Tous les modules utilisés dans le TP sont importés via les instructions suivantes :

```
import random as rd
import matplotlib.pyplot as plt
import time as t
import sys
```

1 Rappel sur la récursivité

Question 1. On utilise la relation : $\forall n \in \mathbb{N}^*,\ \sum_{k=0}^{n} k^p = n^p + \sum_{k=0}^{n-1} k^p$.

```
def somme(n,p):
    if n==0:
        return 0
    else:
        return n**p + somme(n-1,p)
```

Question 2. Une chaine de moins de deux caractères est toujours un palindrome. Si elle a au moins deux caractères, c'est un palindrome si et seulement si ses deux caractères extrémaux sont identiques et la sous-chaine privés de ses caractères extrémaux est un palindrome.

```
def palindrome(seq):
    if len(seq) <= 1:
        return True
    else:
        return seq[0] == seq[-1] and palindrome(seq[1:-1])
```

2 Exemples de tris dits *rapides*

Question 3.

```
def generateur(n):
    return [rd.randint(1,n) for k in range(n)]
```

2.1 Le tri fusion (*mergesort*)

Question 4. Les cas de base de la fonction récursive ci-dessous correspondent aux cas où l'une des deux listes passées en argument est vide.

```
def fusion(lst1, lst2):
    if lst1 == []:
        return lst2
    elif lst2 == []:
        return lst1
    else:
        x1, x2 = lst1[0], lst2[0]
        if x1 < x2:
            return [x1] + fusion(lst1[1:],lst2)
        else:
            return [x2] + fusion(lst1,lst2[1:])
```

Question 5.

```
def tri_fusion(lst):
    n = len(lst)
    if n <= 1:
        return lst
    else:
        lst1 = tri_fusion(lst[:n//2])
        lst2 = tri_fusion(lst[n//2:])
        return fusion(lst1,lst2)

def tri_fusion2(lst):
    n = len(lst)
    if n <= 1:
        return lst
    else:
        lst1 = tri_fusion2(lst[:n//2])
        lst2 = tri_fusion2(lst[n//2:])
        return fusion2(lst1,lst2)
```

Question 6.

```
sys.setrecursionlimit(40000)

for k in [3,4]:
    n = 10**k
    print("n=",n)
    liste = generateur(n)
    t0 = t.time()
    tri_fusion(liste)
    t1 = t.time()
    print(t1 - t0)

for k in [3,4]:
    n = 10**k
    liste = generateur(n)
    t0 = t.time()
    tri_fusion2(liste)
    t1 = t.time()
    print(t1 - t0)
```

La fonction `tri_fusion2` trie plus rapidement que la fonction `tri_fusion` car la méthode de fusion est moins dispendieuse en temps : il n'y a aucun appel récursif limitant donc la quantité de mémoire utilisée.

Question 7.

```
def partition(lst):
    pivot = lst[0]
    inf, sup = [], []
    for x in lst[1:]:
        if x < pivot:
            inf.append(x)
        else:
            sup.append(x)
    return pivot,inf,sup
```

Question 8.

```
def tri_rapide(lst):
    if len(lst) <= 1:
        return lst
    else:
        pivot,inf,sup = partition(lst)
        l1 = tri_rapide(inf)
        l2 = tri_rapide(sup)
        return l1 + [pivot] + l2
```

Question 9.

```
for k in range(3,7):
    n = 10**k
    print("n=",n)
    liste = generateur(n)
    t0 = t.time()
    tri_rapide(liste)
    t1 = t.time()
    print(t1 - t0)
```

3 Principaux tris naïfs (*rappels de sup*)

Question 10.

```
def tri_selection(liste):
    n = len(liste)
    for i in range(n-1):
        j_min = i
        for j in range(i+1,n):
            if liste[j] < liste[j_min]:
                j_min = j
        liste[i], liste[j_min] = liste[j_min], liste[i]
```

Question 11.

```
for k in range(2,5):
    n = 10**k
    print("n=",n)
    liste = generateur(n)
    t0 = t.time()
    tri_selection(liste)
    t1 = t.time()
    print(t1 - t0)
```

3.1 Le tri par insertion

Question 12.

```
def tri_insertion(seq):
    n = len(seq)
    for i in range(1,n):
        x = seq[i]
        j = i - 1
        while j >= 0 and seq[j] >= x :
            seq[j+1] = seq[j]
            j = j-1
        seq[j+1] = x
```

Question 13.

```
for k in range(2,5):
    n = 10**k
    print("n=",n)
    liste = generateur(n)
    t0 = t.time()
    tri_insertion(liste)
    t1 = t.time()
    print(t1 - t0)
```

Question 14. Le principe du tri bulle est de parcourir (n fois) la liste passée en argument et de permuter tous les termes consécutifs mal ordonnés. À la fin de l'itération i, le $(i+1)$-ème plus grand élément de la liste est "remonté" à la bonne place. Cet algorithme trie donc bien la liste passée en argument.

```
def tri_bulle(liste):
    n = len(liste)
    for i in range(n):
        for j in range(n-i-1):
            if liste[j] > liste[j+1]:
                liste[j], liste[j+1] = liste[j+1], liste[j]
```

Question 15. L'idée est de créer une variable booléenne (*drapeau*) qu'on lève dès que la liste est triée.

```
def tri_bulle(liste):
    n = len(liste)
    sorted = False
    i = 0
    while not sorted:
        sorted = True
        for j in range(n-i-1):
            if liste[j] > liste[j+1]:
                liste[j], liste[j+1] = liste[j+1], liste[j]
                sorted = False
        i += 1
```

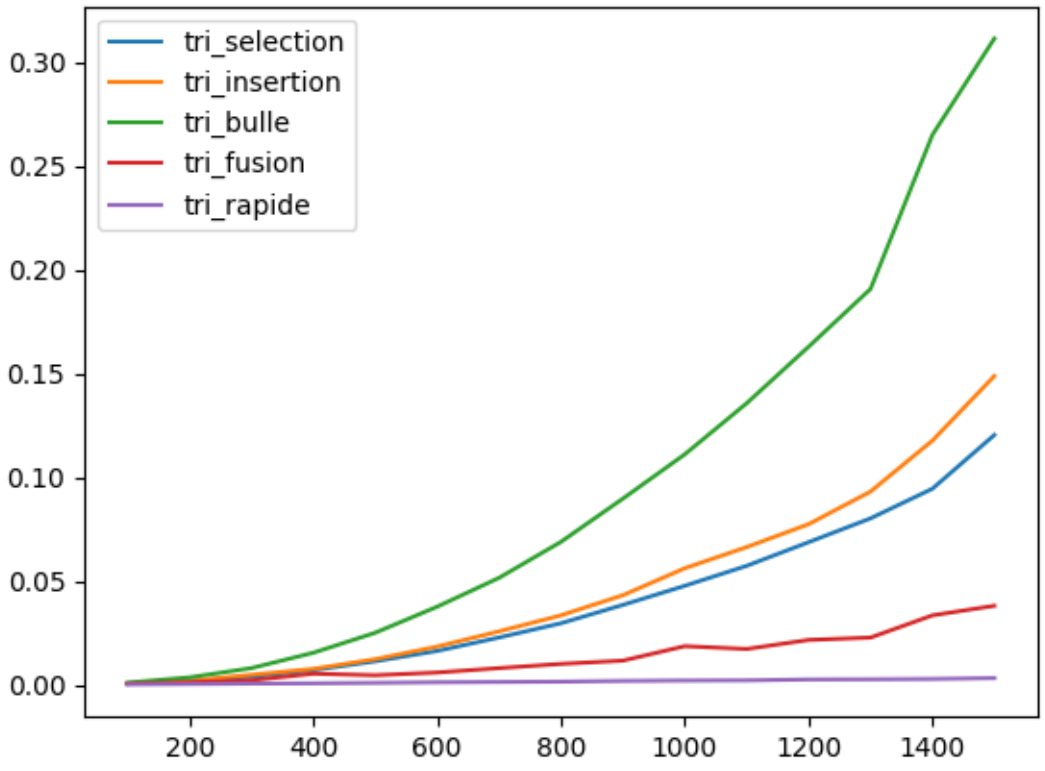
Question 16.

```
for k in range(2,5):
    n = 10**k
    print("n=",n)
    liste = generateur(n)
    t0 = t.time()
    tri_bulle(liste)
    t1 = t.time()
    print(t1 - t0)
```

4 Comparaison des performance des algorithmes

Question 17.

```
longueurs = list(range(100,1600,100))
liste_algos = [
    tri_selection,
    tri_insertion,
    tri_bulle,
    tri_fusion,
    tri_rapide
]
temps = [[] for k in range(5)]
for n in longueurs:
    N = 10
    for k in range(5):
        tps = 0
        for i in range(N):
            liste = generateur(n)
            algo = liste_algos[k]
            t0 = t.time()
            algo(liste)
            t1 = t.time()
            tps += t1-t0
        temps[k].append(tps/N)
for k in range(5):
    algo = liste_algos[k]
    plt.plot(longueurs,temps[k],label=algo.__name__)
plt.legend(loc = "best")
plt.show()
```



* *
*