Algoritmi de sortare

Video care ilustrează comparația dintre:

- ShellSort
- QuickSort
- CocktailSort
- MergeSort
- SelectionSort
- InsertionSort
- CombSort
- BubbleSort

https://www.youtube.com/watch?v=ZZuD6iUe3Pc

În continuare, voi prezenta algoritmii de sortare implementați în temă:

BubbleSort

Sortarea prin metoda bulelor (Bubble Sort) este cel mai simplu algoritm de sortare care funcționează schimbând în mod repetat elementele adiacente dacă acestea sunt în ordinea greșită. Dacă avem N elemente totale, atunci trebuie să repetăm procesul de mai sus de N-1 ori.

- Funcționează bine cu seturi de date mari, unde articolele sunt aproape sortate, deoarece este nevoie de o singură iterație pentru a detecta dacă lista este sortată sau nu. Dar dacă lista este nesortată la o extindere mare, atunci acest algoritm este valabil pentru seturi de date mici sau liste.

- Acest algoritm este cel mai rapid pe un set de date extrem de mic sau aproape sortat.

CountingSort

Sortarea prin numărare (Counting Sort) este un algoritm liniar de sortare, o metodă de sortare a vectorilor ce se bazează pe utilizarea unui vector de frecvență.

- Counting Sort este eficientă dacă intervalul de date de intrare nu este semnificativ mai mare decât numărul de elemente de sortat.

- Nu este o sortare bazată pe comparație. Complexitatea timpului de rulare este O(n) cu spațiu proporțional cu intervalul de date.

- Este adesea folosit ca subrutină la un alt algoritm de sortare, cum ar fi RadixSort.

- Counting Sort folosește un hash parțial pentru a număra apariția obiectului de date în O(1).

- Counting Sort poate fi extinsă pentru a funcționa și pentru input-uri negative.

RadixSort

Counting Sort este un algoritm liniar de sortare care sortează în timp O(n + k) când elementele sunt în intervalul de la 1 la k.

În schimb, pentru elementele care sunt în intervalul 1 - n^2, Counting Sort va avea complexitatea O(n^2).

RadixSort este răspunsul. Ideea sortării Radix este de a face sortare cifră cu cifră începând de la cea mai mică cifră la cea mai semnificativă. Sortarea Radix folosește Counting Sort ca subrutină pentru sortare.

- Dacă avem $\log_2(n)$ biți pentru fiecare cifră, timpul de funcționare al RadixSort-ului pare să fie mai bun decât QuickSort pentru o gamă largă de date de intrare.

- În schimb, RadixSort-ul folosește Counting Sort ca subrutină, iar Counting Sort ocupă spațiu suplimentar pentru sortarea numerelor.

MergeSort

Acest algoritm de sortare se bazează pe algoritmul Divide and Conquer . Împarte matricea de intrare în două jumătăți, se apelează singură pentru cele două jumătăți și apoi rulează pe cele două jumătăți sortate.

Funcția merge () este utilizată pentru fuzionarea a două jumătăți. Fuziunea (arr, l, m, r) este un proces cheie care presupune că arr [l..m] și arr [m + 1..r] sunt sortate și unește cele două sub-matrice sortate în una singură.

- Se folosește acolo unde se știe că datele sunt date similare.

- Principalul avantaj al MergeSort-ului este stabilitatea sa, elementele comparate păstrându-și în egală măsură ordinea originală.

QuickSort

Acest algoritm de sortare se bazează pe algoritmul Divide and Conquer. Acesta alege un element ca pivot și partiționează lista dată în jurul pivotului ales. După partiționarea listei pe baza elementului pivot, QuickSort este din nou aplicat recursiv la două subliste, adică, sublista din stânga elementului pivot și sublista din dreapta elementului pivot.

- QuickSort este cea mai rapidă sortare, dar nu este întotdeauna O (n*logn), deoarece există cele mai grave cazuri în care devine O (n^2) .

- QuickSort este probabil mai eficient pentru seturile de date care se potrivesc în memorie. Pentru seturi de date mai mari, se dovedește a fi ineficient, astfel încât algoritmii precum MergeSort sunt preferați în acest caz.

- QuickSort este o sortare în loc (adică nu necesită spațiu de stocare suplimentar), deci este adecvat să fie utilizată pentru tablouri.

BucketSort

BucketSort (sau BinSort) este un algoritm de sortare care funcționează prin distribuirea elementelor unui tablou într-un număr de bucket-uri. Fiecare bucket este apoi sortat individual, fie folosind un algoritm de sortare diferit, fie aplicând recursiv algoritmul BucketSort.

- BucketSort este util în principal atunci când datele de intrare sunt distribuite uniform într-un anumit interval.
(de ex., intervalul 0,0 - 1,0)