CIORNA:

* Algoritmi sortare teorie resurse:
  + STANFORD COURSES
  + Introduction to Algorithms 3rd edition
  + SKIENNA
* CASE STUDY:

Compilari timp pentru liste[10,100,1000,10^4, 10^5] pentru numere random pentru(operations/second on my CPU):

* + Bubble
  + Insertion
  + selection
  + Merge
  + Quick
  + Quick3
    - Numar interschimbari/numar comparari/timp
* Observari
* References/ External links
* Graphs(performance comparison)
* SCOPE of the study
* Conclusion

We dont have better algorithms, we just have more data

**Nu avem algoritmi mai buni, ci avem mai multe date.**

Scopul lucrarii:

Lucrarea dată are drept scop studiul și analiza printr-o comparație teoretică și experimentală a metodelor de sortare, în special cele studiate sunt: Bubble sort, Insertion sort, Selection sort, Quicksort, Mergesort și Countsort. Într-u studierea aspectelor teoretice, am prezentat o scurtă descriere a fiecărei sortări, principiile, ordinul de complexitate. Pentru partea experimentală, am executat funcții sortare în Python, de 10n elemente, pentru n=1 până la 4(pentru cele de complexitate n2) și pentru n până la 6 sau 7 pentru cele care am putut scoate un timp favorabil(<10 minute). Pentru ca rezultatele să fie cât mai acurate, am repetat experimentul de 10 ori pentru fiecare metodă și număr de elemente. Elementele au fost generate aleator în același program, cu valori cuprinse de la 1 la numărul de valori al tabloului. Rezultatele experimentelor aferente sunt reprezentate în grafice și tablouri. Alte scopuri ale acestei lucrări sunt: să exploreze, să verifice, să valideze, să descrie aspectul de timp de compilare sau/și numărul de operații(comparări și interschimbări) efectuate per algoritm, printr-un studiu empiric.

Obiectiv:

Demonstrarea cu timpi și date concrete a diferențelor între algoritmii usuali de sortare.

Compararea teoretică și experimentală a metodelor de sortare menționate mai sus.

Problema sortării este una trivială, însă aceasta înconjoară societatea și este parte a vieții cotidiene. Astfel, din punct de vedere a programării, în special sortării unor elemente în dependență de valoarea acestora, am efectuat o lucrare în care am evidențiat diferențele dar și similitudinile algoritmilor. Pe piață sunt multe metode de aranjare a unor elemente, însă la baza comparării acestora stau principiile timpului de execuție(complexitatea timpului) și cantității de memorie necesară sortării. Dat fiind faptul că este nevoie de o efiecentizare a metodelor de sortare, în această lucrare s-au analizat doar cele care au avut gradul de complexitate de maxim O(n2).

Standarde Academice:

~~PG 103-104 SKIENA~~

Sortarea este una dintre cele mai studiate probleme în informatică. Literalmente, zeci de algoritmi sunt cunoscuți, fiecare având un avantaj în comparație cu altul.

\*Precizare: Toate sortările și descrierea implementării lor a fost făcută din punct de vedere al aranjării crescătoare.

Metodologia

* Bubble Sort

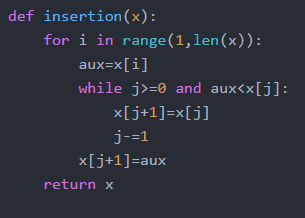
<teoria>

BubbleSort este un algoritm popular, însă ineficient. Acesta are complexitatea în cazul nefavorabil și mediu de O(n2), iar de aceea nu este folosit la sortarea unui număr mare de elemente, decât doar dacă sunt puține elemente care nu sunt în ordinea dorită.

Principiul: Începe de la primul element al tabloului și interschimbă repetat elemente alăturate care nu sunt in ordine.

pg40 comen;</teorie>

* Insertion Sort

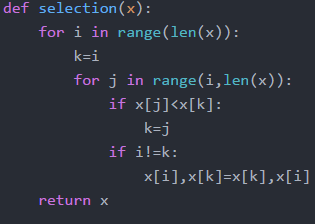
<teoria>Algoritmul de sortare prin inserție este unul eficient când vine vorba de un număr mic de valori. Începem cu o “parte stângă” liberă(când de exemplu avem în mâna dreapta câteva cărți iar în stângă le aranjăm). Valorile preluate din input sunt așezate într-un tablou(array). Astfel preluăm câte un element(key/valoarea auxiliara)din tablou și o aranjăm în poziția corectă în partea I din tablou. Pentru a găsi poziția corectă a elementului, comparăm valoarea cu cele deja existente în tabel, de la stânga la dreapta. Mereu,elementele din partea stângă a tabloului sunt sortate, iar cele din partea dreaptă nu. O altă metodă care folosește acest algoritm este Shell sort.

Algoritm Python:

</teorie>

* Selection sort

<teoria>Sortarea prin selecție are o complexitate O(n2), ceea ce îl face ineficient pentru un număr mare de elemente. El se remarcă însă prin simplicitatea sa.

Principiul de lucru: Tabloul este la fel “împărțit” în două subtablouri; algoritmul găsește valoarea minimă din tabel, o interschimbă cu prima poziție, și repetă acest pas până la finalizarea listei. O altă metodă mai eficientă care folosește același principiu este Shaker Sort, care în aceeași iterație plasează minimul la începutul tabloului și maximul la sfârșitul ei.

~~saylor.org;ccis2k.or~~g;theijes.com;ijser.in;

Algoritm Python:

</teorie>

* MergeSort – sortarea utilizând principiul “Divide et Impera”

<teoria>

+pg120 skiena

* + Divizarea problemei într-un număr de subprobleme.
  + Stăpânirea subproblemelor rezolvându-le recursiv. Dacă dimensiunea subproblemei este rezonabil de mica, rezolvarea acesteia se face într-o maniera simplă.
  + Combinarea soluțiilor subproblemelor în soluția problemei originale.

</teorie>

* QuickSort

<teoria> Sortarea rapidă(QuickSort), ca și Sortarea prin Interclasare(MergeSort) aplică principiul “Divide et Impera”, paradigmă explicată anterior:

* + Divizarea: partiționarea tabloului în 2 subtablouri, astfel încât toate elementele din A[s..p-1] sunt mai mici sau egale cu elemental A[p], iar cele din A[q+1..d] sunt mai mari sau egale. Elementul A[p] este numit pivot.
  + Conquer: Sortarea celor două subtablouri recursive, folosind aceeași metodă.
  + Combinarea: Având în vedere că subtablourile sunt deja sortate, la combinare nu mai este nevoie de o rearanjare. Astfel, tabloul A[s..d] este sortat.

QuickSort, pentru cazul nefavorabil are o complexitate de O(n2) – în cazul în care pivotul ales nu este o alegere potrivită(nu împarte tabloul în 2 subtablouri mai mult sau mai puțin egale); pentru cazul favorabil și mediu complexitatea este O(n log n).

\*Observație: Acest algoritm are nevoie și de un spațiu de memorie O(log n)

pg123 skiena~~;+170 comen;~~ </teorie>

* Count sort(Instabil)

<teoria>

Sortarea prin numărare este o metodă instabilă de sortare(elementele egale din tabloul original ar putea să nu fie în aceeași ordine în cel final). Acesta are complexitatea O(n+k) unde n este numărul de elemente din tablou, iar k este valorea maximă din tablou.

Princiupiul de funcționare: Se parcurge tabloul pentru găsirea valorii maxime. Apoi se crează un tablou de k valori, iar la parcurgerea tabloului original, la fiecare valoare din acesta, se incrementează valorea indexului potrivit din cel de-al doilea tabel. După finalizare se întoarce al doilea tablou, care conține elemente cu valori diferite de 0.

Algoritm/pseudocod; Principiul de funcionare; complexitate; </teorie>

ANALIZA PERFORMANȚEI DE SORTARE:

<Grafurile si tabelele>

COMENTAREA REZULTATELOR

CONCLUIZA CELUI MAI BUN ALGORITM

<concluzia>

</concluzia>

REFERINȚE

* International Journal for Research in Applied Science & EngineeringTechnology (IJRASET)©IJRASET 2015: All Rights are Reserved201Performance Analysis of Sorting Algorithms with C#
* The International Arab Journal of Information Technology, Vol. 7, No. 1, January 2010
* The International Journal Of Engineering And Science (IJES)
* International Journal of Scientific Engineering and Research (IJSER)
* Introduction to algorithms (3rd edition) – Thomas H. Cormen; Charles E. Leiserson; Ronald L. Rivest; Clifford Stein
* The Algorithm Design Manual (2nd edition) – Steven. S. Skiena