O privire generală asupra eficienței algoritmilor de sortare elementari în programare

Vlăduț-Gabriel Ghelbere
Departamentul de Informatică,
Facultatea de matematică si informatică,
Universitatea de Vest, Timişoara
Email: `vladut.ghelbere98@e-uvt.ro`

29 aprilie 2019

Cuprins

Rezumat

În cele mai multe cazuri, fișierele din memorie trebuie sortate. Există la bază două motive pentru această necesitate: pe de-o parte anumiți algoritmi necesită ca fișierele cu care lucrează să fie deja sortate într-un anumit mod și pe de altă parte, într-un fișier de date cu "random access" se pot găsi fișierele căutate mult mai rapid. Această problemă a fost de asemenea abordată și de Stefan Hougardy [5]

Algoritmii de sortare au rămas un subiect foarte abordat în informatică, de la începuturile acesteia dorind să se obțină performanță maximă. Din fericire, această performanță a devenit posibilă datorită algoritmilor de sortare rapizi și eficienți cum ar fi: heap sort, merge sort și alți algoritmi de sortare.

Pe parcursul acestei lucrări vom analiza performanța următorilor algoritmi de sortare: Bubble Sort, Selection Sort, Insertion Sort, Merge Sort și Quick Sort.

I Introducere

1 Prezentarea Conceptului

Se consideră un set finit de obiecte, fiecare având asociată o caracteristică, numită cheie, ce ia valori într-o mulțime pe care este definită o relație de ordine. Sortarea este procesul prin care elementele setului sunt rearanjate astfel încât cheile lor să se afle într-o anumită ordine.

Ex 1. Considerăm setul de valori iintregi: (5,8,3,1,6). În acest caz cheia de sortare coincide cu valoarea elementului. Prin sortare crescătoare se obține setul (1,3,5,6,8) iar prin sortare descrescătoare se obține (8,6,5,3,1).

Ex 2. Considerăm un tabel constând din nume ale studenților și note (Tabela 1):

Vlad	10
Andrei	9
Denis	8
George	9

Tabela 1

În acest caz cheia de sortare poate fi numele sau nota. Prin ordonare crescătoare după nume se obține (Tabela 2):

Andrei	9
Denis	8
George	9
Vlad	10

Tabela 2

...iar prin ordonare descrescătoare după notă se obține (Tabela 3):

Vlad	10
Andrei	9
George	9
Denis	8

Tabela 3

Pentru mai multe exemple legate de modul de funcționare al algoritmilor de sortare(vezi S. Skiena [8] sau Cormen [3])

2 Caracterizarea metodelor de sortare

Metodele de sortare pot fi caracterizate prin:

- O metodă de sortare este considerată stabilă dacă ordinea relativă a elementelor ce au aceeași valoare a cheii nu se modifică în procesul de sortare. De exemplu dacă asupra tabelului cu note din textbfEx. 2 se aplică o metodă stabilă de ordonare descrescătoare după notă se obține ((Vlad,10), (George,9), (Andrei,9), (Denis,8)) pe când dacă se aplică una care nu este stabilă se va putea obține ((Vlad,10), (Andrei,9), (George,9), (Denis,8)).

- **Naturalețe**
  O metodă de sortare este considerată naturală dacă numărul de operații scade odată cu distanța dintre tabloul inițial ¸si cel sortat. O măsură a acestei distanțe poate fi numărul de inversiuni al permutării corespunzătoare tabloului inițial.

- **Eficiență**
  O metodă este considerată eficientă dacă nu necesită un volum mare de resurse. Din punctul de vedere al spațiului de memorie o metodă de sortare pe loc este mai eficientă decât una bazată pe o zonă de manevră de dimensiunea tabloului. Din punct de vedere al timpului de execuție este important să fie efectuate cât mai puține operații. În general, în analiză se iau în considerare doar operațiile efectuate asupra elementelor tabloului (comparații și mutări). O metodă este considerată optimală dacă ordinul său de complexitate este cel mai mic din clasa de metode din care face parte.

- **Simplitate**
  O metodă este considerată simplă dacă este intuitivă și ușor de înțeles.

Primele două proprietăti sunt specifice algoritmilor de sortare pe când ultimele sunt cu caracter general. În continuare vom considera câteva metode elementare de sortare caracterizate prin faptul că sunt simple, nu sunt cele mai eficiente metode, dar reprezintă punct de pornire pentru metode avansate. Pentru fiecare dintre aceste metode se va prezenta o descriere generală și un experiment.

Pentru mai multe informații în legătură cu caracterizarea metodelor de sortare (vezi Knuth [4])

II Implementare

In continuare vom urmări cum funcționează fiecare algoritm în teorie:

1 Bubble Sort

Bubble sort, după cum îi spune și numele, funcționează prin mutarea cheilor "bulelor" din dreapta listei în stânga. Începe prin luarea unei chei de la final și mutând-o în stânga atâta timp cât aceasta este mai mică decât orice cheie pe lângă care trece. Atunci când ajunge la o cheie mai mică decât ea, se schimbă în cheia respectivă și o mută în stânga listei. În acest mod, cu fiecare iterație, cea mai mică cheie se mută în stânga, spre poziția ei finală, și alte chei mai mici se îndreaptă ușor în partea stânga, astfel încât partea nesortata a listei devine mai sortată pe măsură ce bubblesort continuă. [7]

2 Selection Sort

Selection sort funcționează destul de simplu. Începe prin a căuta în listă cea mai mică cheie, și o pune pe prima poziție a listei. Această cheie e acum primul element al listei sortate, aflată în partea stângă. Restul listei este încă nesortat și se caută acum în acel rest cea mai mică cheie, care este apoi pusă la locul ei. Acest proces continuă până când toată lista este sortată.[7]

3 Insertion Sort

Insertion Sort, după cum îi sugerează și numele, este folosit pentru a insera o singură cheie pe poziția sa corectă într-o listă sortată. Începând cu o listă sortată de dimensiune 1, cheia din dreaptă listei sortate este interschimbata în stânga listei cu câte o cheie, până când ajunge în poziția sa corectă. În acest mod, toată lista va ajunge sortată. [7]

4 Merge Sort

Îmbinarea este o tehnică care preia două liste sortate și le combină într-o singură listă sortată. Începând cu cea mai mică cheie de la fiecare dintre cele două liste, scrie cea mai mică cheie într-o a treia listă. Apoi compară următoarea cheie de pe prima listă cu următoarea cheie de pe a doua listă și scrie cea mai mică cheie dintre ele în lista auxiliară. Îmbinarea continuă până când ambele liste sunt combinate într-o singură listă sortată.

Merge Sort funcționează pe următorul principiu: tratează o lista de dimensiunea N ca N liste sortate de dimensiune 1. Apoi îmbină primele două liste într-o singură lista de dimensiune 2, și la fel pentru toate celalalte perechi. Listele sunt apoi îmbinate cu cele alăturate lor, formând liste de dimensiune 4, apoi 8, și așa mai departe până când listele sunt complet sortate. [7]

5 Quick Sort

Quicksort este un celebru algoritm de sortare, dezvoltat de Tony Hoare [6] Quicksort este un algoritm de tipul "divide and conquer". Își împarte continuu

listele în jurul unui pivot, interschimbând cheile mari din partea stângă cu cheile mici din partea dreaptă față de pivot. Partiționarea face ca pivotul să fie pus în poziția sa finală, cu toate cheile mai mari în dreapta, și cheile mai mici în stânga. Partițiile din stânga și din dreaptă sunt apoi sortate recursiv.

III Studiu de caz

1 Descrierea experimentului

Pentru a putea compara eficiența algoritmilor de sortare asupra unor liste random am pregătit următorul experiment:

Consideram:

1. O listă de 10.000 de elemente
2. O listă de 100.000 de elemente
3. 100 liste a câte 1.000 elemente
4. 25 de liste a câte 4.000 de elemente
5. 10 liste a câte 10.000 elemente

Pentru generarea listelor am folosit următorii algoritmi: 1

```
lista=[]
outer=[]

def lista_mica(lista):
    for i in range(10000):
        lista.append(int(random.random()*10000))

def lista_mare(lista):
    for i in range(100000):
        lista.append(int(random.random()*100000))

def out_list(lista,elemente,marime_lista):
    for i in range(marime_lista): # lista care contine mai multe
    liste
        lista.append([])
        for n in range(elemente):
            lista[i].append(int(random.random()*100000))
```

Figura 1: Generarea unor liste cu elemente random în Python

În continuare voi prezenta o scurtă descriere a modului de funcionare al algoritmilor pentru cei nefamiliarizați cu acești algoritmi. Algoritmii folosiți pentru implementare sunt preluați din Fast sorting on Modern Computers (P. Biggar[7])

Pentru fiecare dintre listele de mai sus, s-au efectuat 5 execuții ale programului, pentru a ne asigura că nu apare nici o diferență majoră între timpii de execuție. Fiecare rezultat în urma rulării programului a fost marcat pe grafic cu un pătrat.

2 Bubble Sort

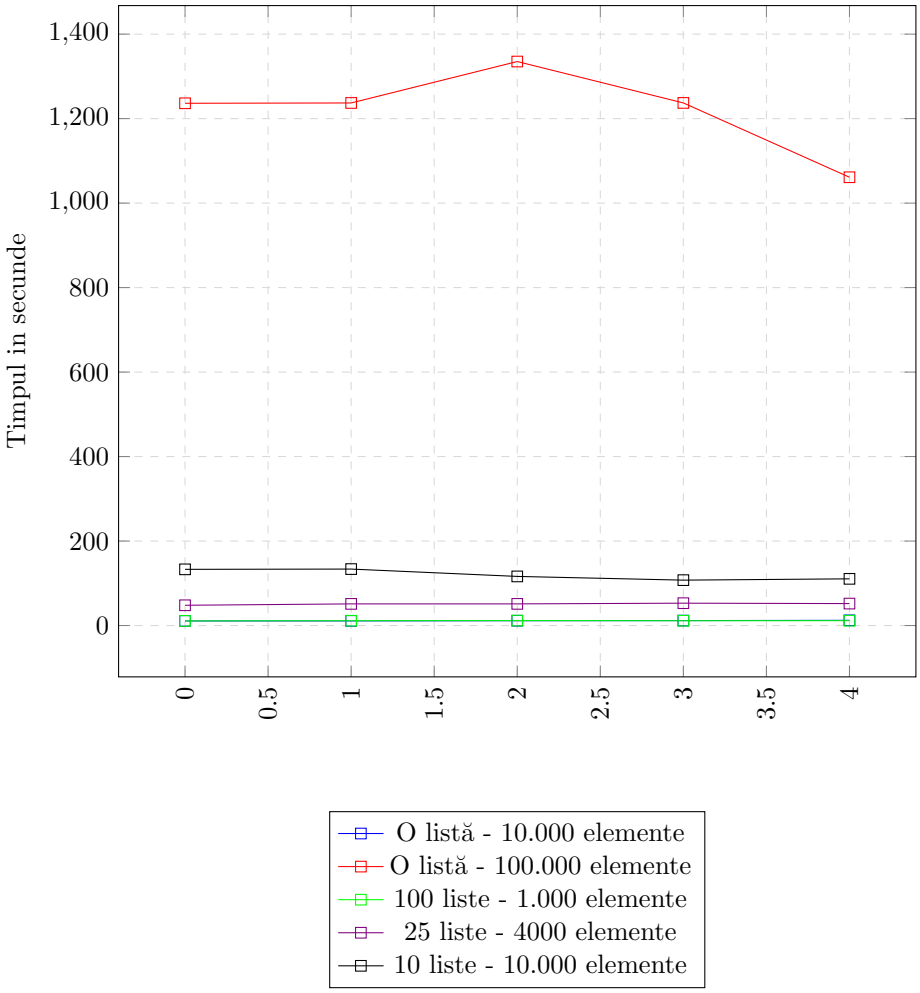

Figura 2: Reprezentare Grafică a timpurilor de execuție cu Bubble Sort

3 Selection Sort

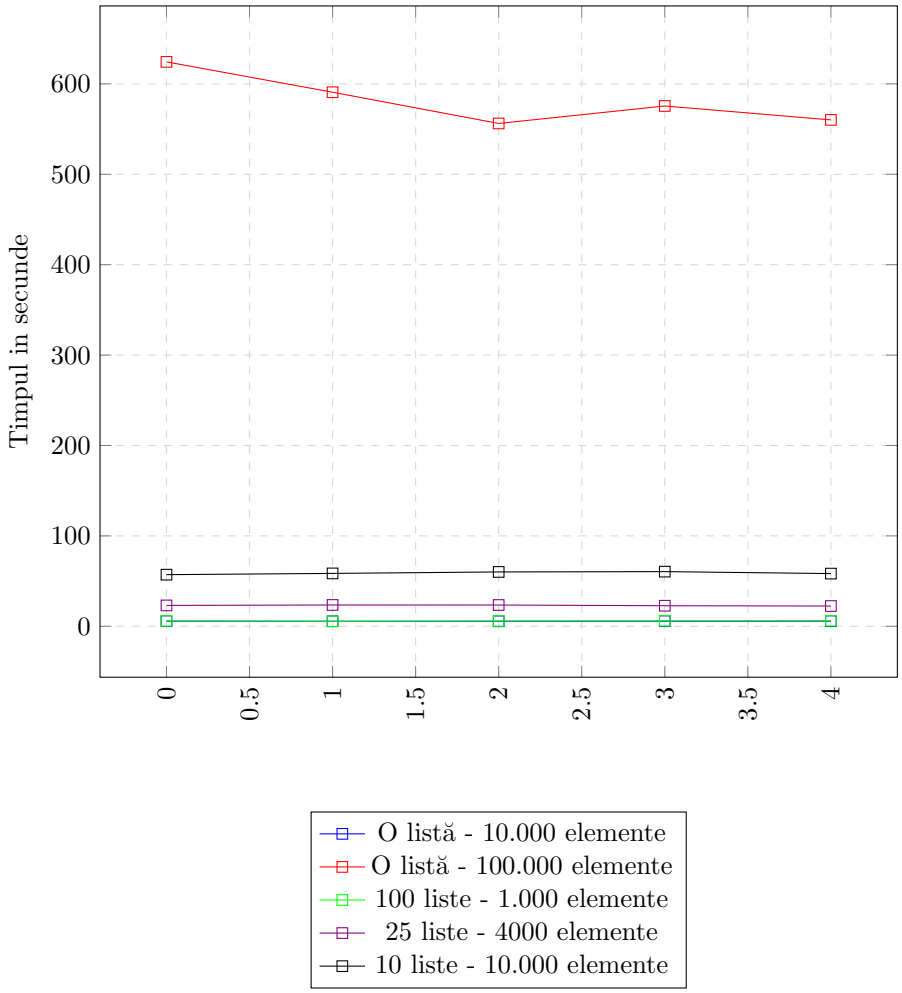

Figura 3: Reprezentare Grafică a timpurilor de execuție cu Selection Sort

4 Insertion Sort

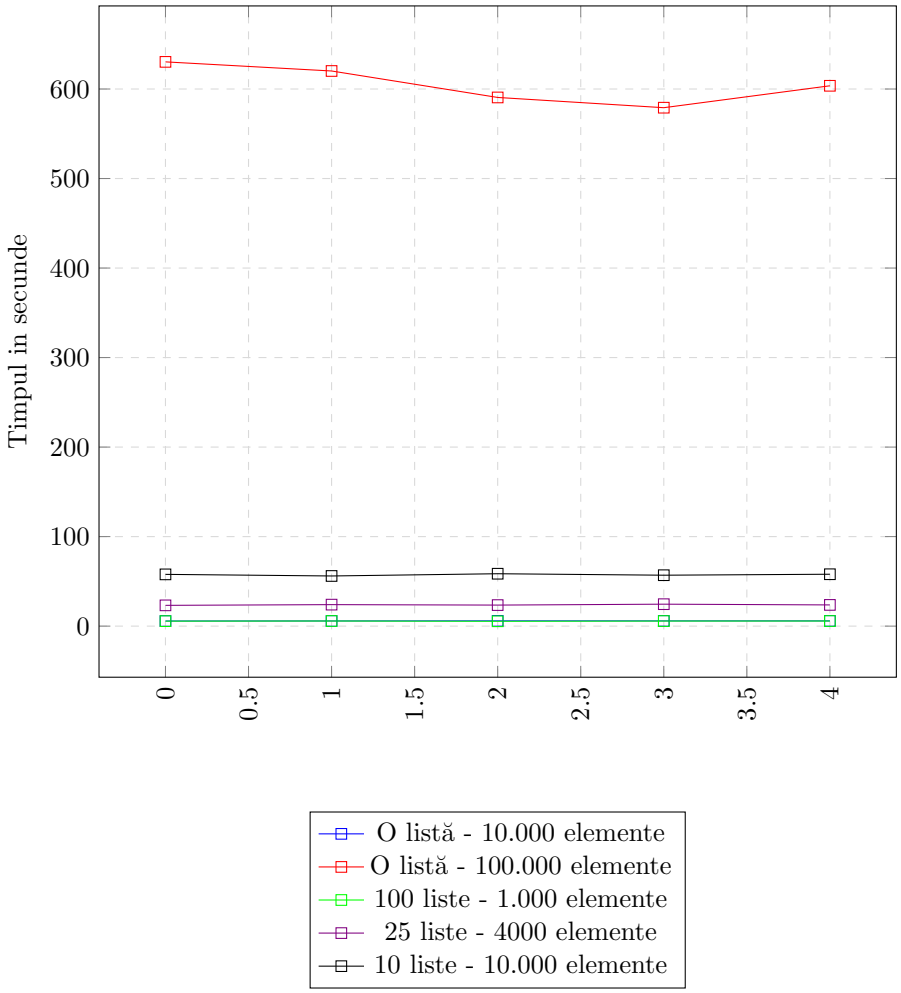

Figura 4: Reprezentare Grafică a timpurilor de execuție cu Insertion Sort

5 Merge Sort

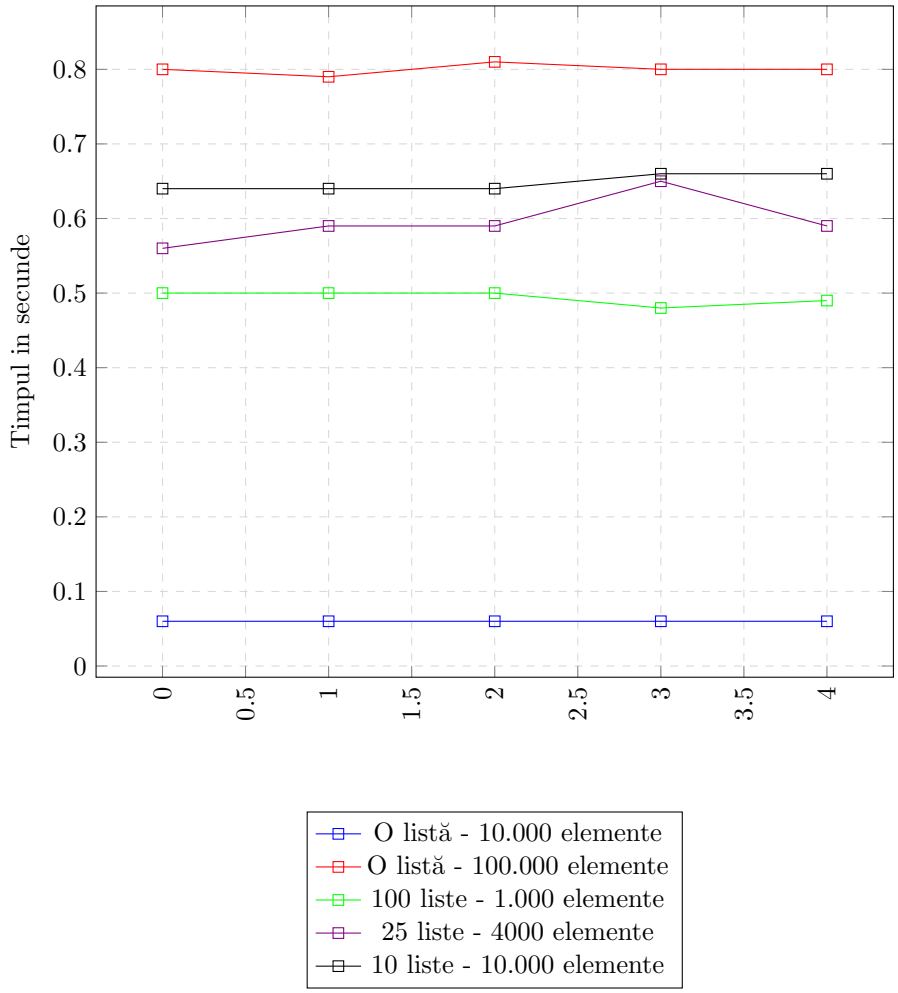

Figura 5: Reprezentare Grafică a timpurilor de execuție cu Merge Sort

6 Quick Sort

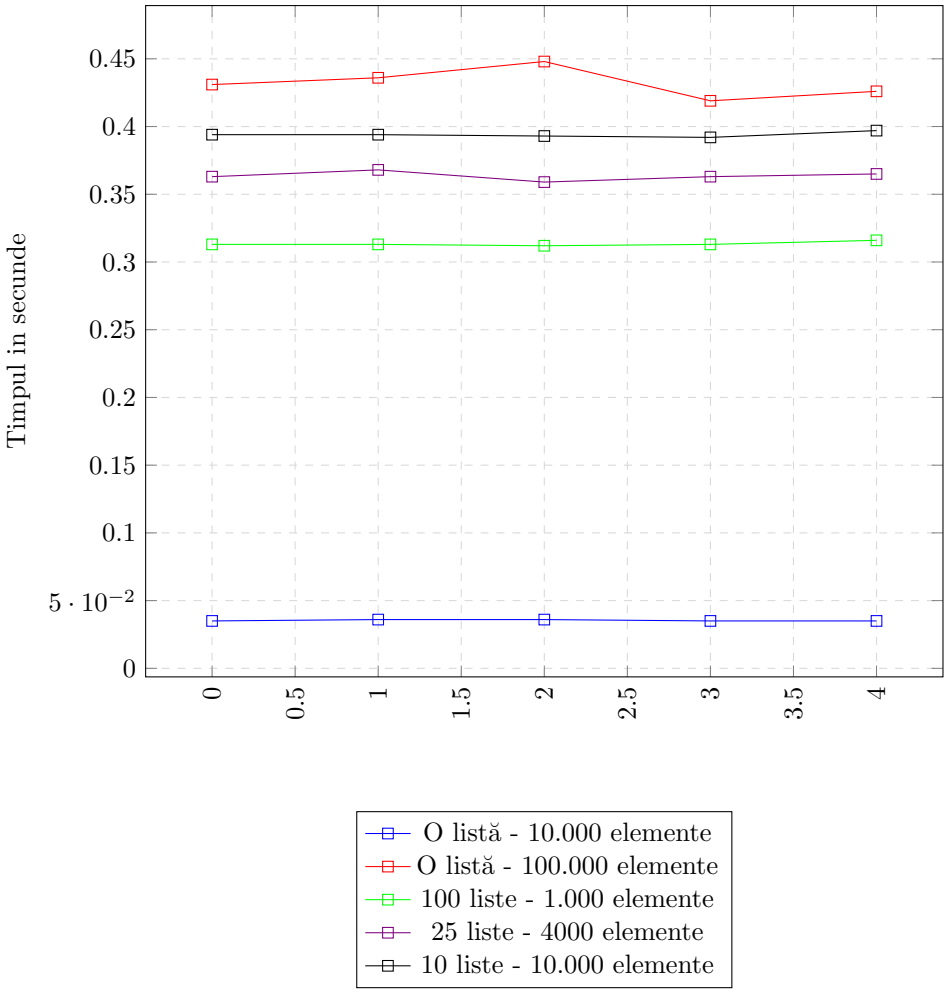

Figura 6: Reprezentare Grafică a timpurilor de execuție cu Quick Sort

7 Note Experiment

7.1 Așteptări inițiale

Înainte să desfășor experimentul, mi-am notat niște posibile rezultate și așteptări pe care le-am avut de la acești algoritmi. Am evidențiat cu culoarea roșie, algoritmii care m-au surprins, iar cu albastru, cei care s-au comportat într-un mod așteptat.

- Mă aștept ca Bubble Sort să fie cel mai încet algoritm dintre cei testați, însă este posibil să se descurce destul de bine pe lista de 10.000 sau pe listele cu foarte puține elemente.
- Selection Sort și Insertion Sort se vor comporta cel mai probabil similar lui Bubble Sort.
- Merge Sort va fi al doilea cel mai rapid algoritm după Quick Sort, care va rămâne în acest caz, cea mai bună opțiune

7.2 Opinii în urma experimentului

După efectuarea experimentului, majoritatea algoritmilor s-au comportat exact cum mă așteptam însă am fost surprins de diferența între Bubble Sort și Insertion/Selection Sort. Mă așteptam să se comporte similar, însă Bubble Sort a durat de 2x mai mult decât ceilalți doi algoritmi. O altă observație care merită notată este diferența dintre Bubble,Insertion și Selection și Merge,Quick Sort. Este mai mult decât remarcabilă.

IV Comparație cu literatura

În literatură, există diverse studii care evaluează performanța sau complexitatea algoritmilor de sortare. Pe lângă abundența de cărți și specialiști care au abordat acest subiect, există și multe articole relevante mai putin cunoscute. Aș vrea să menționez mai jos, câteva dintre articolele studiate de mine în vederea realizării acestei lucrări.

Principala sursă folosită pentru această lucrare este lucrarea lui Paul Biggar & David Gregg [7] care au realizat o muncă remarcabilă comparând comportamentul algoritmilor de sortare asupra branch predictors și caches. Din această lucrare am adaptat codul sursă pentru implementarea algoritmilor de sortare și descrierea sumară a fiecărui algoritm de sortare (Chapter 4 - Elementary Sorts). Este un articol extrem de detaliat care pe langa tipurile elementare de sortare, acopera probleme mult mai complexe.

Un alt articol util care studiază comparativ algoritmii de sortare este realizat de către Opeyemi Adesina [1] în vederea creării celui mai eficient algoritm de sortare. Aici se prezintă atât implementarea algoritmilor de sortare cât și o comparație a acestora asupra unor liste mari de date (10.000-200.000 elemente) și compararea cantității de memorie ocupate de fiecare sortare.

Nu în ultimul rând, unul dintre articolele cele mai bune găsite de mine este cel scris de Akhter N. [2] care compară în detaliu, cu ajutorul graficelor 3D, eficiența fiecărui algoritm în Best Case, Worst Case și Average Case.

V Concluzii si direcții viitoare

1 Concluzii generale

Pe parcursul acestei lucrări am analizat și comparat diferite metode de sortare asupra listelor. Timpurile mele de execuție ale algoritmilor pot să nu coincidă cu cele găsite online sau în alte lucrări datorită diferenței de hardware dar diferența de eficiență se observă destul de bine.

Chiar dacă nu se poate discuta despre un algoritm de sortare eficient universal, pentru mine și probabil mulți alți programatori în devenire, cele mai folosite metode de sortare sunt bubbleSort pentru o listă cu puține elemente, și Quick Sort pentru orice altceva, din două motive, implementare simplă și cod eficient.

Quicksort-ul este considerat unul dintre cei mai buni algoritmi de sortare și aceasta datorită capacității lui de a prelucra o listă foarte mare de termeni. Experimentul efectuat mai sus (Cap.III) nu face decât să confirme această teorie. Deoarece sortarea se realizează pe loc, nu avem nevoie de un spațiu de stocare auxiliar iar cu Bubble Sort o lista de 1000 de elemente (spre exemplu) care necesită o sortare rapidă, poate fi returnată în mult mai puțin de două secunde. Nu trebuie uitați nici algoritmii simpli de sortare, cum ar fi Sélection sort și Insertion sort, care sunt multi mai eficienți atunci când vorbim de o lista redusă de elemente.

Cu siguranță unul dintre cei mai "interesanți" algoritmi de sortare pe care i-am descoperit pregătind această lucrare este Bogo Sort, cunoscut și sub numele de "stupid sort", "slow sort", "shotgun sort" sau "monkey sort". Deși nu se poate vorbi de o metodă de sortare eficientă universal, Bogo sort este cu siguranță cea mai ineficientă metodă, universal, acest algoritm bazându-se pur și simplu pe amestecarea repetitivă a listei, până când aceasta va fi sortată. Nu există o complexitate anume pentru el deoarece se bazează doar pe noroc, o lista de 10 elemente poate fi sortată după prima permutare, sau după o infinitate de permutări. Nu-mi pot imagina o aplicație în care această metodă de sortare este eficientă, decât dacă scopul algoritmului este să dureze extrem de mult timp.

2 Direcții viitoare

Problema algoritmilor de sortare rămâne încă deschisă. În această lucrare am testat doar eficiența algoritmilor de bază. Există mulți alți algoritmi folosiți în programare pe care i-am omis în această lucrare însă cel mai probabil îi voi parcurge pe viitor într-o lucrare mult mai amănunțită și mai bine structurată, subdomeniul algoritmilor de sortare fiind unul destul de vast.

Bibliografie

[1] Opeyemi Adesina. **?**A Comparative Study of Sorting Algorithms**?** In: *African Journal of Computing & ICT* 6 (May 2013), pp. 199–206.

[2] Naeem Akhter, Muhammad Idrees, and Furqan-ur-Rehman. **?**Sorting Algorithms – A Comparative Study**?** In: *International Journal of Computer Science and Information Security,* 14 (May 2016), pp. 930–936.

[3] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

[4] Donald E. Knuth. *Arta Programării Calculatoarelor*. ed .2. 1968.

[5] Stefan Hougardy and Jens Vygen. **?**Sorting Algorithms**?** In: *Algorithmic Mathematics*. Springer, 2016, pp. 91–107.

[6] Ole-Johan Dahl, Edsger Wybe Dijkstra, and C A R Hoare. *Structured programming*. 1990. ISBN: 0-12-200550-3.

[7] Paul Biggar and David Gregg. *Sorting in the Presence of Branch Prediction and Caches - Fast Sorting on Modern Computers*. Tech. rep. Dublin: University of Dublin, 2005. URL: `https://www.scss.tcd.ie/publications/tech-reports/reports.05/TCD-CS-2005-57.pdf`.

[8] Steven S. Skiena. *The Algorithm Design Manual*. Second Edition. Springer, 2008, p. 739. ISBN: 978-1-84800-069-8. DOI: `10.1007/978-1-84800-070-4`.

VI Anexe

1 Tabel complet - liste sortate/random/sortate invers

Sorted array:

	100	1,000	10,000	100,000	1,000,000
Bubble Sort	0.040 ms	0.017 ms	0.057 ms	0.022 ms	0.012 ms
Insertion Sort	0.003 ms	0.004 ms	0.005 ms	0.004 ms	0.024 ms
Merge Sort	0.012 ms	0.23 ms	2.16 ms	20.35 ms	230.38 ms
Heap Sort	0.020 ms	0.33 ms	3.82 ms	42.83 ms	473.36 ms
Quick Sort	0.12 ms	3.73 ms	295.01 ms	29937.61 ms	-

Reverse sorted array:

	100	1,000	10,000	100,000	1,000,000
Bubble Sort	0.092 ms	5.95 ms	445.46 ms	44793.96 ms	-
Insertion Sort	0.049 ms	3.17 ms	259.75 ms	25116.82 ms	-
Merge Sort	0.017 ms	0.23 ms	2.34 ms	21.68 ms	234.51 ms
Heap Sort	0.026 ms	0.37 ms	4.17 ms	41.78 ms	500.63 ms
Quick Sort	0.060 ms	5.94 ms	438.57 ms	44018.92 ms	-

Randomly sorted array:

	100	1,000	10,000	100,000	1,000,000
Bubble Sort	0.050 ms	5.93 ms	445.92 ms	44677.46 ms	-
Insertion Sort	0.015 ms	1.72 ms	126.41 ms	12478.55 ms	-
Merge Sort	0.016 ms	0.22 ms	2.44 ms	29.38 ms	340.39 ms
Heap Sort	0.021 ms	0.28 ms	3.33 ms	43.16 ms	532.56 ms
Quick Sort	0.011 ms	0.16 ms	1.67 ms	20.01 ms	236.51 ms

2 Portal spre proiect in Latex

`https://www.overleaf.com/read/xhmhxbnzqypn`