Tp8-R1.1

Écrire un programme permettant de manipuler une liste de N cases avec des entiers aléatoires (entre 0 et 100). Votre programme doit proposer un menu permettant à l'utilisateur de :

- créer un nouveau tableau (en choisissant N puis en le remplissant correctement),
- afficher le tableau de manière lisible
- trier le tableau (tri à bulle), et afficher le temps de traitement
- trier le tableau (tri par insertion), et afficher le temps de traitement
- rechercher un élément (dans un tableau trié)
- quitter

Concernant le tri :

Pour mesurer le temps, vous utiliserez la fonction `time()` du module `time` de `Python`, qui renvoie le temps en secondes depuis *epoch* sous forme de nombre à virgule flottante (*epoch* sur nos systèmes est le 1er janvier 1970)

Pour chaque algorithme, vous mesurerez les temps nécessaires pour trier des tableaux de 10000 à 50000 nombres et en déduirez la complexité de votre algorithme (pensez à représenter graphiquement vos mesures). Vous estimerez le temps nécessaire à votre algorithme pour trier 1000000 de valeurs

Concernant la recherche :

On attend ici une recherche dichotomique dans un tableau trié. Il faut donc avant tout chose vérifier que le tableau est effectivement trié.

pseudo code

tri à bulle:

Cette partie implémente un tri à bulle qui compare chaque élément avec son voisin et échange leurs positions si nécessaire pour les ordonner. Ce processus est répété jusqu'à ce que tous les éléments soient triés. Bien que simple, ce tri est inefficace pour les grandes tailles de tableau, car sa complexité est O(n2)O(n^2)O(n2).

```
p ← n-1
échange ← Vrai
tant que échange et p>0 faire
      echange ← faux
      pour i de 0 à p-1 faire
         si t[i]>t[i+1] alors
           tmp ← t[i+1]
           t[i+1] ← tmp

           echange ← vrai
         fin si
      fin faire
      p← p-1
fin faire
```

tri par insertion:

Ce tri déplace chaque élément dans une position triée du tableau. Il fonctionne en insérant chaque élément dans la partie triée à sa position correcte, ce qui nécessite des décalages pour faire de la place. La complexité est aussi O(n2)O(n^2)O(n2), mais il peut être plus rapide que le tri à bulle pour certaines configurations.

```
pour i de 1  n-1 faire
      val ← t[i]
      #décalage tant que nécessaire
      j ← i
      tant que j>0 et t[j-1]> val faire
                t[j] ← t[j-1]
                j ← j-1
      fin faire
      #ajout à la bonne place
      t[j] ← val
fin faire
```

recherche dichotomique:

Cette fonction recherche un élément dans un tableau trié en le divisant en deux à chaque étape, réduisant ainsi la taille de la recherche de moitié à chaque fois. Elle ne fonctionne correctement que si le tableau est trié, d'où la vérification initiale de l'ordre des éléments.

```
pour i de 0 à n-2 faire #ATTENTION au -2!
                        #indice du minimum
      #rechercher le minimum
      pour j de i +1 à n-1 faire
            si t[j]<t[i min] alors
                  i min ← j
            finsi
      fin faire

      #échanger les cases si nécessaire
      si imin! = i alors
            tmp ← t[i min]
            t[i min]<t[i]
            t[i] ← tmp
      fin si
finfaire
```

programme:

```python
import time
import random

# Dictionnaire des options du menu, avec des descriptions pour chaque option
menu_options = {
    "1": "Créer un nouveau tableau de N cases (entre 0 et 100) remplies aléatoirement",
    "2": "Afficher le tableau de manière lisible",
    "3": "Trier le tableau (tri à bulle) et afficher le temps de traitement",
    "4": "Trier le tableau (tri par insertion) et afficher le temps de traitement",
    "5": "Rechercher un élément (dans le tableau trié)",
    "6": "Quitter"
}

# Fonction pour afficher le menu des options
def print_menu():
    for key in menu_options.keys():
        print(key, '-- ', menu_options[key])

# Fonction pour créer un tableau de taille n avec des entiers aléatoires entre 0 et 100
def create_table(n: int) -> list:
    table = []
    for i in range(n):
        table.append(random.randint(0, 100))
    return table

# Fonction pour afficher le tableau de manière lisible
def print_table(table: list[int]):
    for i in range(len(table)):
        print(table[i], end=' | ')
    print()
```

```python
# Fonction de tri à bulle pour trier le tableau en place
# Le tri à bulle compare les éléments adjacents et les échange s'ils sont dans le mauvais ordre
def tri_a_bulle(table: list[int]) -> list:
    n = len(table)
    for i in range(n):
        for j in range(0, n - i - 1):
            if table[j] > table[j + 1]:
                table[j], table[j + 1] = table[j + 1], table[j]
    return table

# Fonction de tri par insertion pour trier le tableau en place
# Le tri par insertion prend chaque élément et l'insère dans la bonne position dans la partie triée du tableau
def tri_par_insertion(table: list[int]) -> list:
    n = len(table)
    for i in range(1, n):
        key = table[i]
        j = i - 1
        while j >= 0 and key < table[j]:
            table[j + 1] = table[j]
            j -= 1
        table[j + 1] = key
    return table
```

```python
# Fonction de recherche dichotomique pour trouver un élément dans un tableau trié
# Elle renvoie l'index de l'élément si trouvé, sinon -1
def recherche_dichotomique(table: list[int], x: int) -> int:
    # Vérification si le tableau est trié, sinon retourne -1
    if table != sorted(table):
        print("Le tableau n'est pas trié.")
        return -1

    # Initialisation des indices de début et de fin
    n = len(table)
    debut = 0
    fin = n - 1

    # Recherche dichotomique, divisant l'intervalle en deux à chaque étape
    while debut <= fin:
        milieu = (debut + fin) // 2
        if table[milieu] == x:
            return milieu
        elif table[milieu] < x:
            debut = milieu + 1
        else:
            fin = milieu - 1
    return -1
```

```python
# Code principal, affichant le menu et gérant les choix de l'utilisateur
if __name__ == '__main__':
    table: list[int]  # Déclaration du tableau
    option = str      # Déclaration de la variable pour l'option choisie

    while True:
        print_menu()  # Affiche le menu
        option = ''
        try:
            option = input("Entrez votre choix: ")  # Lit le choix de l'utilisateur
        except:
            print("Entrez un nombre valide")

        # Gestion des options de menu
        if option == '1':
            # Option 1 : Crée un nouveau tableau
            n = int(input("Entrez le nombre de cases: "))
            if n < 0 :
                print("Entrez un nombre supérieur à 0")
                n = int(input("Entrez le nombre de cases: "))
            table = create_table(n)
        elif option == '2':
            # Option 2 : Affiche le tableau actuel
            print_table(table)
        elif option == '3':
            # Option 3 : Trie le tableau avec le tri à bulle et mesure le temps de traitement
            start = time.time()
            table = tri_a_bulle(table)
            end = time.time()
            print_table(table)
            print("Temps de traitement: ", end - start)
```

```python
elif option == '4':
    # Option 4 : Trie le tableau avec le tri par insertion et mesure le temps de traitement
    start = time.time()
    table = tri_par_insertion(table)
    end = time.time()
    print_table(table)
    print("Temps de traitement: ", end - start)
elif option == '5':
    # Option 5 : Recherche d'un élément dans le tableau trié
    x = int(input("Entrez l'élément à rechercher: "))
    index = recherche_dichotomique(table, x)
    if index != -1:
        print("L'élément est à l'index: ", index)
    else:
        print("L'élément n'est pas dans le tableau")
elif option == '6':
    # Option 6 : Quitter le programme
    break
```

La section python représente le cœur de l'application, offrant une interface utilisateur qui guide les actions principales. Elle affiche un menu interactif permettant de manipuler un tableau de valeurs entières selon différentes opérations. Tout commence par la création d'un tableau, où l'utilisateur choisit la taille, puis chaque élément est généré aléatoirement entre 0 et 100. L'utilisateur peut ensuite afficher ce tableau, le trier en utilisant soit le tri à bulle soit le tri par insertion, et mesurer le temps de traitement pour chaque tri. Une fois trié, le tableau

peut être utilisé pour des recherches dichotomiques, avec une vérification préalable pour s'assurer qu'il est effectivement en ordre. L'application vérifie aussi que les choix de l'utilisateur sont valides, offrant une robustesse supplémentaire contre les erreurs de saisie. Ce flux d'interactions permet d'expérimenter différents algorithmes de tri et de recherche tout en observant leurs performances en temps réel sur des données aléatoires de différentes tailles.

jeux de tests:

```
1 --  Créer un nouveau tableau de N cases (entre 0 et 100) remplies aléatoirement
2 --  Afficher le tableau de manière lisible
3 --  Trier le tableau (tri à bulle) et afficher le temps de traitement
4 --  Trier le tableau (tri par insertion) et afficher le temps de traitement
5 --  rechercher un élément (dans le tableau trié)
6 --  Quitter
Entrez votre choix: 1
Entrez le nombre de cases: 23
1 --  Créer un nouveau tableau de N cases (entre 0 et 100) remplies aléatoirement
2 --  Afficher le tableau de manière lisible
3 --  Trier le tableau (tri à bulle) et afficher le temps de traitement
4 --  Trier le tableau (tri par insertion) et afficher le temps de traitement
5 --  rechercher un élément (dans le tableau trié)
6 --  Quitter
Entrez votre choix: 2
69 | 24 | 5 | 77 | 71 | 62 | 94 | 15 | 95 | 72 | 100 | 7 | 1 | 94 | 5 | 58 | 88 | 88 | 59 | 20 | 81 | 52 | 40 |
```

```
Entrez votre choix: 3
1 | 5 | 5 | 7 | 15 | 20 | 24 | 40 | 52 | 58 | 59 | 62 | 69 | 71 | 72 | 77 | 81 | 88 | 88 | 94 | 94 | 95 | 100 |
Temps de traitement:  0.0
```

Tentative de tri à bulle sur un tableau de 23 cases.

```
Entrez votre choix: 1
Entrez le nombre de cases: 23
1 --  Créer un nouveau tableau de N cases (entre 0 et 100) remplies aléatoirement
2 --  Afficher le tableau de manière lisible
3 --  Trier le tableau (tri à bulle) et afficher le temps de traitement
4 --  Trier le tableau (tri par insertion) et afficher le temps de traitement
5 --  rechercher un élément (dans le tableau trié)
6 --  Quitter
Entrez votre choix: 2
2 | 69 | 44 | 45 | 30 | 74 | 51 | 37 | 71 | 90 | 83 | 29 | 5 | 78 | 44 | 19 | 33 | 88 | 49 | 27 | 100 | 86 | 73 |
```

```
Entrez votre choix: 4
2 | 5 | 19 | 27 | 29 | 30 | 33 | 37 | 44 | 44 | 45 | 49 | 51 | 69 | 71 | 73 | 74 | 78 | 83 | 86 | 88 | 90 | 100 |
Temps de traitement:  0.0
```

Recréation d'un tableau de 23 cases pour un tri par insertion

```
1 --  Créer un nouveau tableau de N cases (entre 0 et 100) remplies aléatoirement
2 --  Afficher le tableau de manière lisible
3 --  Trier le tableau (tri à bulle) et afficher le temps de traitement
4 --  Trier le tableau (tri par insertion) et afficher le temps de traitement
5 --  rechercher un élément (dans le tableau trié)
6 --  Quitter
Entrez votre choix: 1
Entrez le nombre de cases: 5
1 --  Créer un nouveau tableau de N cases (entre 0 et 100) remplies aléatoirement
2 --  Afficher le tableau de manière lisible
3 --  Trier le tableau (tri à bulle) et afficher le temps de traitement
4 --  Trier le tableau (tri par insertion) et afficher le temps de traitement
5 --  rechercher un élément (dans le tableau trié)
6 --  Quitter
Entrez votre choix: 2
7 | 81 | 91 | 25 | 30 |
1 --  Créer un nouveau tableau de N cases (entre 0 et 100) remplies aléatoirement
2 --  Afficher le tableau de manière lisible
3 --  Trier le tableau (tri à bulle) et afficher le temps de traitement
4 --  Trier le tableau (tri par insertion) et afficher le temps de traitement
5 --  rechercher un élément (dans le tableau trié)
6 --  Quitter
Entrez votre choix: 5
Entrez l'élément à rechercher: 6
Le tableau n'est pas trié.
```

Création d'un nouveau tableau pour tester si la fonction dichotomie s'active si le tableau n'est pas trié.
Elle ne s'active pas et renvoie que le tableau n'est pas trié.

Dans cette section, différents tests sont exécutés pour vérifier le bon fonctionnement des algorithmes. Par exemple :

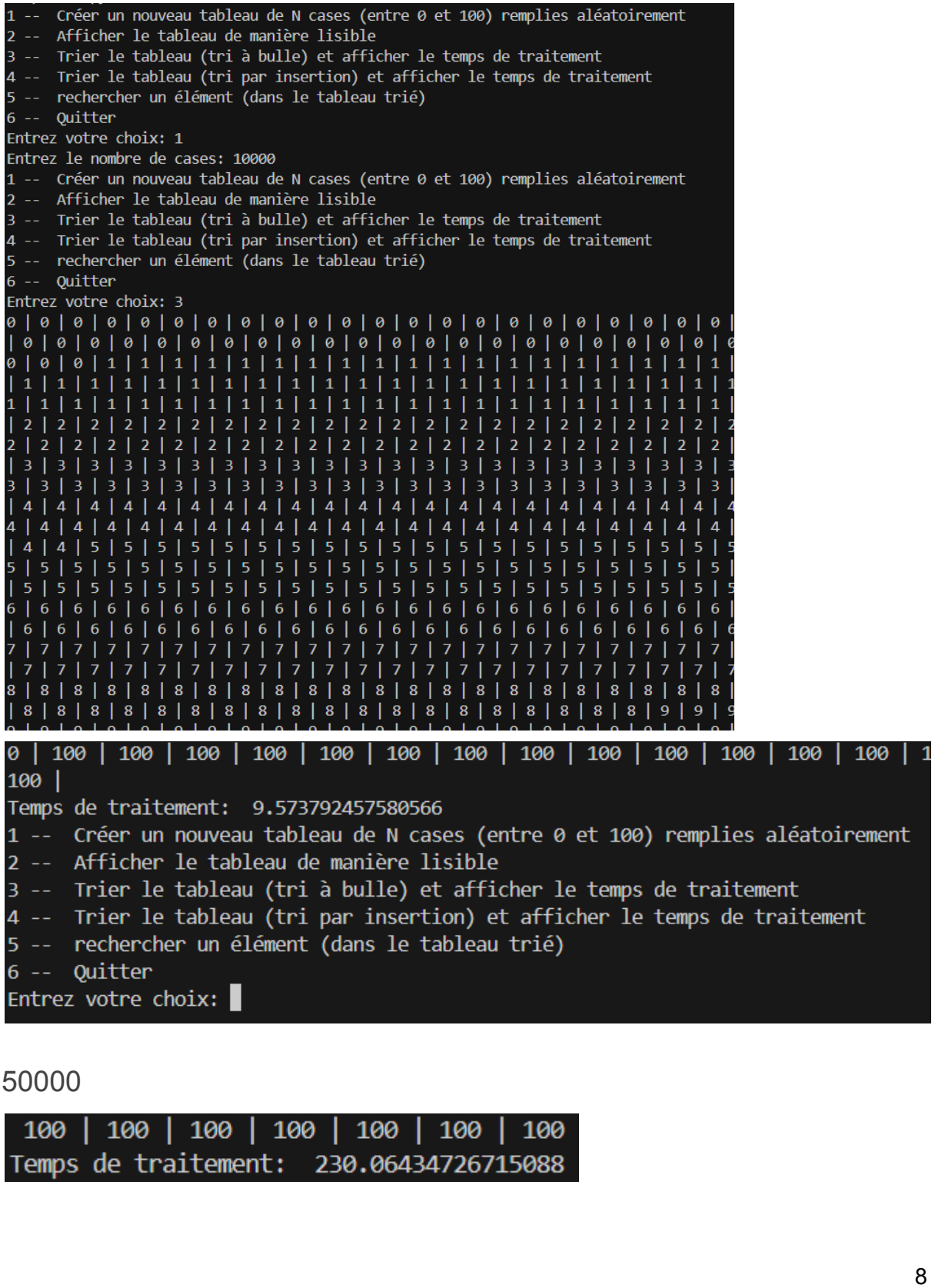
- Un test de tri à bulle sur un tableau de 23 éléments, suivi d'un test de tri par insertion sur un tableau similaire.
- Un test de recherche dichotomique sur un tableau non trié, vérifiant que la fonction signale correctement que le tableau n'est pas trié.
- Des tests de performance sont également réalisés pour observer les temps de tri à bulle et d'insertion sur des tableaux de tailles différentes (10 000, 50 000), permettant de comparer l'efficacité de chaque méthode sur des ensembles plus grands.

Ces jeux de tests assurent que le code fonctionne correctement dans diverses situations et permettent de vérifier les temps de calcul pour différents algorithmes.

recherches sur temps de tri

tri a bulle

10000 valeurs



50000

tri par insertion

10000

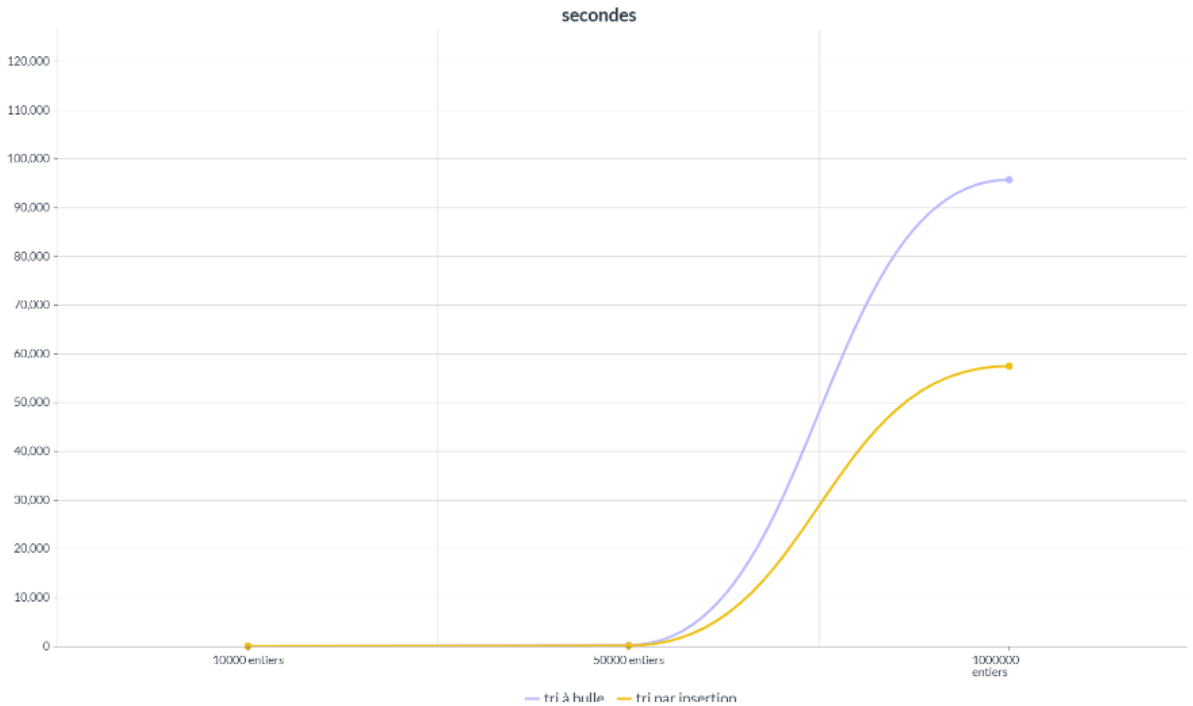
```
Entrez votre choix: 1
Entrez le nombre de cases: 10000
1 --  Créer un nouveau tableau de
2 --  Afficher le tableau de manièr
3 --  Trier le tableau (tri à bulle
4 --  Trier le tableau (tri par ins
5 --  rechercher un élément (dans
6 --  Quitter
Entrez votre choix: 4
Temps de traitement:  5.748672008514404
```

50000

```
Entrez votre choix: 1
Entrez le nombre de cases: 50000
1 --  Créer un nouveau tableau de
2 --  Afficher le tableau de manièr
3 --  Trier le tableau (tri à bull
4 --  Trier le tableau (tri par in
5 --  rechercher un élément (dans
6 --  Quitter
Entrez votre choix: 4
Temps de traitement:  112.91672825813293
```

recherche commune

recherches	10000	50000	100000
tri à bulle	9,573792457580566	230,06434726715088	95 737,92 secondes (soit environ 26,6 heures)
tri par insertion	5,748672008514404	112,91672825813293	57 486,72 secondes (soit environ 15,96 heures)


secondes
120,000
110,000
100,000
90,000
80,000
70,000
60,000
50,000
40,000
30,000
20,000
10,000
0
10000 entiers
50000 entiers
1000000 entiers
tri à bulle
tri par insertion