Introduction

Exercice 1

Fonctions

`average_waiting_time(positions)`

- **But**: Calculer le temps d'attente moyen pour une liste donnée de positions.
- **Complexité Algorithmique**: O(n)
- **Explication**: La fonction parcourt la liste des positions une fois, calcule le temps relatif et total, puis retourne la moyenne.

`parcours(gL)`

- **But**: Générer un ordre de nettoyage en triant la liste et en séparant les valeurs négatives et positives.
- **Complexité Algorithmique**: O(n log n)
- **Explication**: La fonction trie la liste, ce qui a une complexité de O(n log n), puis traite la liste triée pour créer l'ordre de nettoyage.

`parcoursOnSQR(gL)`

- **But**: Générer un ordre de nettoyage en sélectionnant itérativement la position la plus proche.
- **Complexité Algorithmique**: $O(n^2)$
- **Explication**: La fonction trouve itérativement la position la plus proche, ce qui implique des boucles imbriquées, conduisant à une complexité quadratique.

Expérience 1a

- **Description**: Calculer le temps d'attente moyen pour plusieurs listes de positions.
- **Résultats**:
  - Positions: `[-1.5, 1.2, 2.3, 5.0]`, Temps d'attente moyen: 4.75 secondes
  - Positions: `[5, -1.5, 2.3, 1.2]`, Temps d'attente moyen: 12.05 secondes
  - Positions: `[-15, 7, -6, -13]`, Temps d'attente moyen: 39.75 secondes
  - Positions: `[-13, -15, -6, 7]`, Temps d'attente moyen: 22.25 secondes

Expérience 1b

- **Description**: Calculer le temps d'attente moyen pour un autre ensemble de positions.
- **Résultats**:
  - Positions: `[-3, 9.5, -6.2, 2]`, Temps d'attente moyen: 22.275 secondes
  - Positions: `[9.5, -3, 2, -6.2]`, Temps d'attente moyen: 23.425 secondes

Expérience 2

- **Description**: Comparer les temps d'attente moyens pour différents ordres de positions.
- **Résultats**:
  - Temps d'attente moyen (Ordre Trié): 10.8 secondes

    - Temps d'attente moyen (Ordre Voisin le Plus Proche): 12.8 secondes
    - Temps d'attente moyen (Ordre Optimal Proposé): 9.2 secondes

Expérience 3

- **Description**: Mettre en place un algorithme permettant d'optimiser le parcours du facteur.
- **Résultats**:
    - Temps d'attente moyen de `parcours`: 0.11.10 minutes
    - Temps d'attente moyen de `parcoursOnSQR`: 1.56.41 minutes

Conclusion

Nous avons opté pour la solution en O(n log n). Au vu des résultats obtenus, les deux solutions donnent en moyenne des valeurs suffisamment proches pour être négligeables. Cependant, le temps d'exécution est nettement inférieur à celui d'une fonction en $O(n^2)$.

Exercice 2

Fonctions

`generate_random_graph(num_nodes, num_edges)`

- **But**: Générer un graphe aléatoire avec un nombre donné de nœuds et d'arêtes.
- **Complexité Algorithmique**: O(n + m)
- **Explication**: La fonction utilise l'algorithme gnp_random_graph de NetworkX, qui a une complexité linéaire par rapport au nombre de nœuds et d'arêtes.

`max_degree_matching(G)`

- **But**: Trouver un appariement maximal en sélectionnant les nœuds de degré maximal.
- **Complexité Algorithmique**: O(n log n + m)
- **Explication**: La fonction trie les nœuds par degré décroissant (O(n log n)) et parcourt les voisins pour créer des appariements (O(m)).

`min_degree_matching(G)`

- **But**: Trouver un appariement minimal en sélectionnant les nœuds de degré minimal.
- **Complexité Algorithmique**: O(n log n + m)
- **Explication**: La fonction trie les nœuds par degré croissant (O(n log n)) et parcourt les voisins pour créer des appariements (O(m)).

Expériences

Expérience 2a

- **Description**: Comparer les tailles moyennes des appariements pour les méthodes de degré maximal et minimal sur plusieurs essais.
- **Résultats**:
    - Taille moyenne de l'appariement (Degré Maximal): 48.83

    - Taille moyenne de l'appariement (Degré Minimal): 55.50

Expérience 2b

- **Description**: Comparer les temps d'exécution moyens pour les méthodes de degré maximal et minimal sur plusieurs essais.
- **Résultats**:
    - Temps moyen d'exécution (Degré Maximal): 0.00004 secondes
    - Temps moyen d'exécution (Degré Minimal): 0.00003 secondes

Conclusion

Les résultats montrent que la méthode d'appariement par degré minimal tend à produire des appariements légèrement plus grands que la méthode par degré maximal. Cependant, les deux méthodes ont des temps d'exécution similaires, avec une légère préférence pour la méthode par degré minimal en termes de rapidité.

Exercice 3

This script measures the execution time of two functions, `function_1` and `function_2`, for different input sizes, fits polynomial curves to the measured times, and plots the results.

Functions:

`measure_time(func, n_values)`

- **Description**: Measures the execution time of a given function `func` for different input sizes specified in `n_values`.
- **Résultats**: Temps d'attente pour l'exécution de la fonction donnée en paramètre par rapport à chaque variable du tableau donné.

Variables:

- **n_values**: A list of input sizes for which the execution times are measured.
- **times_1**: A list of measured execution times for `function_1`.
- **times_2**: A list of measured execution times for `function_2`.
- **poly_fit_1**: Coefficients of the polynomial fit (degree 4) for the measured times of `function_1`.
- **poly_fit_2**: Coefficients of the polynomial fit (degree 2) for the measured times of `function_2`.
- **fit_times_1**: Fitted polynomial values for `function_1` based on `poly_fit_1`.
- **fit_times_2**: Fitted polynomial values for `function_2` based on `poly_fit_2`.

Plots:

- **Subplot 1**: Measured vs Fitted Times for `function_1`.
- **Subplot 2**: Measured vs Fitted Times for `function_2`.

Exercice 4

Functions:

`choose_signs(numbers: np.ndarray, b: float) -> np.ndarray`

This function chooses the signs of the elements in the input array `numbers` such that the absolute value of the sum of the elements, each multiplied by its chosen sign, is minimized. The function returns an array of signs (1 or -1) for each element in `numbers`.

Parameters:

- **numbers (np.ndarray)**: A numpy array of numbers for which the signs need to be chosen.
- **b (float)**: A constant value to be added to the sum of the elements in `numbers`.

Returns:

- **np.ndarray**: An array of signs (1 or -1) for each element in `numbers` that minimizes the absolute value of the sum.