Sortiranje prebrajanjem (Counting sort) i Radix Sort

15. siječnja 2016.

Ante Mijoč

Uvod

Teorem

Ako je f(n) broj usporedbi u algoritmu za sortiranje temeljenom na usporedbama (eng. comparison-based sorting algorithm). Tada je $f(n) \in \Omega(n \log n)$.

Tada je očito i vremenska složenost samog algoritma $\in \Omega(n \log n)$.

Znamo da postoje algoritmi za sortiranje temeljeni na usporedbama čija je vremenska složenost u najgorem (Merge sort, Heapsort) i u prosječnom (Quicksort) $\in \mathcal{O}(n \log n)$.

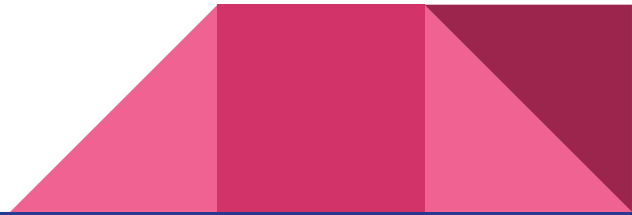
Tj. što se tiče takvih algoritama ne postoji bolji (do na konstantu).

Uvod

Međutim sortovi koji su tema ove prezentacije imaju linearnu složenost.

Naravno da bi to bilo uopće moguće, ne sortiramo uspoređivanjem.

Potrebne su nam određene pretpostavke o ulazu (ne radi za svaki ulaz).



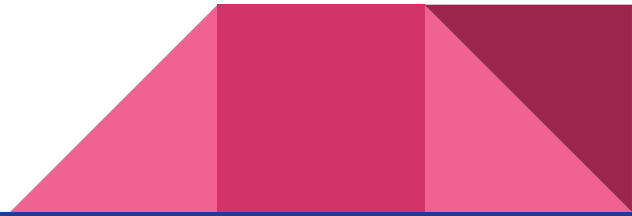
Sortiranje prebrojavanjem (Counting sort)

Pretpostavljamo da je ulaz n integera u rasponu od 0 do m.

A izlaz je sortiran niz.

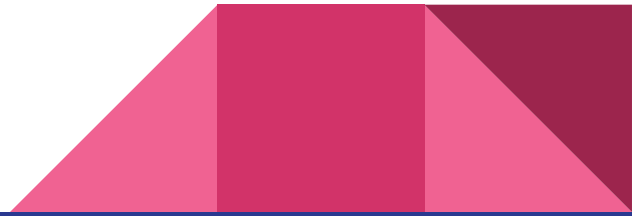
Koristimo i dva pomoćna niza, jedan duljine n, a drugi duljine m+1.

Oznake:
a-ulazni niz
b-pomoćni niz u kojem sortiramo (duljine n)
c-pomoćni niz za prebrojavanje (duljine m)



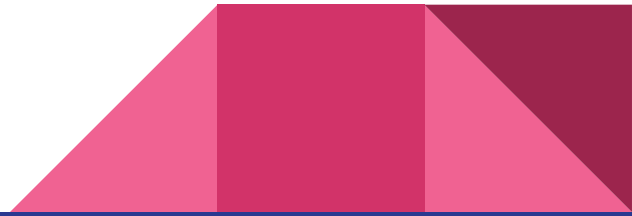
Pseudokod

```
counting_sort (a, m)
        for k = 0 to m                  //postavi na nulu

                c(k) = 0;

        for i = 1 to n                  //prebroji

                c(a[i]) += 1

        for k = 1 to m                  //sumiraj

                c(k) += c(k-1)
```



Pseudokod

```
for i = n downto 1                  //sortiraj
        b[c[a[i]] = a[i]
        c[a[i]] -= 1
for i = 1 to n                      //iskopiraj b u a
        a[i] = b[i]
```

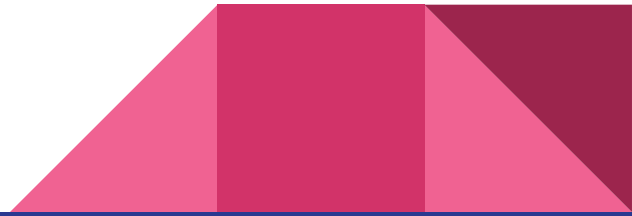


Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

m
7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
0	0	0	0	0	0	0	0



Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
0	1	0	0	0	0	0	0

Primjer

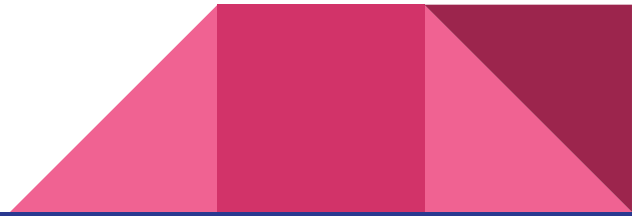
a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
0	1	0	0	1	0	0	0

Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
0	1	0	1	1	0	0	0



Primjer

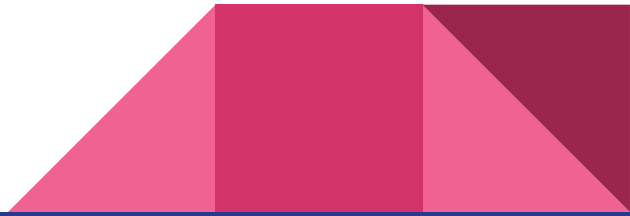
a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
1	1	0	1	1	0	0	0

Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

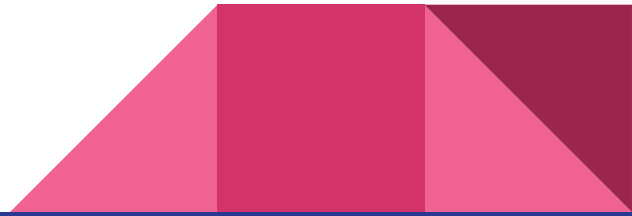
c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
1	2	0	1	1	0	0	0



Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

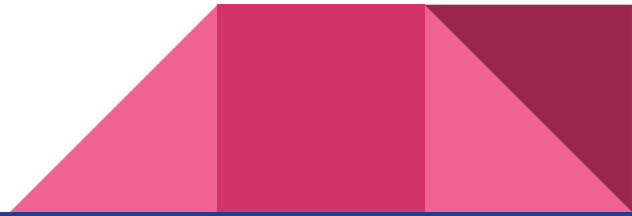
c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
1	2	0	1	1	0	0	1



Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
1	2	0	1	1	0	0	1



Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
1	3	0	1	1	0	0	1

Primjer

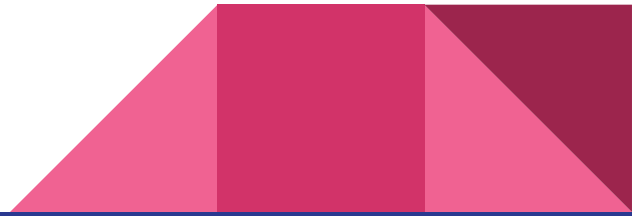
a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
1	3	3	1	1	0	0	1

Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
1	3	3	4	1	0	0	1



Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
1	3	3	4	5	0	0	1

Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
1	3	3	4	5	5	0	1

Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
1	3	3	4	5	5	5	1

Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
1	3	3	4	5	5	5	6

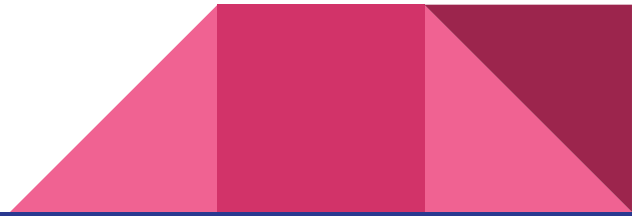
Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
1	3	3	4	5	5	5	6

Pseudokod (podsjetnik)

```
for i = n downto 1                  //sortiraj
        b[c[a[i]] = a[i]
        c[a[i]] -= 1
for i = 1 to n-1                    //kopiraj b u a
        a[i] = b[i]
```



Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
1	3	3	4	5	5	5	6

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]
*	*	*	*	*	7

Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
1	3	3	4	5	5	5	5

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]
*	*	*	*	*	7

Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
1	3	3	4	5	5	5	5

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]
*	*	1	*	*	7

Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
1	2	3	4	5	5	5	5

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]
*	*	1	*	*	7

Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
1	2	3	4	5	5	5	5

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]
0	*	1	*	*	7

Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
0	2	3	4	5	5	5	5

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]
0	*	1	*	*	7

Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
0	2	3	4	5	5	5	5

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]
0	*	1	3	*	7

Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
0	2	3	3	5	5	5	5

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]
0	*	1	3	*	7

Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
0	2	3	3	5	5	5	5

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]
0	*	1	3	4	7

Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
0	2	3	3	4	5	5	5

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]
0	*	1	3	4	7

Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
0	2	3	3	4	5	5	5

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]
0	1	1	3	4	7

Primjer

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
1	4	3	0	1	7

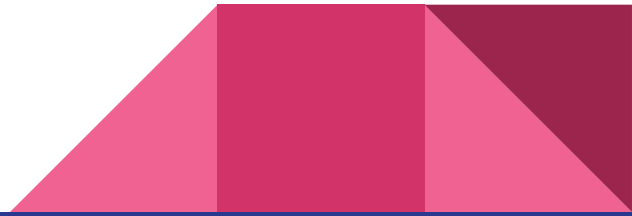
c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
0	1	3	3	4	5	5	5

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]
0	1	1	3	4	7

Primjer

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]
0	1	1	3	4	7

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
0	1	1	3	4	7



O sortiranju prebrojavanjem

Vremenska složenost algoritma je $\in \theta(m+n)$.
Jer uvijek ima točno 2m+4n elementarnih operacija (za svaki dopušteni ulaz).

Za razliku od algoritama za sortiranje temeljenim na usporedbama, sortirenje prebrojavanjem nije inplace sort. Tj. zahtjeva dodatnu memoriju za sortiranje.
Prostorna složenost je $\in \theta(m+n)$.

Primjetimo da ukoliko $m \leq n$ ili $m \in \mathcal{O}(n)$ da su pripadne složenosti $\theta(n)$.

Algoritam je korektan/potpun i stabilan (eng. stable).

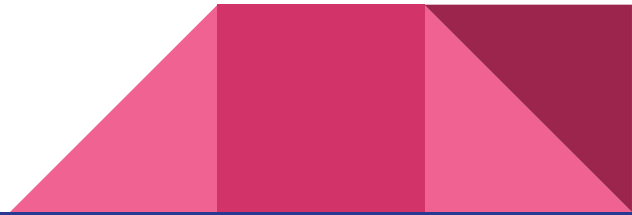
Radix sort

Pretpostavljamo da je ulaz n k-znamenkastih brojeva u bazi m.

A izlaz je sortiran niz.

Kao subrutinu koristimo sortiranje prebrojavanjem.

Oznake:
a-ulazni niz
b-pomoćni niz u kojem sortiramo (duljine n)
c-pomoćni niz za prebrojavanje (duljine m)



Pseudokod

```
radex_sort(a, k)
        for i = 1 to k
                counting_sort(a, m-1)          // kljuc (key) je i-ta najmanje znacajna znamenka
```

Primjer

Ulaz

13103	26440	16342	20101	801

Nakon prvog koraka

26440	20101	801	16342	13103

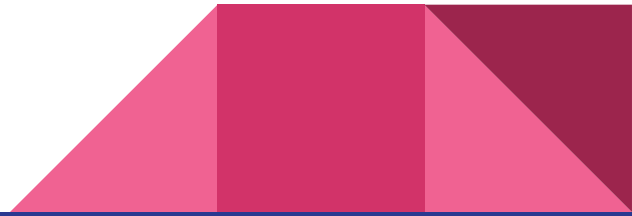
Primjer

Nakon drugog koraka

20101	801	13103	26440	16342

Nakon trećeg koraka

20101	13103	16342	26440	801



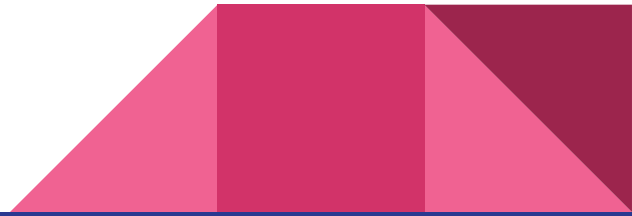
Primjer

Nakon četvrtog koraka

20101	801	13103	16342	26440

Nakon petog (zadnjeg) koraka

801	13103	16342	20101	26440



O radix sortu

Vremenska složenost algoritma je $\in \theta(km+kn)$.
Oznake:
n - duljina niza
k - max broj znamenki
m- baza

Prostorna složenost je $\in \theta(m+n)$.

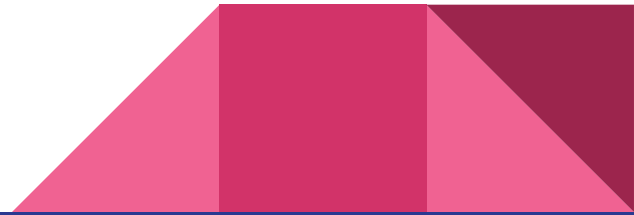
Ako su m konstantan ili $\in \mathcal{O}(n)$ i k konstantni tada su pripadne složenosti $\theta(n)$.
Algoritam je korektan/potpun stabilan (eng. stable).

Napomene

U Radix sortu umjesto sortiranja prebrojavanjem (Counting sort) možemo koristiti bilo koji drugi stabilan (stable) sort (npr. Bubble sort, Insertion sort, Merge sort).

Ako nam nije zadana maksimalna vrijednost jednim prolazom kroz niz je možemo pronaći. To očito neće utjecati na složenost.

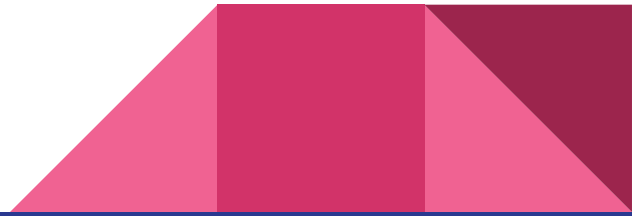
Prednost nad algoritmima za sortiranje temeljenim na usporebama je linearna složenost, a nedostaci su potreba za dodatnom memorijom.



Testiranje

Što se testiranja tiče uspoređivati ćemo vremena izvođenja implementacija Counting i Radix sorta međusobno i sa Quick sortom.

Ulaz su tekstualne datoteke s generiranim nizovima integera sa stranice www.random.org



C:\Users\djurdja\Documents\oaa_test\oaa_seminar_test.exe

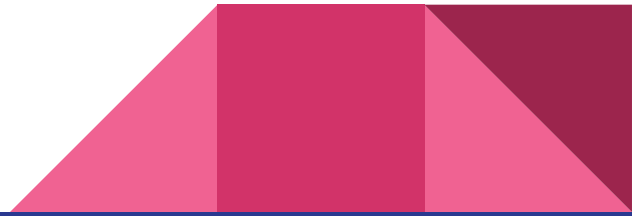
```
counting sort test_100_9.txt 0.000000
radix sort test_100_9.txt 0.000000
quick sort test_100_9.txt 0.000000
counting sort test_100_99.txt 0.000000
radix sort test_100_99.txt 0.000000
quick sort test_100_99.txt 0.000000
counting sort test_100_999.txt 0.000000
radix sort test_100_999.txt 0.000000
quick sort test_100_999.txt 0.000000
counting sort test_100_9999.txt 0.000000
radix sort test_100_9999.txt 0.000000
quick sort test_100_9999.txt 0.000000
counting sort test_1000_9.txt 0.000000
radix sort test_1000_9.txt 0.000000
quick sort test_1000_9.txt 0.000000
counting sort test_1000_99.txt 0.000000
radix sort test_1000_99.txt 0.000000
quick sort test_1000_99.txt 0.000000
counting sort test_1000_999.txt 0.000000
radix sort test_1000_999.txt 0.000000
quick sort test_1000_999.txt 0.000000
counting sort test_1000_9999.txt 0.000000
radix sort test_1000_9999.txt 0.000000
quick sort test_1000_9999.txt 0.000000
counting sort test_10000_9.txt 0.000000
radix sort test_10000_9.txt 0.000000
quick sort test_10000_9.txt 0.034000
counting sort test_10000_99.txt 0.004000
radix sort test_10000_99.txt 0.004000
quick sort test_10000_99.txt 0.006000
counting sort test_10000_999.txt 0.004000
radix sort test_10000_999.txt 0.005000
quick sort test_10000_999.txt 0.005000
counting sort test_10000_9999.txt 0.004000
radix sort test_10000_9999.txt 0.006000
quick sort test_10000_9999.txt 0.005000

Process returned 0 (0x0)   execution time : 0.277 s
Press any key to continue.
```

C:\Users\djurdja\Documents\oaa_test\oaa_seminar_test.exe

```
counting sort test_100_9.txt 0.092000
radix sort test_100_9.txt 0.106000
quick sort test_100_9.txt 0.090000
counting sort test_100_99.txt 0.079000
radix sort test_100_99.txt 0.093000
quick sort test_100_99.txt 0.125000
counting sort test_100_999.txt 0.094000
radix sort test_100_999.txt 0.094000
quick sort test_100_999.txt 0.094000
counting sort test_100_9999.txt 0.203000
radix sort test_100_9999.txt 0.125000
quick sort test_100_9999.txt 0.078000
counting sort test_1000_9.txt 0.375000
radix sort test_1000_9.txt 0.396000
quick sort test_1000_9.txt 0.588000
counting sort test_1000_99.txt 0.425000
radix sort test_1000_99.txt 0.469000
quick sort test_1000_99.txt 0.469000
counting sort test_1000_999.txt 0.458000
radix sort test_1000_999.txt 0.532000
quick sort test_1000_999.txt 0.515000
counting sort test_1000_9999.txt 0.547000
radix sort test_1000_9999.txt 0.625000
quick sort test_1000_9999.txt 0.547000
counting sort test_10000_9.txt 3.236000
radix sort test_10000_9.txt 3.496000
quick sort test_10000_9.txt 23.044000
counting sort test_10000_99.txt 3.567000
radix sort test_10000_99.txt 4.241000
quick sort test_10000_99.txt 6.205000
counting sort test_10000_999.txt 3.564000
radix sort test_10000_999.txt 4.967000
quick sort test_10000_999.txt 4.942000
counting sort test_10000_9999.txt 4.052000
radix sort test_10000_9999.txt 5.764000
quick sort test_10000_9999.txt 5.234000

Process returned 0 (0x0)   execution time : 79.543 s
Press any key to continue.
```

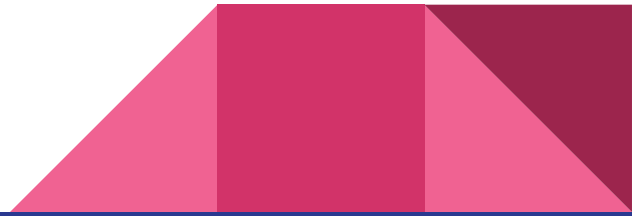
Counting sort

n\m	9	99	999	9999
100	0.092	0.079	0.094	0.203
1000	0.375	0.425	0.458	0.547
10000	3.236	3.567	3.564	4.052



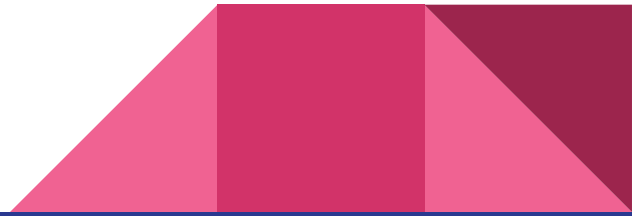
Radex sort

n\m	9	99	999	9999
100	0.106	0.093	0.094	0.125
1000	0.396	0.469	0.532	0.625
10000	3.496	4.241	4.967	5.764



Quick sort

n\m	9	99	999	9999
100	0.090	0.125	0.094	0.078
1000	0.588	0.469	0.515	0.547
10000	23.044	6.205	4.942	5.234



Literatura

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, *Introduction to Algorithms,* 2000.

[2] Counting sort, https://en.wikipedia.org/wiki/Counting_sort (5.1.2016.)

[3] Radex sort, https://en.wikipedia.org/wiki/Radix_sort (5.1.2016.)

[4] Counting sort and Radex sort, http://www.opendatastructures.org/ods-java/11_2_Counting_Sort_Radix_So.html (5.1.2016.)