Projet Partie 1: Analyse

1. **PriorityQueue1**

- Cette implémentation repose sur un simple tableau trié.
- Avantages :
  - La structure est simple à mettre en œuvre.
  - L'insertion d'éléments est relativement simple à gérer.
- Inconvénients :
  - Les opérations d'insertion et de retrait peuvent être coûteuses en termes de performances, car elles nécessitent le déplacement d'éléments pour maintenir le tri.
  - Les performances peuvent se dégrader rapidement à mesure que la taille de la file à priorité augmente.
- Convient aux scénarios où la taille de la file à priorité est petite et les performances ne sont pas une préoccupation majeure.

2. **PriorityQueue2**

- Cette implémentation repose sur un tableau structuré en monceau (heap) où l'élément maximum est à la racine.
- Avantages :
  - Les opérations d'insertion et de retrait sont basées sur des propriétés de monceau et sont plus efficaces que dans PriorityQueue1.
  - Le monceau garantit que les performances restent stables, même avec une grande quantité de données.
- Inconvénients :
  - L'implémentation est plus complexe que PriorityQueue1, mais elle garantit de meilleures performances.
- Convient aux scénarios où des performances élevées sont nécessaires et où une implémentation personnalisée est préférée.

3. **PriorityQueue3:**

- Cette implémentation utilise la classe standard `java.util.PriorityQueue`.
- Avantages :
  - `java.util.PriorityQueue` est optimisée pour les performances, basée sur un monceau et fournit une interface simple.
  - Les opérations d'insertion, de retrait et de recherche du minimum sont toutes efficaces et ne nécessitent pas de gestion manuelle de la structure de données.
- Inconvénients :
  - Moins de flexibilité par rapport à l'implémentation personnalisée de PriorityQueue2.
- Convient à la plupart des cas d'utilisation où des performances élevées et une simplicité d'utilisation sont essentielles.

Conclusion :

Le choix entre ces trois implémentations dépend des besoins spécifiques de votre application. Si la simplicité est votre principale préoccupation et que vous ne voulez pas gérer manuellement la structure de données, `java.util.PriorityQueue` (PriorityQueue3) est recommandée. Si vous avez besoin de performances optimales et êtes prêt à gérer une structure de données plus complexe, PriorityQueue2 (basée sur un monceau) est une excellente option. PriorityQueue1 est adaptée à des cas très spécifiques où la taille de la file à priorité est petite et où les performances ne sont pas cruciales.

	PQ1	PQ2	PQ3
K1	35882ms	35830ms	34246ms
K10	35532ms	34351ms	35853ms
K50	34763ms	34979ms	35585ms
K100	34979ms	36033ms	32956ms
K500	34933ms	31994ms	31973ms
K2000	38464ms	32133ms	32745ms

Tableau montrant les différents temps d'exécutions pour différents K en utilisant les trois différentes implémentations.

Find KNN Performance Analysis: K's Value vs Execution Time Using Difference Priority Queue Implementations