**QUICKSORT**

***AVANTAJE:***

- necesită, în cazul mediu, O(log2 n) memorie suplimentară, mult mai redusă decât sortarea prin interclasare.

- cea mai bună alegere când viteza este foarte importantă, indiferent de dimensiunea setului de date

***DEZAVANTAJE:***

* poate duce la umplerea stivei când se utilizează seturi largi de date
* în cazul cel mai defavorabil performanța metodei scade catastrofal (când la fiecare partiționare este selectată cea mai mare (cea mai mica) valoare ca și pivot, dar și când avem duplicate)
* algoritmul de sortare rapidă este instabil
* are performanțe slabe în cazul sortărilor banale, atunci când operează asupra unor liste aproape ordonate.

Obs: Are performanțe deosebite în cazul tablourilor dezordonate.

Timpul de rulare al acestui algoritm depinde de partiționare: dacă este sau nu realizată într-un mod balansat. Dacă partiționarea este una balansată, algoritmul are timp de rulare Θ(n ∗ log n), în caz contrar ordinul de timp fiind unul pătratic.

Cel mai defavorabil caz: procedura de partiționare produce o regiune cu n − 1 elemente și alta cu 1 element.

Cel mai favorabil caz: procedura de partiționare ar produce două subșiruri de mărime n/2.

Cazul mediu: nici perfect balansate și nici total nebalansate.

**COUNTINSORT**

Este un algoritm de sortare util pentru situația de numere pozitive întregi care nu depășesc o anumită limită superioară.

Cel mai bun caz de a utiliza acest algoritm: avem multe valori de sortat care se repetă/sunt uniform distribuite peste un rang dat.

O alegere bună de sortare dacă: lista conține numere întregi/limita superioară se cunoaște/cele mai multe dintre elemente (mai mici ca limita dată) sunt prezente în șir/dacă memoria suplimentară nu este o prioritate.

COMPLEXITATE

Θ(n + k), n lungime input, k limita superioară (valoarea maximă din input)

De multe ori, k este O(n), de aceea counting sort se consideră a avea complexitate O(n) timp și spațiu.

Obs. Counting sort funcționează cel mai bine dacă limita superioară a elementelor care urmează să fie sortate este mai mică decât numărul de itemi care urmează să fie sortați. (k = O(n))

**CARACTERISTICI SORTĂRI**

**Metoda selecției:**

* cel mai intuitiv mod de sortare (+)
* chiar și cel mai perfect sortat input, necesită minim o parcurgere (-)
* se folosește când faci ceva rapid și "dirty"(+)
* in-place(+)

**Metoda bulelor:**

* util pentru seturi mici de date (+)
* ușor de implementat (+)
* in-place (+)
* eficiență redusă (-)

**Sortare Interclasare:**

* poate fi folosit pentru date de orice mărime (+)
* stable (+)
* nu e in-place (-)
* se folosește intens pentru sortarea de linked list, unde accesul secvențial este necesar

**QuickSort:**

* rapid în cazuri de date random, (+)
* in-place (+)
* rapid (ciclul repetitiv din interior este foarte scurt și se poate optimiza foarte bine) (+)
* are performanțe deosebite în cazul tablourilor dezordonate (+)
* foarte nesigur pentru cazul defavorabil (O(n^2)) (-)
* performanțe slabe pentru tablourile banale (-)

**Counting sort:**

* o alegere bună de sortare dacă: lista conține numere întregi sau dacă limita superioară se cunoaște (+)
* folosește spațiu suplimentar (-)
* trebuie modificat pentru numere negative

**BucketSort**:

* poate fi folosit ca și un algoritm "extern" de sortare (+)
* rezultate foarte bune atunci când datele sunt uniform distribuite (+)
* complexitatea crește foarte mult atunci când mărim numărul de inputuri