Algoritmi de sortare

Prezentarea Algoritmilor

1. Radix Sort

Algoritmul radix sort este un algoritm de sortare stabil, care sortează elementele unei liste prin compararea cifrelor fiecărui element, începând cu cifra cea mai puțin semnificativă și până la cifra cea mai semnificativă.

Radix sort împarte lista de sortat într-o serie de bucăți, în funcție de cifra cea mai puțin semnificativă a fiecărui element, apoi le sortează pe fiecare bucățică în parte. După ce fiecare cifră a fost sortată, lista este reconstruită combinând bucățile în ordinea lor originală.

Pentru a fi eficient, radix sort necesită ca numerele din lista de sortat să aibă un număr fix de cifre. În caz contrar, elementele trebuie să fie aliniate și să fie umplute cu zero-uri în partea stângă până când toate elementele au același număr de cifre.

Radix sort este eficient în cazul în care numărul de cifre este mic în comparație cu dimensiunea listei și este de obicei mai rapid decât algoritmi de sortare mai generali, cum ar fi quicksort sau mergesort, atunci când lista conține un număr mare de elemente cu numere lungi.

2. Merge Sort

Merge sort este un algoritm de sortare eficient, care împarte lista de sortat în bucăți din ce în ce mai mici, apoi le combină în ordine pentru a forma lista sortată finală.

Algoritmul merge sort funcționează astfel:

1. Imparte lista nesortată în două bucăți egale.
2. Sortează fiecare bucățică folosind recursivitate prin apelarea aceluiași algoritm merge sort.
3. Combină bucățile sortate pentru a forma lista sortată finală.

Combinația bucăților sortate este realizată prin interclasarea lor în ordine, astfel încât elementele din bucățile sortate să fie inserate în ordine în lista sortată finală. Avantajul principal al merge sort este faptul că acesta poate sorta liste mari și neordonate într-un timp rezonabil, fără a avea nevoie de multă memorie suplimentară.

Dezavantajul acestui algoritm este că poate fi dificil de implementat și înțeles, în special pentru programatori începători. De asemenea, merge sort are o complexitate de timp O(nlogn), ceea ce înseamnă că poate fi mai lent decât alte algoritmi de sortare, cum ar fi quicksort, în cazul unor liste de dimensiuni mici.

3. Shell Sort

Shell sort este un algoritm de sortare care își propune să îmbunătățească performanța algoritmului de sortare prin inserție prin gruparea elementelor listei în intervale mai mari de comparare.

În Shell sort, lista este împărțită în bucăți mai mici numite subliste și este aplicat algoritmul de sortare prin inserție pentru fiecare sublistă. În loc să fie comparate toate elementele într-o singură iterație, elementele din fiecare sublistă sunt sortate separat, utilizând algoritmul de sortare prin inserție, reducând astfel numărul de operații de comparare și swap necesare pentru a sorta lista.

Pe măsură ce algoritmul progresează, lungimea sublistelor este redusă treptat, iar elementele sunt sortate din ce în ce mai aproape, până când lista este în cele din urmă sortată complet.

Shell sort are o complexitate medie de timp O(n^1.5), ceea ce îl face mai rapid decât algoritmul de sortare prin inserție obișnuit, dar mai încet decât quicksort sau mergesort. Cu toate acestea, Shell sort este mai ușor de implementat și de înțeles decât unele alte algoritmi de sortare, cum ar fi radix sort.

4. Head Sort

Heap sort este un algoritm de sortare care se bazează pe utilizarea structurii de date heap (sau arbore binar de tip heap) pentru a ordona elementele listei.

În heap sort, elementele listei sunt înșirate într-un arbore binar de tip heap, astfel încât fiecare nod să fie mai mare (sau mai mic) decât nodurile sale fiice, respectând proprietatea heap-ului. Apoi, elementul cel mai mare (sau cel mai mic) este extras din heap și pus în lista sortată. Heap-ul este apoi reconstruit prin restabilirea proprietății heap-ului, iar procesul este repetat pentru a extrage și a pune în listă elementul următor cel mai mare (sau cel mai mic).

Pentru a construi un heap din lista de sortat, se începe prin transformarea listei într-un heap parțial (adică un heap care nu respectă întotdeauna proprietatea heap-ului), prin transformarea listei într-un arbore binar de tip heap, începând cu ultimul nod și continuând către rădăcina arborelui.

Heap sort are o complexitate de timp de O(n log n), ceea ce îl face eficient pentru liste mai mari. În plus, heap sort este un algoritm stabil de sortare, adică ordinea relativă a elementelor identice nu se va schimba după sortare.

Un dezavantaj al heap sort este că are nevoie de o cantitate suplimentară de memorie pentru a construi heap-ul, ceea ce poate fi problematic în cazul listelor foarte mari. De asemenea, în practică, heap sort poate fi mai lent decât alte algoritmi de sortare, cum ar fi quicksort sau mergesort, în anumite cazuri.

5. Quick Sort

Quick sort este un algoritm de sortare bazat pe principiul divide and conquer (împarte și cucerește). În acest algoritm, lista de sortat este împărțită în două subliste mai mici: o sublistă cu elemente mai mici decât pivotul și o sublistă cu elemente mai mari decât pivotul.

Pivotul este ales inițial din lista de sortat și este plasat astfel încât toate elementele din stânga sa să fie mai mici decât pivotul, iar toate elementele din dreapta sa să fie mai mari. Apoi, algoritmul este recursiv aplicat la cele două subliste astfel încât fiecare sublistă să fie sortată separat.

În Quick sort, pivotul este de obicei ales ca fiind elementul din mijlocul listei, dar poate fi și ales aleatoriu sau ca fiind primul sau ultimul element. În timpul procesului de sortare, elementele sunt mutate în stânga sau în dreapta pivotului, astfel încât pivotul să ajungă la poziția sa finală în lista sortată.

Una dintre strategiile de alegere a pivotului este alegerea mediei din cele trei elemente: primul, ultimul și elementul din mijlocul listei de sortat. Această strategie poate îmbunătăți performanța Quick sort-ului în cazul unor liste de intrare care nu sunt complet aleatoare sau când elementele sunt aproape de sortate. În schimb, în cazul unor liste complet aleatoare, alegerea pivotului în acest mod poate duce la performanțe mai proaste decât alegerea pivotului aleatoriu.

Quick sort este un algoritm eficient de sortare, cu o complexitate medie de timp de O(n log n). Cu toate acestea, în cazul celor mai proaste scenarii, poate avea o complexitate de timp de O(n^2). Pentru a evita acest lucru, sunt utilizate diferite strategii de alegere a pivotului, precum alegerea pivotului aleatoriu sau mediana dintre primele trei elemente ale listei.

Quick sort este utilizat frecvent în practică datorită performanței bune și a ușurinței de implementare. De asemenea, poate fi îmbunătățit prin utilizarea unor metode de optimizare, cum ar fi inserarea sortării pentru listele mai mici sau utilizarea unui algoritm de sortare diferit pentru subliste de dimensiuni mici.

6. Algoritmul Default

C++ este un limbaj de programare care oferă mai multe metode de sortare, dar una dintre cele mai simple și eficiente metode este sortarea prin intermediul funcției std::sort().

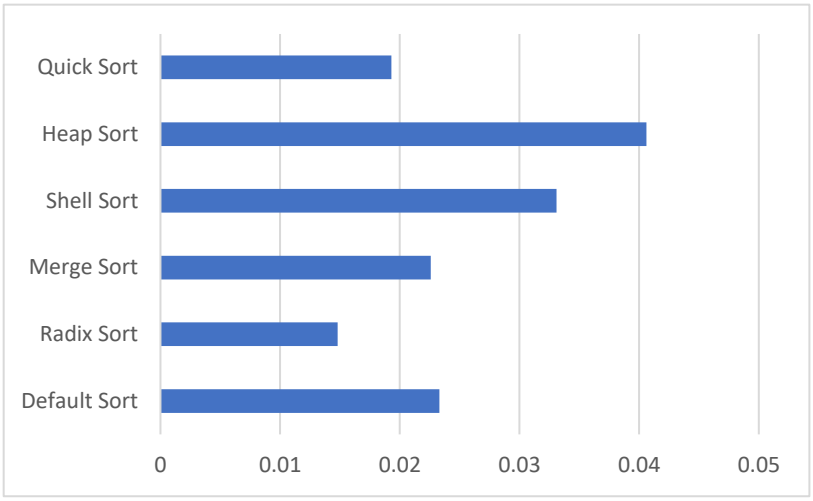
Această funcție este disponibilă în biblioteca standard a limbajului C++ și poate fi utilizată pentru a sorta un vector sau o listă de elemente în ordine crescătoare sau descrescătoare. Funcția std::sort() utilizează de obicei algoritmul introsort, care combină trei algoritmi de sortare: quick sort, heap sort și insertion sort.

Pentru a utiliza funcția std::sort(), trebuie să specificăm intervalul de elemente pe care dorim să îl sortăm, adică primul și ultimul element din vector sau listă. Funcția va sorta automat elementele din intervalul specificat în ordine crescătoare, așa cum este specificat implicit. Dacă dorim să sortăm în ordine descrescătoare, putem specifica acest lucru prin intermediul unui comparator, prin care specificăm funcția de comparație utilizată în sortare.

Compararea Algoritmilor

1. Comaparea algoritmilor in cazul vectorilor aleatorii

Algoritm	Timp
Default Sort	0.0233s
Radix Sort	0.0148s
Merge Sort	0.0226s
Shell Sort	0.0331s
Heap Sort	0.0406s
Quick Sort	0.0193s



2. Compararea algoritmilor in cazul vectorilor sortati descrescator

Algoritm	Timp
Default Sort	0.0073s
Radix Sort	0.0156s
Merge Sort	0.0113s
Shell Sort	0.0196s
Heap Sort	0.0316s
Quick Sort	0.0053s