Exercice 1

A

timeit est une fonction qui calcule le temps d'exécution d'une fonction qui prend en paramètre global pour avoir accès au fonction et variable du namespace ainsi que numbers qui est un paramètre qui spécifie combien de fois exécuter la fonction.

Ce qui est retourné est le temps total d'exécution de la fonction multiplier par nombre de fois où elle a été exécutée.

Cela sert à minimiser la marge d'erreur et a voir la performance sur plusieurs itérations.

B

Algo 1	Algo 2	Algo 3	Algo 4
0.03220805700038909	0.010821561999819096	0.0037342890000218176	0.4248259099999814

On remarque que le plus rapide est l'algo de complexité On suivi par les agol 1 et 2 de complexité On**2 et bien évidemment le dernier est l'algo 4 de complexité On**3

C

Le programme calcul le temps d'exécution de la fonction somme pour les nombres allant de 1 à 1000 avec un pas de 100, deux tableaux sont remplis.

le premier avec les temps d'exécution de la fonction exécute une seule fois pour chaque n,

le deuxième avec les temps d'exécution de la fonction exécute 100 fois pour chaque n.

n	temps 1 exécution	temps 100 execution
1	1.1576999895623885e-05	6.51459995424375e-05
101	1.997099934669677e-05	0.0005197880000196164
201	1.4060000466997735e-05	0.0011396190002415096
301	1.90669998119 11955e-05	0.00169302500034 9735
401	2.3580000743095297e-05	0.0022514090005643084
501	9.939200026565231e-05	0.003384136999557086
601	3.644099979283055e-05	0.003361129000040819
701	4.04 4399975100532e-05	0.0039529949 99963266
801	4.663900017476408e-05	0.005711273000088113
901	5.3577000471705105e-05	0.005390480000642128

La ligne 16 exécute donc la fonction une seule fois alors que la ligne 17 l'exécute 100 fois.

D

Le programme part de celui de la question C et ensuite affiché à l'aide de matplot un graphique qui a comme abscisses la valeur de n pour chaque temps d'exécution et comme ordonnées le temps d'exécution en seconde.

Pour mettre en évidence la complexité en temps de la fonction qui calcul le factoriel de n ils suffit de modifier la ligne 8 en remplaçant le + par un *.

Exercice 2

A

Voici la fonction qui effectue un tri par sélection en python :

```python
def tri_selection(t):
    n = len(t)
    for i in range(n - 1):
        min_idx = i
        for j in range(i + 1, n):
            if t[j] < t[min_idx]:
                min_idx = j
        if min_idx != i:
            t[i], t[min_idx] = t[min_idx], t[i]
```

B

Voici le programme principal qui permet de faire le pire et le meilleur des cas

```python
# Pour le meilleur cas (tableau déjà trié dans l'ordre croissant) :
T_meilleur_cas = list(range(1, 101))
tri_selection(T_meilleur_cas)
print("Tableau après tri (meilleur cas) :", T_meilleur_cas)

# Pour le "pire des cas" (tableau trié dans l'ordre décroissant) :
T_pire_cas = list(range(100, 0, -1))
tri_selection(T_pire_cas)
print("Tableau après tri (pire des cas) :", T_pire_cas)
```

C

Le protocole d'expérimentation serait le suivant :

- Exécuter la fonction tri_selection avec des valeur différentes mais le tableau de même taille
- Chaque fois que la fonction entre dans la condition if t[j] < t[min_idx] incrémenter une variable
- récupérer cette variable dans un tableau et en faire une moyenne de toutes les valeurs du tableau

D

```
x, y1 = [], []

for i in range(1, 10000, 100):
    n = [random.randint(1,  500) for _ in range(0, i)]

    t1 = timeit.timeit("tri_selection(n)", globals=globals(), number=1)
    x.append(i)
    y1.append(t1)


# Tracer le graphique
plt.plot(x, y1, marker='o', linestyle='-', color='b', label='prems')
plt.xlabel('Abscisses')
plt.ylabel('Ordonnés')

# Afficher le graphique
plt.savefig("triselect.png")
```

Exercice 3

A

```python
def rechMax(T):
    a = T[0]
    count_test = 0
    for i in range(1, len(T)):
        if T[i] > a:
            a = T[i]
        count_test += 1
    return count_test, a
```

Initialisation du tableau de valeur :

```python
T = [random.randint(-500,  500) for _ in range(0, 100)]
```

B

```python
T_meilleur_cas = list(range(1, 101))
count, max = rechMax(T_meilleur_cas)
print("Max Best Case :", max, "number of test", count)

T_pire_cas = list(range(100, 0, -1))
rechMax(T_pire_cas)
print("Max Worst Case :", max, "number of test", count)
```

C

Le protocole d'expérimentation serait le suivant :

- Exécuter la fonction rechMax avec des valeur différentes mais le tableau de même taille
- Chaque fois que la fonction entre dans la boucle i incrémenter une variable
- récupérer cette variable dans un tableau et en faire une moyenne de toutes les valeurs du tableau

D

```
x, y1 = [], []

for i in range(1, 10000, 100):
    n = [random.randint(-500,  500) for _ in range(0, i)]

    t1 = timeit.timeit("rechMax(n)", globals=globals(), number=1)
    x.append(i)
    y1.append(t1)


# Tracer le graphique
plt.plot(x, y1, marker='o', linestyle='-', color='b', label='prems')
plt.xlabel('Abscisses')
plt.ylabel('Ordonnés')

# Afficher le graphique
plt.savefig("triselect.png")
```

Exercice 4

A

```
def wits_first(n, know):
    for i in range(1, n + 1):
        isastar = True
        for j in range(1, n + 1):
            if (i != j) and not know(j, i):
                isastar = False
                break
        for j in range(1, n + 1):
            if (i != j) and know(i, j):
                isastar = False
                break
        if isastar:
            return i
```

```
    return 0
```

B

```
def find_star(n, know):
    candidate = 1
    for i in range(2, n + 1):
        if know(candidate, i):
            candidate = i

    # Now, validate if the 'candidate' is actually a star.
    for i in range(1, n + 1):
        if i != candidate:
            if not know(i, candidate) or know(candidate, i):
                return 0

    return candidate
```

C

Exécutez la fonction find_star sur de nombreuses matrices KNOW générées aléatoirement, mais toujours de la même taille.

Chaque fois que la boucle principale de find_star est exécutée, incrémentez une variable (par exemple, compteur).

À la fin de chaque exécution de find_star, stockez la valeur du compteur dans un tableau.

Une fois toutes les exécutions terminées, calculez la moyenne des valeurs du tableau de compteurs.

```
x, temps_wits_first, temps_find_star = [], [], []

for i in range(10, 1001, 100):  # Changer la plage selon vos besoins
    KNOW = [[random.randint(0, 1) for j in range(i)] for k in range(i)]
```

```
    t1 = timeit.timeit(lambda: wits_first(i, know), number=1)
    t2 = timeit.timeit(lambda: find_star(i, know), number=1)

    x.append(i)
    temps_wits_first.append(t1)
    temps_find_star.append(t2)

# Tracer le graphique
plt.plot(x, temps_wits_first, marker='o', linestyle='-', color='b',
label='wits_first')
plt.plot(x, temps_find_star, marker='x', linestyle='-', color='r',
label='find_star')
plt.xlabel('Taille de la matrice')
plt.ylabel('Temps (secondes)')
plt.legend()
plt.title('Comparaison de performance')
plt.grid(True)
plt.savefig("comparaison_performance.png")
plt.show()
```