ALGORITMI SORTARE

GRUIA GABRIEL - 131

OVERVIEW

1. STL Sort
2. Bubble Sort
3. Count Sort
4. Radix Sort
5. Merge Sort
6. Quick Sort

STL Sort

Algoritm *nativ* de sortare, cu o *complexitate de timp mare* - O(nlogn), ce se bazeaza pe *comparatii* intre elemente; *inceata* pentru sortarea listelor cu numar mare de elemente; nu conteaza *elementul maxim* pentru timpul de rulare.

TESTE:

1. *randomArray: N = 10000, MAX = 1000 - exec. time: 0 s*
2. *ascSortedArray: N = 10000, MAX = 1000 - exec. time: 0 s (best case)*
3. *descSortedArray: N = 10000, MAX = 1000 - exec. time: 0 s*
4. *constArray: N = 10000, MAX = 1000 - exec. time: 0 s*
5. *almostSortedArray: N = 10000, MAX = 1000 - exec. time: 0 s*

1. *randomArray: N = 10000000, MAX = 1000 - exec. time: 12 s*
2. *ascSortedArray: N = 10000000, MAX = 1000 - exec. time: 0 s (best case)*
3. *descSortedArray: N = 10000000, MAX = 1000 - exec. time: 7 s*
4. *constArray: N = 10000000, MAX = 1000 - exec. time: 8 s*
5. *almostSortedArray: N = 10000000, MAX = 1000 - exec. time: 7 s*

BUBBLE Sort

Algoritm *simplu de implementat*, insa cu o *complexitate de timp mare* - O(n^2), ceea ce o face *impractica* pentru sortarea listelor cu numar mare de elemente; nu conteaza *elementul maxim* pentru timpul de rulare.

TESTE:

1. *randomArray: N = 100, MAX = 1000 - exec. time: 0 s*
2. *ascSortedArray: N = 100, MAX = 1000 - exec. time: 0 s (best case)*
3. *descSortedArray: N = 100, MAX = 1000 - exec. time: 0 s*
4. *constArray: N = 100, MAX = 1000 - exec. time: 0 s*
5. *almostSortedArray: N = 100, MAX = 1000 - exec. time: 0 s*

1. *randomArray: N = 10000, MAX = 1000 - exec. time: 2 s*
2. *ascSortedArray: N = 10000, MAX = 1000 - exec. time: 0 s (best case)*
3. *descSortedArray: N = 10000, MAX = 1000 - exec. time: 0 s*
4. *constArray: N = 10000, MAX = 1000 - exec. time: 4 s*
5. *almostSortedArray: N = 10000, MAX = 1000 - exec. time: 0 s*

COUNT Sort

Algoritm ce creaza un array de frecventa de (MAX + 1) elemente, in care stocheaza nr. de aparitii a fiecarui element din array-ul initial; are o *complexitate de timp mica* - O(n + k), ceea ce o face *practica* pentru sortarea listelor cu numar mare de elemente aflate intr-un range; CONTEAZA *elementul maxim* la rulare, acesta determinand lungimea array-ului auxiliar de sortare.

TESTE:

1. *randomArray: N = 1000000, MAX = 100000 - exec. time: 0 s*
2. *ascSortedArray: N = 1000000, MAX = 100000 - exec. time: 0 s*
3. *descSortedArray: N = 10000, MAX = 100000 - exec. time: 0 s*
4. *constArray: N = 1000000, MAX = 100000 - exec. time: 0 s (best case)*
5. *almostSortedArray: N = 1000000, MAX = 100000 - exec. time: 0 s*

1. *randomArray: N = 10000000, MAX = 10000000 - exec. time: 0 s*
2. *ascSortedArray: N = 10000000, MAX = 10000000 - exec. time: 0 s*
3. *descSortedArray: N = 10000000, MAX = 10000000 - exec. time: 0 s*
4. *constArray: N = 10000000, MAX = 10000000 - exec. time: 0 s (best case)*
5. *almostSortedArray: N = 10000000, MAX = 10000000 - exec. time: 0 s*

RADIX Sort

Algoritm *non-comparativ* ce se bazeaza pe ordonarea repetata de *tip bucket* a elementelor in functie de cea mai semnificativa/ nesemnificativa cifra, insa cu o *complexitate de* O((n+b) * logb(MAX)): b = baza, ceea ce o face *practica* pentru sortarea listelor cu numar mare de elemente; CONTEAZA *elementul maxim* pentru timpul de rulare.

TESTE:

1. *randomArray: N = 10000000, MAX = 10000 - exec. time: 1 s*
2. *ascSortedArray: N = 10000000, MAX = 10000 - exec. time: 1 s*
3. *descSortedArray: N = 10000000, MAX = 10000 - exec. time: 1 s*
4. *constArray: N = 10000000, MAX = 10000 - exec. time: 1 s*
5. *almostSortedArray: N = 10000000, MAX = 10000 - exec. time: 1 s*

1. *randomArray: N = 10000000, MAX = 10000000 - exec. time: 35 min*
2. *ascSortedArray: N = 10000000, MAX = 10000000 - exec. time: 34 min*
3. *descSortedArray: N = 10000000, MAX = 10000000 - exec. time: 34 min*
4. *constArray: N = 10000000, MAX = 10000000 - exec. time: 34 min*
5. *almostSortedArray: N = 10000000, MAX = 10000000 - exec. time: 35 min*

MERGE Sort

Algoritm de tip *divide et impera* ce imparte un array in 2 sub-array-uri de lungimi egale in mod repetat pana ce sub-array-urile nu mai pot fi impartite, apoi le interclaseaza folosind recursivitatea, cu o *complexitate de* O(nlogn), ceea ce o face *practica* pentru sortarea listelor cu numar mare de elemente; nu conteaza *elementul maxim* pentru timpul de rulare.

TESTE:

1. *randomArray: N = 1000000, MAX = 1000   - exec. time: 0 s*
2. *ascSortedArray: N = 1000000, MAX = 1000   - exec. time: 0 s (best case)*
3. *descSortedArray: N = 1000000, MAX = 1000   - exec. time: 0 s*
4. *constArray: N = 1000000, MAX = 1000   - exec. time: 0 s*
5. *almostSortedArray: N = 1000000, MAX = 1000   - exec. time: 0 s*

1. *randomArray: N = 10000000, MAX = 1000 - exec. time: 1 s*
2. *ascSortedArray: N = 10000000, MAX = 1000 - exec. time: 1 s (best case)*
3. *descSortedArray: N = 10000000, MAX = 1000 - exec. time: 1 s*
4. *constArray: N = 10000000, MAX = 1000 - exec. time: 1 s*
5. *almostSortedArray: N = 10000000, MAX = 1000 - exec. time: 1 s*

QUICK Sort

Algoritm de tip *divide et impera* ce imparte un array in 2 sub-array-uri in functie de un pivot (ales de mine ca mediana din 3) in mod repetat pana ce sub-array-urile nu mai pot fi impartite, apoi le sorteaza folosind recursivitatea, cu o *complexitate medie de* O(nlogn), ceea ce o face *practica* pentru sortarea listelor NESORTATE cu numar relativ mic de elemente; nu conteaza *elementul maxim* pentru timpul de rulare.

TESTE:

1. *randomArray: N = 1000000, MAX = 1000 - exec. time: 0 s*
2. *ascSortedArray: N = 1000000, MAX = 1000 - exec. time: 0 s (best case)*
3. *descSortedArray: N = 1000000, MAX = 1000 - exec. time: 0 s*
4. *constArray: N = 1000000, MAX = 1000 - exec. time: 0 s*
5. *almostSortedArray: N = 1000000, MAX = 1000 - exec. time: 0 s*

1. *randomArray: N = 10000000, MAX = 1000 - exec. time: 8 s*
2. *ascSortedArray: N = 10000000, MAX = 1000 - exec. time: too slow*
3. *descSortedArray: N = 10000000, MAX = 1000 - exec. time: too slow*
4. *constArray: N = 10000000, MAX = 1000 - exec. time: 10 s*
5. *almostSortedArray: N = 10000000, MAX = 1000 - exec. time: too slow*

CONCLUZII

Bubble vs. STL

- mai incet si impractic pe teste mari

Count vs. STL

+ mai rapid pe teste mari
- impractic pe teste cu range mare

Radix vs. STL

+ mai rapid pe teste mari de numere mici sau teste mici cu numere mari

Merge vs. STL

+ mai rapid
- foloseste mai multa memorie

Quick vs. STL

+ mai rapid pe teste nesortate
- impractic pe teste sortate

PREFERINTE

1. teste mari de numere in range mic - COUNT
2. teste mari de numere in range mare - MERGE
3. teste mici de numere - QUICK

END