Sortarea, analiza timpului și eficienței

Galben Casian-Petrică
Departamentul de Informatică
Facultatea de Matematică şi Informatică
Universitatea de Vest Timişoara, România
Email: casian.galben00@e-uvt.ro

May 2020

Rezumat

În acestă lucrare am realizat o comparație între metodele de sortare. Implementând 5 metode de sortare în Python, folosind platforma Anaconda, am testat pentru fiecare metodă de sortare în cât timp se vor sorta listele create în Python. Listele au fost create folosind funcția "Random". Folosind "Time" importat din pachetul predefinit de Python 3, am reușit să calculez timpul de execuție al algormului.

Încercând să găsesc pe internet fișiere cu milioane de elemente, am decis să generez eu propriile fișiere text în care am pus listele respective. Generând astfel prin funcția "lista", listele nesortate, având acum mai multe fișiere cu liste de diferite lungimi. În prezentare este descris modul în care algoritmii sortează listele și cât timp durează ca listele să fie sortate.

Pe tot parcursul lucrării o să analizăm modul în care aceste sortări se manifestă pentru liste de lungimi mari.

Cuprins

1 Introducere

1.1 Motivaţie

Această lucrare își propune să răspundă la simpla întrebare: "Cum putem realiza operația de sortare cât mai repede?". Ținând cont de faptul că în viața de zi cu zi ne confruntăm cu numeroase situații în care intervine necesitatea de a sorta informații, uneori chiar și fără să conștientizăm acest lucru, problema sortării se impune a fi analizată antent și optimizată prin metode cât mai facile, pe care le vom pune în evidență în următoarele capitole.

1.2 Descriere informală a soluţiei

O să putem observa pe parcursul lucrării, cum putem sorta rapid, eficient și care este cea mai bună metodă de sortare pentru a liste de lungimi mari.

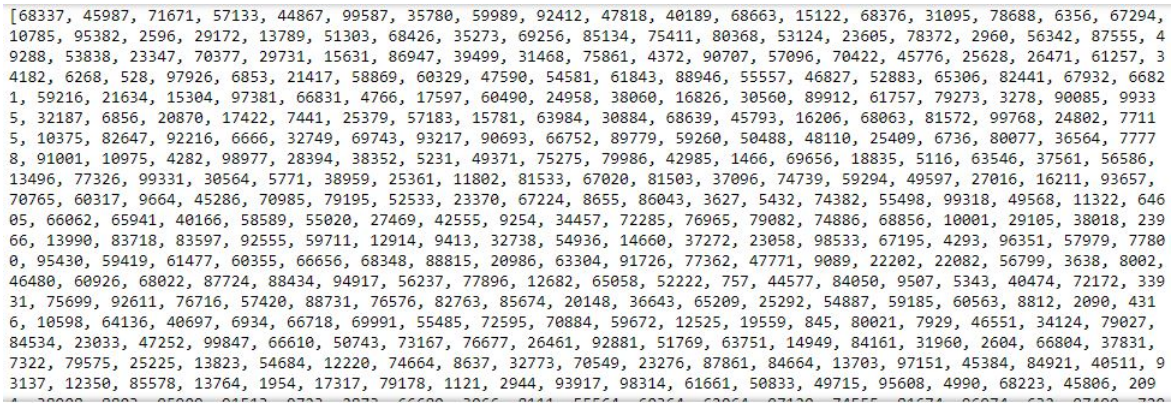
1.3 Exemple simple ce ilustrează problema şi soluţia

Un bun exemplu pe care îl voi expune este cel al sortării studentilor/elevilor după notele obținute la examene/teste. Când deja sunt multe persoane lucrurile se complică și de aceea avem nevoie de o metodă de sortare.

1.4 Exemplu complex

Unul dintre exemplele expuse în lucrare este cel al sortării listei de 100000 de elemente. O lista generată random. În cazurile în care avem de lucru cu liste de lungimi mici, metodele de sortare se comportă relativ asemănator. Când lucram cu liste de lungimi mari ne punem problema "Ce metodă de sortare este mai rapidă?"

O ilustrație a cum poate arăta o listă de 100000 elemente, o putem observa pe următaorea pagină.



1.5 Declaraţie de originalitate

În această lucrare, am încercat să expun problema sortării listelor de lungimi mari, astfel am implementat în Python3 5 metode de sortare deja cunoscute de lume(Metoda Inserției, Metoda Selecției, Metoda Sortarii Rapide, Metoda Bulelor, HeapSort) și am testat pe liste (generate de mine) de lungimi diferite, timpul de execuție al algoritmului.

2 Descrierea formală a problemei şi soluţiei

2.1 Introducere

În lume au fost create multe metode de sortare, dar noi o să discutăm doar despre 5 dintre ele:

- Metoda Inserției (Insertion Sort);
- Metoda Selecției (Selection Sort);
- Metoda Sortării Rapide (Quick Sort);
- Metoda Bulelor (Bubble Sort);
- HeapSort;

Acestea vor fi prezentate, pe rând, iar apoi vom observa care dintre aceste metode de sortare are timpul de execuție mai bun.

2.2 Metoda Insertiei (Insertion Sort)

Consideram șirul: x[1], x[2], · · · , x[n]. Se parcurge secvențial șirul din element în element. Se inserează elementul curent (x[i]) în subșirul pe care îl avem, precedentul acestuia, (x[1], x[2], · · · , x[i-1]) astfel încât acesta să rămână ordonat: x[1], x[2], · · · , x[i-1], x[i], x[i+1], · · · , x[n]. Subșirul ce conține elementele deja sortate crește la fiecare pas, astfel încât, după ce parcurgem toate elementele din șir, secvența este sortată în întregime. Algoritmul de sortare prin inserție este stabil si are ordinul de complexitate $O(n^2)$.

2.3 Metoda Selecției (Selection Sort)

Consideram șirul: x[1], x[2], · · · , x[n]. Se parcurge pe părți șirul (din element în element). Se determină elementul minim din subșirul (x[i], x[i+1], · · · , x[n]), și se înlocuiește cu elementul curent (x[i]), [x[1], x[2], · · · , x[i-1], x[i], x[i+1], · · · , x[n]]. Subșirul ce conține elementele deja sortate se mărește la fiecare pas, astfel încât, după ce parcurgem toate elementele, secvența este sortată în întregime. Are o complexitate a timpului de $O(n^2)$.

Acest lucru face sortarea prin selecție ineficientă pe listele mari și, în general, are un efect mai slab decât tipul de execuție al sortării prin inserție.

2.4 Metoda Sortării Rapide (Quick Sort)

Considerăm șirul: x[1], x[2], · · · , x[n]. Se alege din șir un element, numit pivot. [x[1], x[2], · · · , x[i-1], x[i],x[i+1], · · · , x[n]] Se realizează împărțirea șirului (în raport cu acest pivot): se reordonează șirul astfel încât elementele cu valori mai mici decât pivotul se plasează înainte de pivot, iar elementele cu valori mai mari decât pivotul se plasează după acesta, iar cele egale în oricare din părți).

După împărțire, pivotul se află în poziția finală. Se aplică recursiv aceeași procedură subșirului de elemente cu valori mai mici, iar separat subșirului cu valori mai mari. Quick Sort are ca ordin de complexitate $\theta(n \log n)$.

2.5 Metoda Bulelor (Bubble Sort)

Tabloul este parcurs de la stânga spre dreapta și elementele adiacente sunt comparate. Dacă nu sunt in ordinea dorită atunci se interschimbă. Procesul

este repetat până când tabloul ajunge să fie ordonat.

$$[x[1], x[2], \cdots, x[m-1], x[m], x[m+1], \cdots, x[n]]$$

Această variantă de implementare este cea mai puțin eficientă dintre acestea. Variantele mai bune evită execuția de (n-1) ori a ciclului exterior, oprind prelucrarea când tabloul este sortat deja. Bubble Sort este un algoritm stabil, chiar dacă nu este eficient.

2.6 HeapSort

Algoritmul de sortare "Heap Sort" a fost creat în dorinţa de a îmbunătăţi algoritmul sortării pin selecţie (elementul minim sau maxim din vector se va aşeza in locul primului sau ultimului element al vectorului, algoritmul reluându-se pentru cele n-1 elemente rămase). Fiind dat un vector x cu n componente se cere ca acesta să fie sortat crescător sau descrescător prin metoda Heap-urilor.

În cazul în care vectorul se organizează ca un MinHeap, prima componentă reţine cea mai mică valoare, apoi se schimbă conţinuturile componentelor 1 şi n, deoarece în acest caz, ultima componentă reţine valoarea cea mai mare. Din prima componentă şi MinHeap-urile cu vârfurile x[2] şi x[3] se formează un nou MinHeap. Se interschimbă conţinuturile componentelor 1 şi n-1.

3 Modelarea şi implementarea problemei şi soluţiei

3.1 Manualul de sistem

În această parte o să discutăm despre cum am implementat agoritmii deja cunoscuți de lume. Implementarea algoritmilor de sortare a fost facută în Python 3 ca și limbaj de programare, folosind Anaconda ca și cale către locul unde puteam folosii limbajul.

- **Metoda Inserției**

```
def insertie(x):
    start=time()
```

```
for i in range(1,len(x)):
        aux=x[i]
        j=i-1
        while j>=0 and aux<x[j]:
                x[j+1]=x[j]
                j=j-1
        x[j+1]=aux
print(x)
print("Codul a durat {}".format(time()-start))
```

- **Metoda Selecției**

```
def selectie(x):
    start=time()
    for i in range(len(x)):
            k=i
            for j in range(i+1, len(x)):
                    if x[k]>x[j]:
                    k=j
            if k!=i:
                    x[i],x[k]=x[k],x[i]
    print(x)
    print("Codul a durat {}".format(time()-start))
```

- **Metoda Sortării Rapide**

```
def partitie(x, s, d):
    pivot = x[s]
    low = s + 1
    high = d
    while True:
            while low <= high and x[high] >= pivot:
                    high = high - 1
            while low <= high and x[low] <= pivot:
                    low = low + 1
            if low <= high:
                    x[low], x[high] = x[high], x[low]
            else:
                    break
```

```
    x[s], x[high] = x[high], x[s]
    return high

def quick_sort(x, s, d):
    if s >= d:
    return

    p = partitie(x, s, d)
    quick_sort(x, s, p-1)
    quick_sort(x, p+1, d)

start=time()
quick_sort(n5, 0, len(n5) - 1)
print("Codul a durat {}".format(time()-start))
```

- **Metoda Bulelor**

```
def bubbleSort(x):
    start=time()
    n = len(x)
    for i in range(n):
            for j in range(0, n-i-1):
                    if x[j] > x[j+1] :
                            x[j], x[j+1] = x[j+1], x[j]
    print(x)
    print("Codul a durat {}".format(time()-start))
```

- **HeapSort**

```
def heap(x, n, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2
            if l < n and x[i] < x[l]:
            largest = l
    if r < n and x[largest] < x[r]:
            largest = r
    if largest != i:
            x[i],x[largest] = x[largest],x[i]
```

```
            heap(x, n, largest)

def heapSort(x):
    n = len(x)
    for i in range(n, -1, -1):
            heap(x, n, i)
    for i in range(n-1, 0, -1):
            x[i], x[0] = x[0], x[i]
            heap(x,i,0)
    start=time()
    print(n)
    print("Codul a durat {}".format(time()-start))
```

3.2 Manual de utilizare

Implementând 5 metode de sortare în Python, folosind platforma Anaconda, am testat pentru fiecare metodă de sortare în cât timp se vor sorta listele create în Python. Listele au fost create folosind funția "Random". Folosind "Time" importat din pachetul predefinit de Python 3, am reușit să calculez timpul de execuție al algormului.

Pașii pentru a putea accesa locul unde putem implementa algoritmii sunt:

- Descărcare Anaconda de pe Google Chrome;
- Creare cont și conectare pe platformă;
- Accesare Python 3;
- Implemantarea algoritmilor descriși;
- Testarea algoritmilor asupra unor exemple simple (liste lungimi mici);
- Testarea algoritmilor asupra unor exemple mai complicate (liste lungimi mari);

4 Studiu de caz / Experiment

4.1 Metoda Inserției

- **Metoda Inserției asupra unei liste de lungime 10**

```
def insertie(x):
    start=time()
    for i in range(1,len(x)):
        aux=x[i]
        j=i-1
        while j>=0 and aux<x[j]:
            x[j+1]=x[j]
            j=j-1
        x[j+1]=aux
    print(x)
    print("Codul a durat {}".format(time()-start))
insertie(n6)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
Codul a durat 0.0010001659393310547
```

- **Metoda Inserției asupra unei liste de lungime 100000**

```
def insertie(x):
    start=time()
    for i in range(1,len(x)):
        aux=x[i]
        j=i-1
        while j>=0 and aux<x[j]:
            x[j+1]=x[j]
            j=j-1
        x[j+1]=aux
    print(x)
    print("Codul a durat {}".format(time()-start))
insertie(n1)
```

```
99705, 99706, 99707, 99708, 99709, 99710, 99711, 99712, 99713, 99714, 99715, 99716, 99717, 99718, 99719, 99720, 99721, 99722,
99723, 99724, 99725, 99726, 99727, 99728, 99729, 99730, 99731, 99732, 99733, 99734, 99735, 99736, 99737, 99738, 99739, 99740,
99741, 99742, 99743, 99744, 99745, 99746, 99747, 99748, 99749, 99750, 99751, 99752, 99753, 99754, 99755, 99756, 99757, 99758,
99759, 99760, 99761, 99762, 99763, 99764, 99765, 99766, 99767, 99768, 99769, 99770, 99771, 99772, 99773, 99774, 99775, 99776,
99777, 99778, 99779, 99780, 99781, 99782, 99783, 99784, 99785, 99786, 99787, 99788, 99789, 99790, 99791, 99792, 99793, 99794,
99795, 99796, 99797, 99798, 99799, 99800, 99801, 99802, 99803, 99804, 99805, 99806, 99807, 99808, 99809, 99810, 99811, 99812,
99813, 99814, 99815, 99816, 99817, 99818, 99819, 99820, 99821, 99822, 99823, 99824, 99825, 99826, 99827, 99828, 99829, 99830,
99831, 99832, 99833, 99834, 99835, 99836, 99837, 99838, 99839, 99840, 99841, 99842, 99843, 99844, 99845, 99846, 99847, 99848,
99849, 99850, 99851, 99852, 99853, 99854, 99855, 99856, 99857, 99858, 99859, 99860, 99861, 99862, 99863, 99864, 99865, 99866,
99867, 99868, 99869, 99870, 99871, 99872, 99873, 99874, 99875, 99876, 99877, 99878, 99879, 99880, 99881, 99882, 99883, 99884,
99885, 99886, 99887, 99888, 99889, 99890, 99891, 99892, 99893, 99894, 99895, 99896, 99897, 99898, 99899, 99900, 99901, 99902,
99903, 99904, 99905, 99906, 99907, 99908, 99909, 99910, 99911, 99912, 99913, 99914, 99915, 99916, 99917, 99918, 99919, 99920,
99921, 99922, 99923, 99924, 99925, 99926, 99927, 99928, 99929, 99930, 99931, 99932, 99933, 99934, 99935, 99936, 99937, 99938,
99939, 99940, 99941, 99942, 99943, 99944, 99945, 99946, 99947, 99948, 99949, 99950, 99951, 99952, 99953, 99954, 99955, 99956,
99957, 99958, 99959, 99960, 99961, 99962, 99963, 99964, 99965, 99966, 99967, 99968, 99969, 99970, 99971, 99972, 99973, 99974,
99975, 99976, 99977, 99978, 99979, 99980, 99981, 99982, 99983, 99984, 99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992,
99993, 99994, 99995, 99996, 99997, 99998, 99999]
Codul a durat 1066.1489803791046
```

4.2 Metoda Selecției

- **Metoda Selecției asupra unei liste de lungime 10**

```
def selectie(x):
    start=time()
    for i in range(len(x)):
        k=i
        for j in range(i+1, len(x)):
            if x[k]>x[j]:
                k=j
        if k!=i:
            x[i],x[k]=x[k],x[i]
    print(x)
    print("Codul a durat {}".format(time()-start))

selectie(n6)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
Codul a durat 0.0
```

- **Metoda Selecției asupra unei liste de lungime 100000**

```
def selectie(x):
    start=time()
    for i in range(len(x)):
        k=i
        for j in range(i+1, len(x)):
            if x[k]>x[j]:
                k=j
        if k!=i:
            x[i],x[k]=x[k],x[i]
    print(x)
    print("Codul a durat {}".format(time()-start))

selectie(n1)
```

```
99705, 99706, 99707, 99708, 99709, 99710, 99711, 99712, 99713, 99714, 99715, 99716, 99717, 99718, 99719, 99720, 99721, 99722,
99723, 99724, 99725, 99726, 99727, 99728, 99729, 99730, 99731, 99732, 99733, 99734, 99735, 99736, 99737, 99738, 99739, 99740,
99741, 99742, 99743, 99744, 99745, 99746, 99747, 99748, 99749, 99750, 99751, 99752, 99753, 99754, 99755, 99756, 99757, 99758,
99759, 99760, 99761, 99762, 99763, 99764, 99765, 99766, 99767, 99768, 99769, 99770, 99771, 99772, 99773, 99774, 99775, 99776,
99777, 99778, 99779, 99780, 99781, 99782, 99783, 99784, 99785, 99786, 99787, 99788, 99789, 99790, 99791, 99792, 99793, 99794,
99795, 99796, 99797, 99798, 99799, 99800, 99801, 99802, 99803, 99804, 99805, 99806, 99807, 99808, 99809, 99810, 99811, 99812,
99813, 99814, 99815, 99816, 99817, 99818, 99819, 99820, 99821, 99822, 99823, 99824, 99825, 99826, 99827, 99828, 99829, 99830,
99831, 99832, 99833, 99834, 99835, 99836, 99837, 99838, 99839, 99840, 99841, 99842, 99843, 99844, 99845, 99846, 99847, 99848,
99849, 99850, 99851, 99852, 99853, 99854, 99855, 99856, 99857, 99858, 99859, 99860, 99861, 99862, 99863, 99864, 99865, 99866,
99867, 99868, 99869, 99870, 99871, 99872, 99873, 99874, 99875, 99876, 99877, 99878, 99879, 99880, 99881, 99882, 99883, 99884,
99885, 99886, 99887, 99888, 99889, 99890, 99891, 99892, 99893, 99894, 99895, 99896, 99897, 99898, 99899, 99900, 99901, 99902,
99903, 99904, 99905, 99906, 99907, 99908, 99909, 99910, 99911, 99912, 99913, 99914, 99915, 99916, 99917, 99918, 99919, 99920,
99921, 99922, 99923, 99924, 99925, 99926, 99927, 99928, 99929, 99930, 99931, 99932, 99933, 99934, 99935, 99936, 99937, 99938,
99939, 99940, 99941, 99942, 99943, 99944, 99945, 99946, 99947, 99948, 99949, 99950, 99951, 99952, 99953, 99954, 99955, 99956,
99957, 99958, 99959, 99960, 99961, 99962, 99963, 99964, 99965, 99966, 99967, 99968, 99969, 99970, 99971, 99972, 99973, 99974,
99975, 99976, 99977, 99978, 99979, 99980, 99981, 99982, 99983, 99984, 99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992,
99993, 99994, 99995, 99996, 99997, 99998, 99999]
Codul a durat 938.7636942863464
```

4.3 Metoda Sortării Rapide

- **Metoda Sortării Rapide asupra unei liste de lungime 10**

```
def quick_sort(x, s, d):
    if s >= d:
        return

    p = partitie(x, s, d)
    quick_sort(x, s, p-1)
    quick_sort(x, p+1, d)

start=time()
quick_sort(n6, 0, len(n6) - 1)
print("Codul a durat {}".format(time()-start))
print(n6)
```

```
Codul a durat 0.0
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- **Metoda Sortării Rapide asupra unei liste de lungime 100000**

```
    quick_sort(x, p+1, d)

start=time()
quick_sort(n1, 0, len(n1) - 1)
print(n1)
print("Codul a durat {}".format(time()-start))
```

```
99687, 99688, 99689, 99690, 99691, 99692, 99693, 99694, 99695, 99696, 99697, 99698, 99699, 99700, 99701, 99702, 99703, 99704,
99705, 99706, 99707, 99708, 99709, 99710, 99711, 99712, 99713, 99714, 99715, 99716, 99717, 99718, 99719, 99720, 99721, 99722,
99723, 99724, 99725, 99726, 99727, 99728, 99729, 99730, 99731, 99732, 99733, 99734, 99735, 99736, 99737, 99738, 99739, 99740,
99741, 99742, 99743, 99744, 99745, 99746, 99747, 99748, 99749, 99750, 99751, 99752, 99753, 99754, 99755, 99756, 99757, 99758,
99759, 99760, 99761, 99762, 99763, 99764, 99765, 99766, 99767, 99768, 99769, 99770, 99771, 99772, 99773, 99774, 99775, 99776,
99777, 99778, 99779, 99780, 99781, 99782, 99783, 99784, 99785, 99786, 99787, 99788, 99789, 99790, 99791, 99792, 99793, 99794,
99795, 99796, 99797, 99798, 99799, 99800, 99801, 99802, 99803, 99804, 99805, 99806, 99807, 99808, 99809, 99810, 99811, 99812,
99813, 99814, 99815, 99816, 99817, 99818, 99819, 99820, 99821, 99822, 99823, 99824, 99825, 99826, 99827, 99828, 99829, 99830,
99831, 99832, 99833, 99834, 99835, 99836, 99837, 99838, 99839, 99840, 99841, 99842, 99843, 99844, 99845, 99846, 99847, 99848,
99849, 99850, 99851, 99852, 99853, 99854, 99855, 99856, 99857, 99858, 99859, 99860, 99861, 99862, 99863, 99864, 99865, 99866,
99867, 99868, 99869, 99870, 99871, 99872, 99873, 99874, 99875, 99876, 99877, 99878, 99879, 99880, 99881, 99882, 99883, 99884,
99885, 99886, 99887, 99888, 99889, 99890, 99891, 99892, 99893, 99894, 99895, 99896, 99897, 99898, 99899, 99900, 99901, 99902,
99903, 99904, 99905, 99906, 99907, 99908, 99909, 99910, 99911, 99912, 99913, 99914, 99915, 99916, 99917, 99918, 99919, 99920,
99921, 99922, 99923, 99924, 99925, 99926, 99927, 99928, 99929, 99930, 99931, 99932, 99933, 99934, 99935, 99936, 99937, 99938,
99939, 99940, 99941, 99942, 99943, 99944, 99945, 99946, 99947, 99948, 99949, 99950, 99951, 99952, 99953, 99954, 99955, 99956,
99957, 99958, 99959, 99960, 99961, 99962, 99963, 99964, 99965, 99966, 99967, 99968, 99969, 99970, 99971, 99972, 99973, 99974,
99975, 99976, 99977, 99978, 99979, 99980, 99981, 99982, 99983, 99984, 99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992,
99993, 99994, 99995, 99996, 99997, 99998, 99999]
Codul a durat 1.0884652137756348
```

4.4 Metoda Bulelor

- **Metoda Bulelor asupra unei liste de lungime 50**

```
def bubbleSort(x):
    start=time()
    n = len(x)
    for i in range(n):
        for j in range(0, n-i-1):
             if x[j] > x[j+1] :
                x[j], x[j+1] = x[j+1], x[j]
    print(x)
    print("Codul a durat {}".format(time()-start))
bubbleSort(n5)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25,
35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]
Codul a durat 0.0010418891906738281
```

- **Metoda Bulelor asupra unei liste de lungime 100000**

```
def bubbleSort(x):
    start=time()
    n = len(x)
    for i in range(n):
        for j in range(0, n-i-1):
             if x[j] > x[j+1] :
                x[j], x[j+1] = x[j+1], x[j]
    print(x)
    print("Codul a durat {}".format(time()-start))
bubbleSort(n1)
```

```
99687, 99688, 99689, 99690, 99691, 99692, 99693, 99694, 99695, 99696, 99697, 99698, 99699, 99700, 99701, 99702, 99703, 99704,
99705, 99706, 99707, 99708, 99709, 99710, 99711, 99712, 99713, 99714, 99715, 99716, 99717, 99718, 99719, 99720, 99721, 99722,
99723, 99724, 99725, 99726, 99727, 99728, 99729, 99730, 99731, 99732, 99733, 99734, 99735, 99736, 99737, 99738, 99739, 99740,
99741, 99742, 99743, 99744, 99745, 99746, 99747, 99748, 99749, 99750, 99751, 99752, 99753, 99754, 99755, 99756, 99757, 99758,
99759, 99760, 99761, 99762, 99763, 99764, 99765, 99766, 99767, 99768, 99769, 99770, 99771, 99772, 99773, 99774, 99775, 99776,
99777, 99778, 99779, 99780, 99781, 99782, 99783, 99784, 99785, 99786, 99787, 99788, 99789, 99790, 99791, 99792, 99793, 99794,
99795, 99796, 99797, 99798, 99799, 99800, 99801, 99802, 99803, 99804, 99805, 99806, 99807, 99808, 99809, 99810, 99811, 99812,
99813, 99814, 99815, 99816, 99817, 99818, 99819, 99820, 99821, 99822, 99823, 99824, 99825, 99826, 99827, 99828, 99829, 99830,
99831, 99832, 99833, 99834, 99835, 99836, 99837, 99838, 99839, 99840, 99841, 99842, 99843, 99844, 99845, 99846, 99847, 99848,
99849, 99850, 99851, 99852, 99853, 99854, 99855, 99856, 99857, 99858, 99859, 99860, 99861, 99862, 99863, 99864, 99865, 99866,
99867, 99868, 99869, 99870, 99871, 99872, 99873, 99874, 99875, 99876, 99877, 99878, 99879, 99880, 99881, 99882, 99883, 99884,
99885, 99886, 99887, 99888, 99889, 99890, 99891, 99892, 99893, 99894, 99895, 99896, 99897, 99898, 99899, 99900, 99901, 99902,
99903, 99904, 99905, 99906, 99907, 99908, 99909, 99910, 99911, 99912, 99913, 99914, 99915, 99916, 99917, 99918, 99919, 99920,
99921, 99922, 99923, 99924, 99925, 99926, 99927, 99928, 99929, 99930, 99931, 99932, 99933, 99934, 99935, 99936, 99937, 99938,
99939, 99940, 99941, 99942, 99943, 99944, 99945, 99946, 99947, 99948, 99949, 99950, 99951, 99952, 99953, 99954, 99955, 99956,
99957, 99958, 99959, 99960, 99961, 99962, 99963, 99964, 99965, 99966, 99967, 99968, 99969, 99970, 99971, 99972, 99973, 99974,
99975, 99976, 99977, 99978, 99979, 99980, 99981, 99982, 99983, 99984, 99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992,
99993, 99994, 99995, 99996, 99997, 99998, 99999]
Codul a durat 2438.455954313278
```

4.5 HeapSort

- **HeapSort asupra unei liste de lungime 10**

```
def heap(x, n, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2
    if l < n and x[i] < x[l]:
        largest = l
    if r < n and x[largest] < x[r]:
        largest = r
    if largest != i:
        x[i],x[largest] = x[largest],x[i]
        heap(x, n, largest)

def heapSort(x):
    n = len(x)
    for i in range(n, -1, -1):
        heap(x, n, i)
    for i in range(n-1, 0, -1):
        x[i], x[0] = x[0], x[i]
        heap(x,i,0)
start=time()
heapSort(n6)
print(n6)
print("Codul a durat {}".format(time()-start))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
Codul a durat 0.0
```

- **HeapSortr asupra unei liste de lungime 100000**

```
def heap(x, n, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2
    if l < n and x[i] < x[l]:
        largest = l
    if r < n and x[largest] < x[r]:
        largest = r
    if largest != i:
        x[i],x[largest] = x[largest],x[i]
        heap(x, n, largest)

def heapSort(x):
    n = len(x)
    for i in range(n, -1, -1):
        heap(x, n, i)
    for i in range(n-1, 0, -1):
        x[i], x[0] = x[0], x[i]
        heap(x,i,0)
start=time()
heapSort(n1)
print(n1)
print("Codul a durat {}".format(time()-start))
```

```
99687, 99688, 99689, 99690, 99691, 99692, 99693, 99694, 99695, 99696, 99697, 99698, 99699, 99700, 99701, 99702, 99703, 99704,
99705, 99706, 99707, 99708, 99709, 99710, 99711, 99712, 99713, 99714, 99715, 99716, 99717, 99718, 99719, 99720, 99721, 99722,
99723, 99724, 99725, 99726, 99727, 99728, 99729, 99730, 99731, 99732, 99733, 99734, 99735, 99736, 99737, 99738, 99739, 99740,
99741, 99742, 99743, 99744, 99745, 99746, 99747, 99748, 99749, 99750, 99751, 99752, 99753, 99754, 99755, 99756, 99757, 99758,
99759, 99760, 99761, 99762, 99763, 99764, 99765, 99766, 99767, 99768, 99769, 99770, 99771, 99772, 99773, 99774, 99775, 99776,
99777, 99778, 99779, 99780, 99781, 99782, 99783, 99784, 99785, 99786, 99787, 99788, 99789, 99790, 99791, 99792, 99793, 99794,
99795, 99796, 99797, 99798, 99799, 99800, 99801, 99802, 99803, 99804, 99805, 99806, 99807, 99808, 99809, 99810, 99811, 99812,
99813, 99814, 99815, 99816, 99817, 99818, 99819, 99820, 99821, 99822, 99823, 99824, 99825, 99826, 99827, 99828, 99829, 99830,
99831, 99832, 99833, 99834, 99835, 99836, 99837, 99838, 99839, 99840, 99841, 99842, 99843, 99844, 99845, 99846, 99847, 99848,
99849, 99850, 99851, 99852, 99853, 99854, 99855, 99856, 99857, 99858, 99859, 99860, 99861, 99862, 99863, 99864, 99865, 99866,
99867, 99868, 99869, 99870, 99871, 99872, 99873, 99874, 99875, 99876, 99877, 99878, 99879, 99880, 99881, 99882, 99883, 99884,
99885, 99886, 99887, 99888, 99889, 99890, 99891, 99892, 99893, 99894, 99895, 99896, 99897, 99898, 99899, 99900, 99901, 99902,
99903, 99904, 99905, 99906, 99907, 99908, 99909, 99910, 99911, 99912, 99913, 99914, 99915, 99916, 99917, 99918, 99919, 99920,
99921, 99922, 99923, 99924, 99925, 99926, 99927, 99928, 99929, 99930, 99931, 99932, 99933, 99934, 99935, 99936, 99937, 99938,
99939, 99940, 99941, 99942, 99943, 99944, 99945, 99946, 99947, 99948, 99949, 99950, 99951, 99952, 99953, 99954, 99955, 99956,
99957, 99958, 99959, 99960, 99961, 99962, 99963, 99964, 99965, 99966, 99967, 99968, 99969, 99970, 99971, 99972, 99973, 99974,
99975, 99976, 99977, 99978, 99979, 99980, 99981, 99982, 99983, 99984, 99985, 99986, 99987, 99988, 99989, 99990, 99991, 99992,
99993, 99994, 99995, 99996, 99997, 99998, 99999]
Codul a durat 2.955139398574829
```

4.6 Descrierea părții experimentale

Aspectele ce apar în această parte pot fi:

- Cum şi de ce se structurează datele?;
- Cum se rulează algoritmul asupra unor probleme diferite?;
- Cum sunt rezultatele problemelor (Favorabile/Nefavorabile)?;

```
def interval(i,j):
    import random
    numere = [x for x in range(i,j)]
    random.shuffle(numere)
    return(numere)
l=[]
l=interval(1,100000)
print(l)

with open("fisier6.txt","w") as f:
    f.write(str(l))
    f.close()
from time import time
```

```
def lista(fisier1):
    l=[]
    n=[]
    with open(fisier1,"r") as f:
        l=f.read()
    l=l[:-1]
    l=l[1:]
    n=[int(i) for i in l.split(",")]
    return(n)
n1=lista("fisier1.txt")
print(n1)
```

În aceste două imagini este prezentat modul în care s-a generat lista de lungime 100000. Lista generânsu-se random într-un fișier txt. Din fișierul txt s-a extras conținutul și prin funcția listă, s-a generat lista finală.

Pe parcursul experimentului, tot programul a fost împărțit in subprograme care au fost apelate la momentul oportun. Împărțirea codului în funcții, ajută foarte mult la:

- Organizarea codului;
- Vizibilitatea acestuia;
- Evitarea cât mai multor erori;
- Evitarea rescrierii codului de fiecare dată;

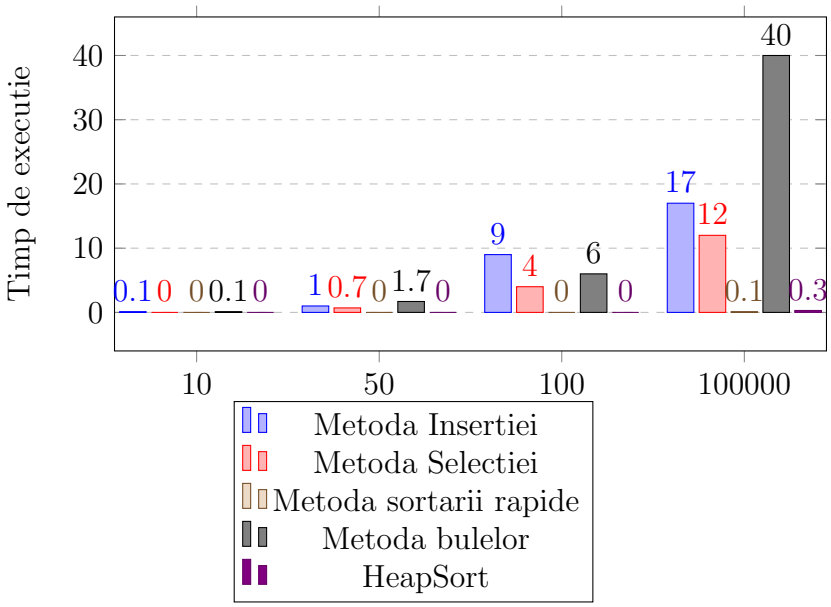
În momentul în care lista de lungime 100000 a fost sortată de fiecare algoritm de sortare prezentat pe parcursul lucrării, s-a generat pentru fiecare un timp de execuție diferit:

- Metoda Inserției (17 minute);
- Metoda Selecției (11 minute);
- Metoda Sortării Rapide (1 secundă);
- Metoda Bulelor (40 minute);
- HeapSort (2 secunde);

5 Concluzii şi direcţii viitoare

În concluzie , metodele de sortare enumerate mai sus, sunt eficiente pentru liste de lungimi mici. Când deja sortăm liste de lungimi mari(100000+),

lucrurile nu mai sunt așa frumoase. În această lucrare am putut observa că cele mai bune metode de sortare sunt sortările: "Quick Sort" și "Heap Sort". Acest lucru se poate observa din timpul de execuție pe care programul îl are atunci când sortăm liste mari. Ca și un top al metodelor de sortare enumerat mai sus, pe primul loc(cea mai bună metodă) este "Quick Sort", apoi "Heap Sort" , "Selection Sort", "Insertion Sort" si într-un final "Bubble Sort". Ca și directive viitoare, pot recomanda ca algoritmii sa fie implementați și testați de persoane au aceeași problemă. Timpul este în minute.



- Roșu-Metoda Inserției;
- Albastru-Metoda Selecției;
- Verde-Metoda Sortării Rapide;
- Violet-Metoda Bulelor;
- Portocaliu-HeapSort;

6 Bibliografie

Daniela Zaharie,"Introducere in proiectarea si analiza algoritmilor", Informatica, Algoritmi, 2008, ISBN: 97897367312119736731219, Description: 247 p. : fig., tab., 24 cm.