque appel récursif, cela fait $\boxed{B_N = N - 2}$

3. On ne gagne qu'une comparaison, dans le cas récursif, par rapport au premier algorithme.

Exercice 3 *Euler vectoriel*

Correction.

1. `Y0` représente la position (condition) initiale, à l'instant `t0`.
2. `dt` représente l'écart temporel (pas) entre deux instants discrétisés t_i et $t_{i+1} = t_i + dt$.
3. Il y aura autant d'itérations que de passages dans la boucle `while`, soit $\dfrac{tmax - t0}{dt} = 10/(10^{-3}) = 10^4$ itérations, correspondant à des instants $t_i = i \times dt$ pour i allant de 1 à 10^4 et le dernier passage dans la boucle, qui provoque la sortie de la boucle while lors du test $t_{10^4} < tmax$.
4. Cette méthode est appelée $\boxed{\text{méthode d'Euler}}$ vectorielle, à un pas.
5. Comme $x' = v$ et $v' = x'' = -\sin(t)x$, on obtient $(x, v)' = (v, -\sin(t)x)$, d'où :

```
def f_harmonique(t,Y):
   x,v = Y
   return [v,-np.sin(t)*x]
```

6. On appelerait la fonction de la manière suivante :

```
tps,sol = Solut_approch(f_harmonique,[1,0])
pos= sol[:,0]
vit = sol[:,1]
```

`pos` contient la liste des valeurs approchées des $x(t_i)$ pour i allant de 0 à $\left\lfloor \dfrac{tmax - t0}{dt} \right\rfloor$.

`vit` contient la liste des valeurs approchées des $x'(t_i)$ pour i allant de 0 à $\left\lfloor \dfrac{tmax - t0}{dt} \right\rfloor$.