Algoritmi de sortare

Nume: Titirigă Tiberiu-Nicolae
Grupa: 133

Algoritmii implementați

Shell Sort

Merge Sort

Radix Sort

Counting Sort

Quick Sort

Shell sort

Shell sort este o generalizare a Insertion Sort-ului, care sortează elementele situate la distanta apoi se reduce succesiv acest interval dintre numere. In funcție de secvența folosita, complexitatea se modifica, fapt observat in urma numeroaselor teste efectuate. Secventele analizate sunt *Knuth* **și** *Hibbard***.**

Knuth Gap

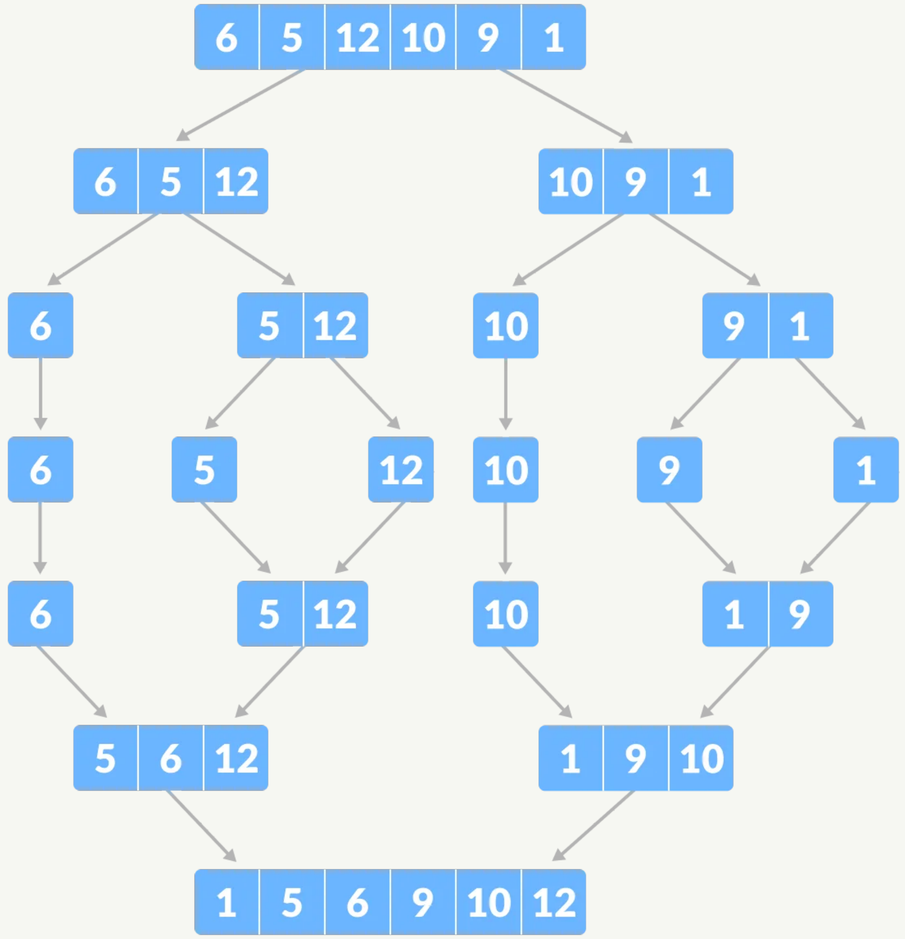
- Secvența e de tipul (3^k-1)/2: 1, 4, 13, 40, 121,... , continuând pana la maxim n/3.
- Pentru generarea acestor numere am folosit o funcție, pastrandu-le într-un vector pe care îl parcurgem de la coada la cap in algoritmul implementat.
- Pentru N<=10^7, complexitatea data de aceasta secvența este aproape similară cu cea oferită de *Hibbard*, insa pentru 10^8 numere, aceasta creste cu 15-20%, in funcție de test.

Hibbard's Gap

- Acest Gap este de forma 2^k-1: 1, 3, 7, 15,... .
- Generarea se face cu aceeași funcție in care este schimbată recurenta șirului, vectorul parcurgandu-se la fel, de la sfârșit spre început.
- Cu toate ca worst-case complexity este N^(3/2), similar cu secvența anterioară, pentru N=10^8 acesta începe sa devină vizibil mai eficient. Comparativ cu sortarea din STL, algoritmul pt N mic este de aproximativ 5 ori mai lent, insa odată cu creșterea lui N, spre 10^8, acesta devine aproximativ de 10 ori mai lent.

Merge sort

- Merge Sort este un algoritm cunoscut, bazat pe principiul Divide and Conquer. Vectorul este împărțit in subvectori care sunt sortati apoi sunt recombinati pentru a oferi soluția ordonata.
- Complexitatea de timp este O(n*logn), iar cea de spațiu este O(n).

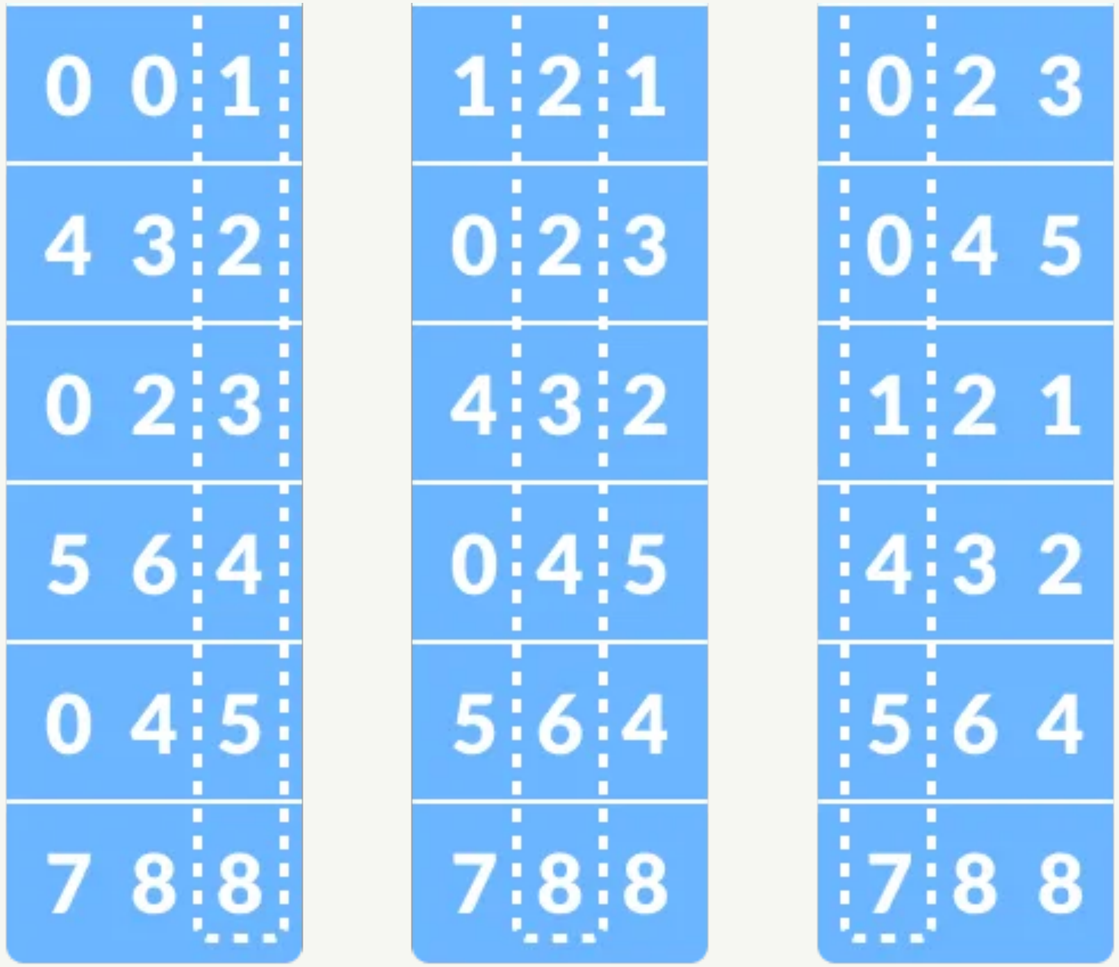


Merge Sort

In toate testele efectuate, acest algoritm este cel mai lent, fapt datorat alocării repetate de memorie prin crearea vectorilor.

Radix sort LSD

- Algoritmul sortează numerele pe rând, in funcție de cifre. Least Significant Digit(LSD) presupune ordonarea mai întâi după cifra unităților, apoi a zecilor, repetându-se pana la prima cifra a celui mai mare număr.
- In ciuda principiului similar, diversitatea algoritmului consta in variația de baze in care poate fi folosit.



Shell Sort

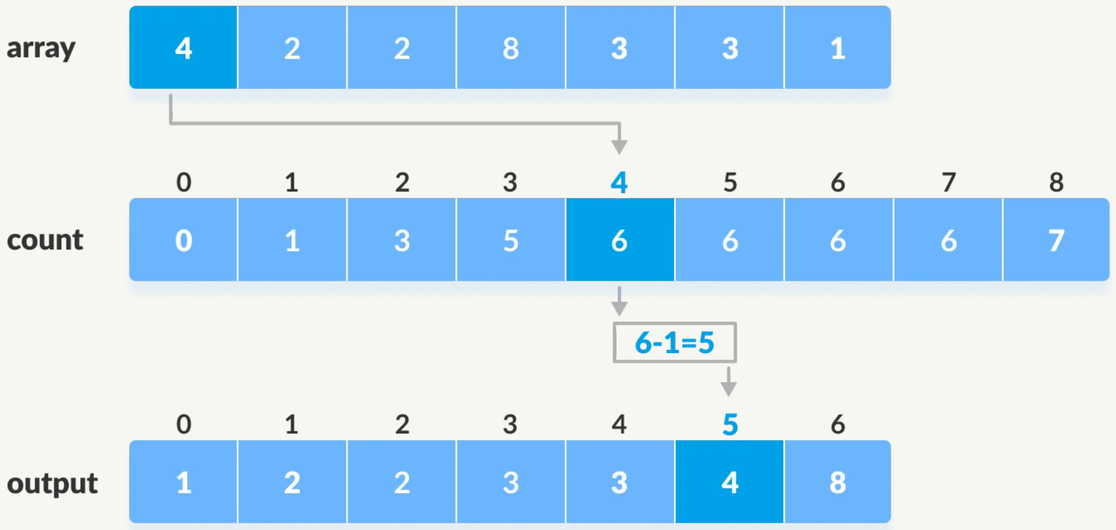
In proiectul realizat, bazele studiate sunt 2^16 și baza 10, cea standard. In ciuda deferentei mari de baze, acest algoritm oferă o eficienta foarte similară, pentru baza 2^16 oferind un timp de execuție ușor mai bun.

Algoritmul este mult mai bun decât Shell Sort sau Merge Sort, fiind destul de apropiat de Quick Sort, fiind de aproximativ 3 ori mai lent decât sortarea din STL.

Complexitatea algoritmului este O(n+k).

Counting Sort

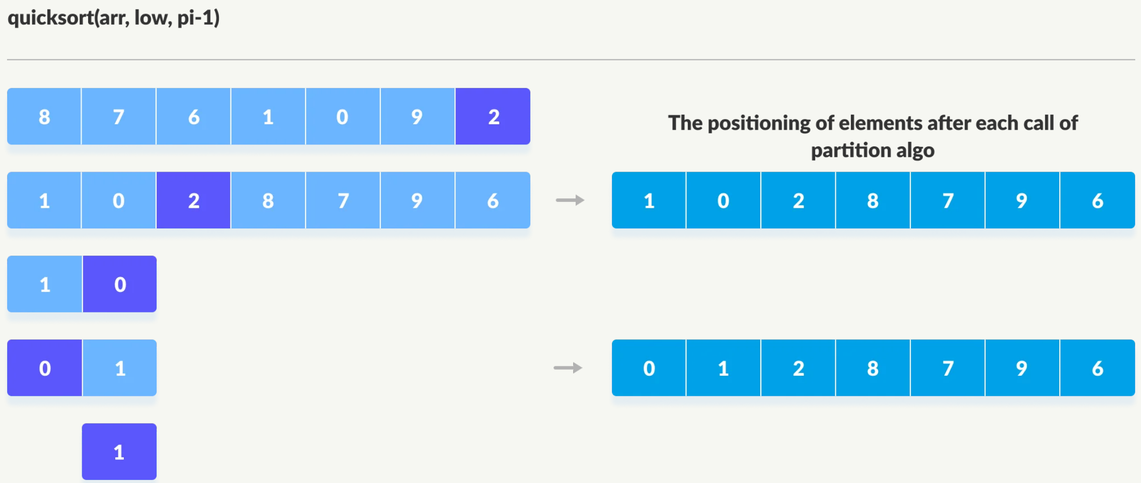
- Acest algoritm se bazează pe numărarea aparițiilor elementelor unui vector și afisarea lor prin parcurgerea vectorului de ocurență.
- Desi complexitatea de timp este buna, O(n+k), pentru N foarte mare, devine o problema complexitatea de spațiu, care este O(N).
- In implementarea efectuată, acesta este singurul algoritm care se apropie de complexitatea sortarii STL



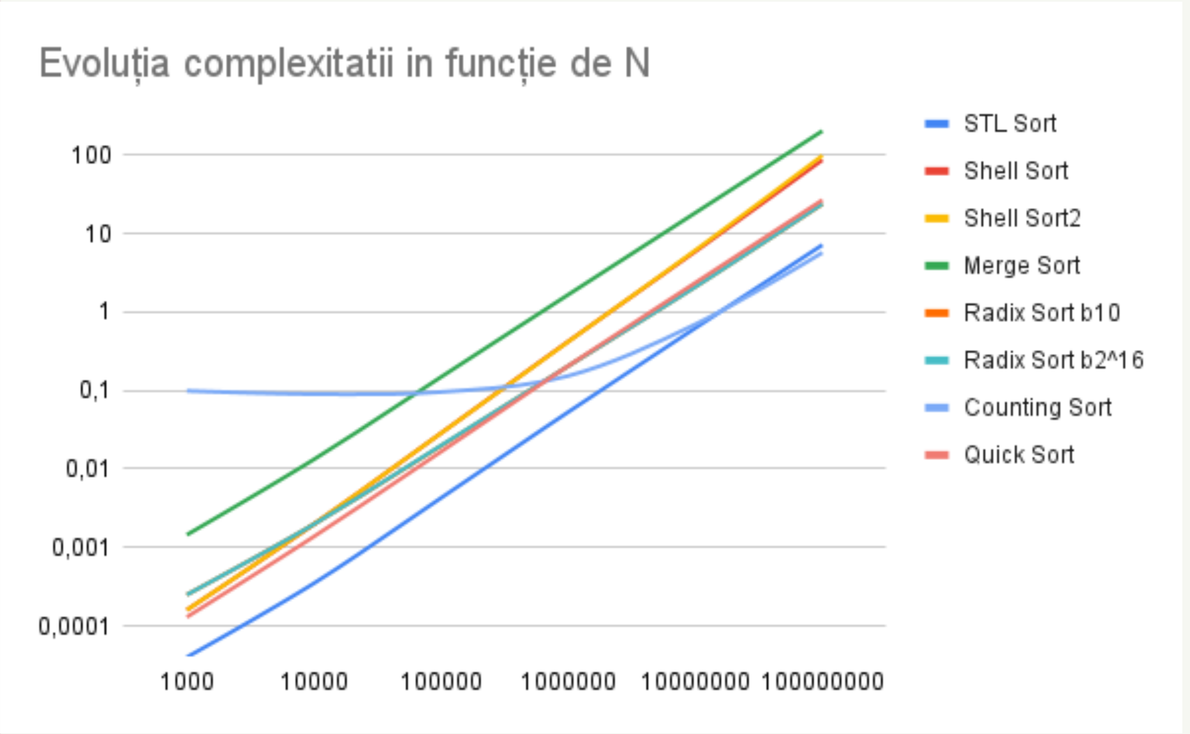
Quick Sort

- Sortarea aceasta se face prin alegerea unui pivot, in cazul meu îl aleg random, și se aseaza elementele mai mici in stânga pivotului, iar elementele mai mari in dreapta, acest proces repetandu se pana se ordonează crescător toate subsecventele generate de pivot.

- Complexitatea creste dacă pivotul ales este de fiecare data mai mare decât restul elementelor, in acest caz devenind O(n^2), insa average, complexitatea acestui algoritm este O(n*logn).



- In graficul prezentat se poate observa evoluția timpului realizat de fiecare algoritm. Se poate observa ca algoritmii de același tip sunt similari. Singurul care se apropie de STL Sort este Counting Sort-ul, pentru N mai mare.
- Se poate observa ca in toate cazurile Merge Sort este cel mai lent. Cu toate acestea, Shell Sort se apropie de o eficienta similară pentru N foarte mare.
- Testele au fost efectuate pentru numere de maxim Max<=10^6.



```
Test 7, N=10000000, reverse sorted:
Stl Sort: 0.01829
Shell sort, gap (3^k-1)/2: 1.14396sec  //  Sortat cu succes
Shell sort, gap (2^k-1): 1.15338sec  //  Sortat cu succes
MergeSort: 17.52612sec  //  Sortat cu succes
RadixSort baza 10: 2.11877sec  //  Sortat cu succes
RadixSort baza 2^16: 2.10811sec  //  Sortat cu succes
CountingSort: 0.43338sec  //  Sortat cu succes
QuickSort: 1.67634sec  //  Sortat cu succes
```

```
Test 8, N=10000000, sorted:
Stl Sort: 0.01076
Shell sort, gap (3^k-1)/2: 0.76702sec  //  Sortat cu succes
Shell sort, gap (2^k-1): 0.74398sec  //  Sortat cu succes
MergeSort: 16.88583sec  //  Sortat cu succes
RadixSort baza 10: 2.13363sec  //  Sortat cu succes
RadixSort baza 2^16: 2.15616sec  //  Sortat cu succes
CountingSort: 0.43399sec  //  Sortat cu succes
QuickSort: 1.64268sec  //  Sortat cu succes
```

```
----------------------------------------------------------------------
Test 5, N = 10000000:
Stl Sort: 0.62429
Shell sort, gap (3^k-1)/2: 6.00586sec  //  Sortat cu succes
Shell sort, gap (2^k-1): 6.11404sec  //  Sortat cu succes
MergeSort: 18.35119sec  //  Sortat cu succes
RadixSort baza 10: 2.09614sec  //  Sortat cu succes
RadixSort baza 2^16: 2.09114sec  //  Sortat cu succes
CountingSort: 0.69687sec  //  Sortat cu succes
QuickSort: 2.35572sec  //  Sortat cu succes
```

- In testele anterioare efectuate, întâi numerele sunt sortate descrescător, apoi crescător, și in dreapta, generate random.
- Putem observa ca Shell Sort este cam de 5 ori mai rapid dacă elementele sunt deja sortate doar ca descrescător.
- Merge Sort-ul iei păstrează aproape același timp de execuție, la fel și Radix Sort și Counting Sort.

- Doar Quick-Sort-ul este mai rapid, insa nu cu mult.
- De asemenea, STL Sort este de doua ori mai rapid la ordonarea numerelor crescătoare, decât la cea a celor descrescătoare, și de 60 de ori mai rapida decât a celor generate random.

Sfârșit