Řadící algoritmy

1. Definice

- **Řazení** je proces uspořádání dat do specifikovaného pořadí, obvykle podle velikosti nebo abecedy.
- **Příklad aplikace**: Seřazení seznamu čísel pro vyhledávání, uspořádání seznamu jmen podle abecedy.
- Řadící algoritmy lze rozdělit podle **metody** (např. výměnou, porovnáním), **stability** (některé algoritmy zachovávají pořadí rovných prvků, jiné ne) a **složitosti** (jak rychle algoritmus dokáže seřadit data).

---

2. Typy řadících algoritmů

2.1. Bubble Sort (Bublinkové řazení)

- **Princip**: Algoritmus opakovaně prochází seznamem a porovnává sousední prvky. Pokud jsou v špatném pořadí, vymění je.
    - Tato operace se opakuje pro všechny prvky seznamu, dokud není seznam seřazen.
- **Postup**:
    - Porovnání prvního a druhého prvku seznamu.
    - Pokud jsou v špatném pořadí, zamění se.
    - Algoritmus pokračuje se zbývajícími prvky.
    - Tento proces se opakuje, dokud seznam není celý seřazený.
- **Složitost**:
    - **Nejhorší případ**: O(n2), kde n je počet prvků.
    - **Průměrný případ**: O(n2).
    - **Nejlepší případ**: O(n) (pokud je seznam již seřazen, lze detekovat a algoritmus skončí dříve).
- **Výhody**:
    - Jednoduchost implementace.
    - Stabilní algoritmus (zachovává pořadí rovných prvků).
- **Nevýhody**:
    - Velmi neefektivní pro velké seznamy.

2.2. Selection Sort (Výběrové řazení)

- **Princip**: Algoritmus opakovaně vybírá nejmenší (nebo největší) prvek a vkládá ho na správné místo v seznamu.
    - Pro každý prvek seznamu najde minimální prvek a umístí jej na aktuální pozici.
- **Postup**:

    - Najde nejmenší prvek v ne-seřazené části seznamu.
    - Vymění ho s prvním nenavštíveným prvkem.
    - Opakuje pro každý další prvek v seznamu.
- **Složitost**:
    - **Nejhorší, průměrný a nejlepší případ**: O(n2).
- **Výhody**:
    - Není potřeba tolik výměn jako u Bubble Sort.
- **Nevýhody**:
    - Opět velmi neefektivní pro velké seznamy.
    - Nezachovává stabilitu.

2.3. Insertion Sort (Vkládací řazení)

- **Princip**: Algoritmus pracuje podobně jako při třídění karet. Vkládá prvek na jeho správné místo v již seřazené části seznamu.
    - Tento algoritmus je efektivní pro malé množství dat nebo seznamy, které jsou již částečně seřazeny.
- **Postup**:
    - Začíná se druhým prvkem.
    - Tento prvek je porovnán s předchozím a vložen na správné místo v seřazené části seznamu.
    - Tento proces se opakuje pro každý další prvek.
- **Složitost**:
    - **Nejhorší a průměrný případ**: O(n2).
    - **Nejlepší případ**: O(n) (pokud je seznam již seřazen).
- **Výhody**:
    - Stabilní algoritmus.
    - Efektivní pro malé seznamy nebo již částečně seřazené seznamy.
- **Nevýhody**:
    - Opět neefektivní pro velké seznamy.

2.4. Merge Sort (Sloučovací řazení)

- **Princip**: Algoritmus používá princip "rozděl a panuj". Seznam je opakovaně dělen na poloviny, až dosáhne velikosti 1, a potom jsou jednotlivé poloviny sloučeny zpět v seřazeném pořadí.
- **Postup**:
    1. Rozdělte seznam na dvě poloviny.
    2. Seřaďte obě poloviny rekurzivně.
    3. Sloučte seřazené poloviny do jednoho seřazeného seznamu.

- **Složitost**:
    1. **Nejhorší, průměrný a nejlepší případ**: O(nlogn).
- **Výhody**:
    1. Velmi efektivní pro velké seznamy.
    2. Stabilní algoritmus.
- **Nevýhody**:
    1. Vyžaduje dodatečnou paměť pro sloučení.
    2. Není to in-place algoritmus (neprovádí řazení přímo v původní struktuře).

2.5. Quick Sort (Rychlé řazení)

- **Princip**: Rychlé řazení využívá metodu "rozděl a panuj", kde se vybere "pivotní" prvek a seznam se rozdělí na dvě části: jednu s prvky menšími než pivot a druhou s prvky většími. Tyto části se pak rekurzivně třídí.
- **Postup**:
    1. Vyberte pivotní prvek (např. první prvek, poslední prvek nebo prvek vybraný náhodně).
    2. Seřaďte seznam tak, že všechny prvky menší než pivot jsou vlevo a všechny prvky větší než pivot jsou vpravo.
    3. Rekurzivně opakujte tento proces pro levý a pravý podseznam.
- **Složitost**:
    1. **Nejhorší případ**: O(n2) (pokud je pivot vždy nejmenší nebo největší prvek).
    2. **Průměrný a nejlepší případ**: O(nlogn).
- **Výhody**:
    1. Velmi rychlý pro velké seznamy.
    2. Efektivní v průměru.
- **Nevýhody**:
    1. V nejhorším případě může být neefektivní.
    2. Není stabilní algoritmus.

2.6. Heap Sort (Hromadné řazení)

- **Princip**: Tento algoritmus používá strukturu dat zvanou **halda**, která je binární strom uspořádaný tak, že každý rodič je větší (nebo menší) než jeho potomci. Algoritmus používá heap k efektivnímu vybírání maximálního (nebo minimálního) prvku a jeho přesunutí na správné místo v seznamu.
- **Postup**:
    1. Vytvořte haldu z daného seznamu.
    2. Opakovaně odstraňujte kořen haldy (maximální nebo minimální prvek) a umístěte jej na konec seznamu.

3. Po každém odstranění kořene proveďte úpravu haldy, aby zůstala vyvážená.

- **Složitost**:
    1. **Nejhorší, průměrný a nejlepší případ**: O(nlogn).
- **Výhody**:
    1. Efektivní pro velké seznamy.
    2. Nepotřebuje dodatečnou paměť pro sloučení jako Merge Sort.
- **Nevýhody**:
    1. Není stabilní.
    2. Méně intuitivní a těžší na implementaci než jiné algoritmy.