SORTIRANJE PREBRAJANJEM I RADIX SORT

Petar Bratulić

Seminar iz kolegija Oblikovanje i analiza algoritama
Prirodoslovno – matematički fakultet
Zagreb, 21. siječnja 2019.

UVOD

Mergesort, Heapsort, Quicksort – složenost $O(n \log n)$

Sva dosad obrađena sortiranja imaju zajedničko svojstvo: sortirani poredak temelji se samo na uspoređivanju između elemenata.

Sva takva soritranja moraju u najgorem slučaju napraviti $\Omega(n \log n)$ uspoređivanja da bi sortirali polje od n elemenata.

Može li bolje od toga?

Da, ali uz neke prepostavke o podacima, koriste podatke na neki specifičan način.

Soritranje prebrajanjem i Radix Sort – linearna složenost

SORTIRANJE PREBRAJANJEM

Pretpostavka na ulazne podatke - svaki od n ulaznih elemenata je neki cijeli broj u rasponu od 1 do nekog cijelog broja *k*.

Kada je $k = O(n)$, složenost algoritma je $O(n)$.

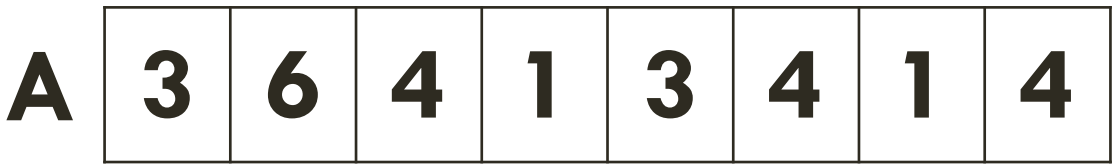
Osnovna ideja: za svaki element *x* pronaći koliko je elemenata manjih od *x*.

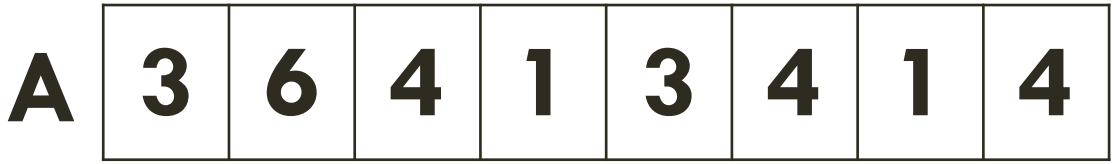
Ta informacija se koristi kako bi element *x* izravno smjestili na njegovu poziciju u polju koje sortiramo.

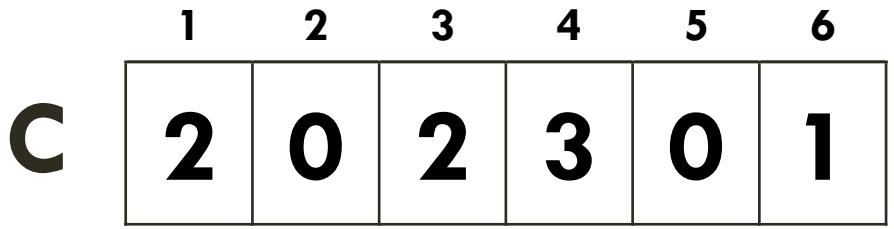
Ulazni podaci: polje A[1 . . n] (ulazni podaci), k (najveći broj koji se pojavljuje u A).

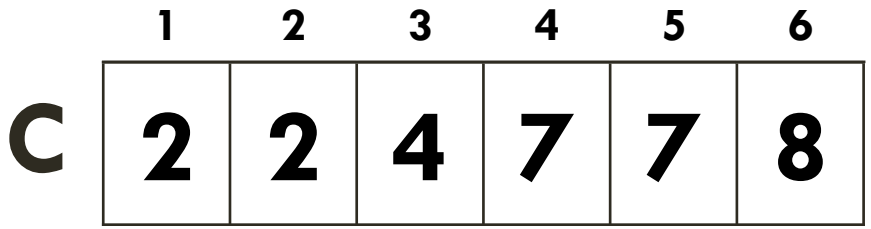
Potrebna su još pomoćna polja B[1 . . n] (polje u koje se kopiraju podaci u soritiranom poretku, nakon soritranja B se kopira u A) i C[1 . . k] u koje spremamo broj ponavljanja svakog elementa iz A.

PRIMJER: POLJE DULJINE 8, K = 6

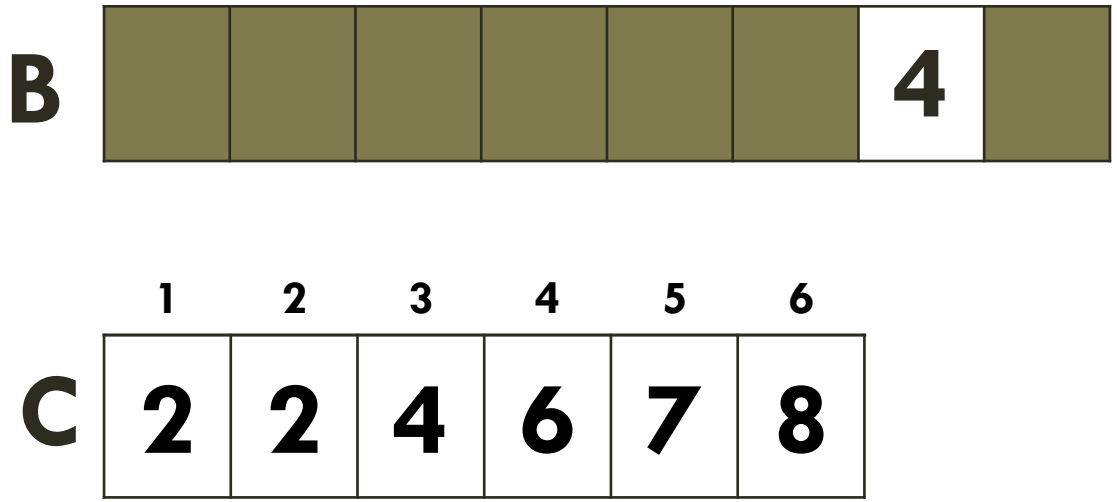


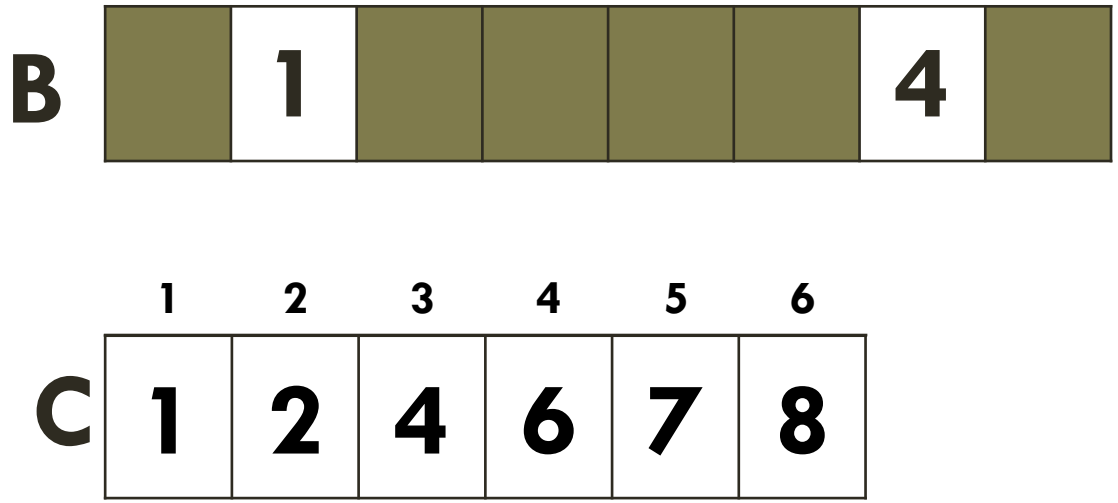


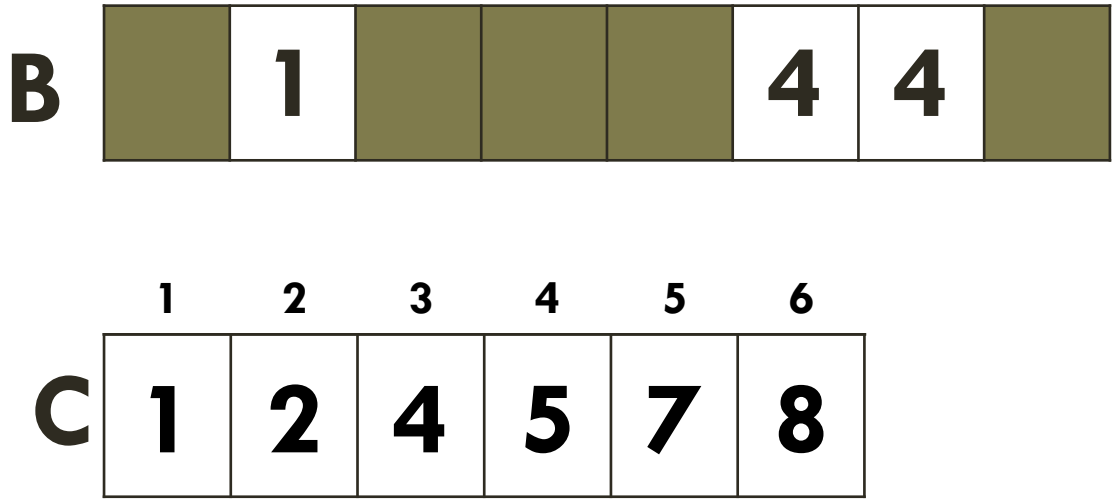


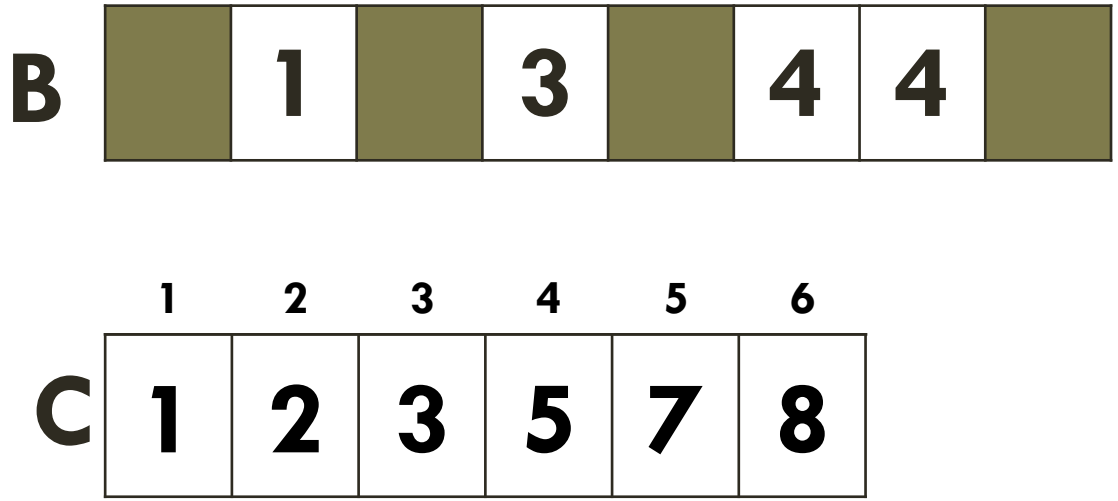


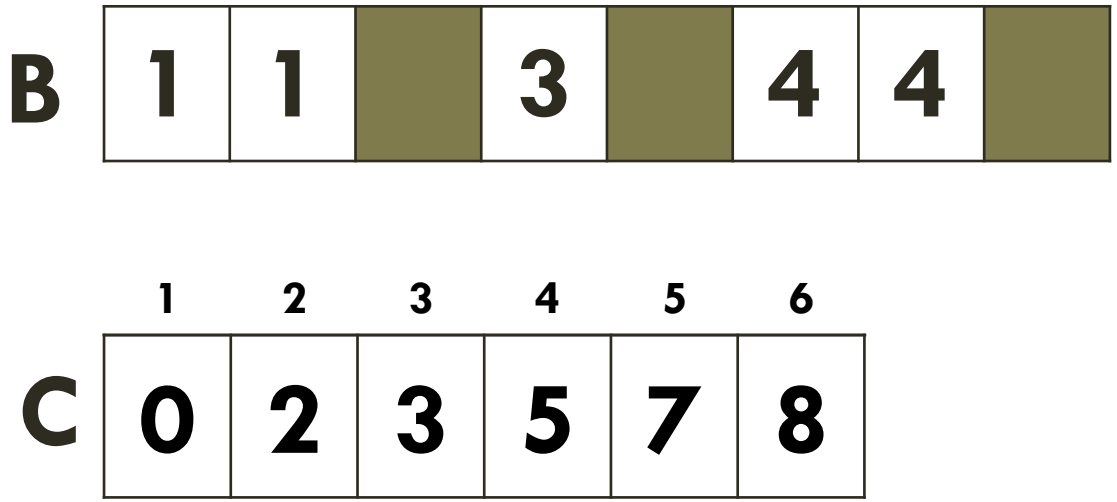
Elementi se soritraju tako da se prvo uzima element s najvećim indeksom: prvo se uzima 4, pa dalje 1, 4, 3...

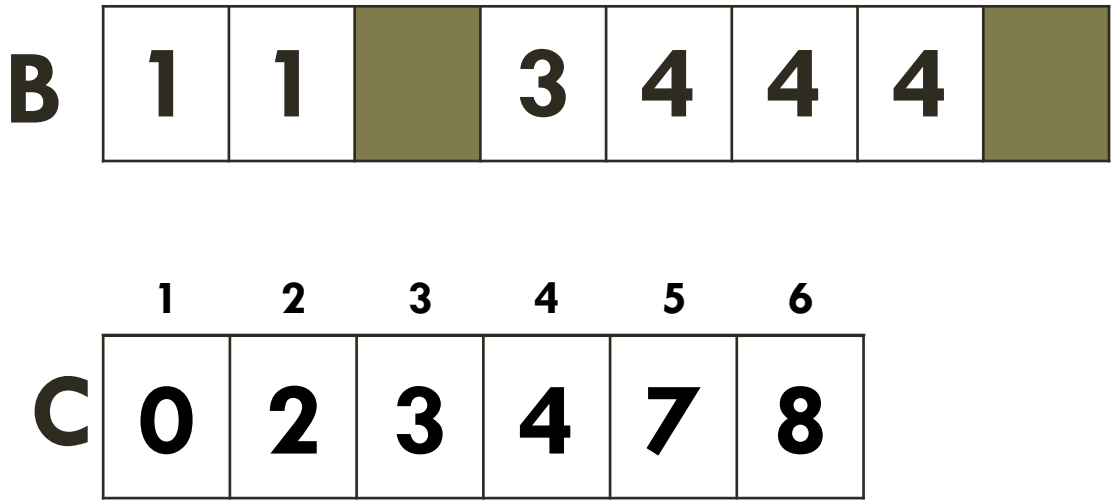

B
4
1 2 3 4 5 6
C
2 2 4 6 7 8

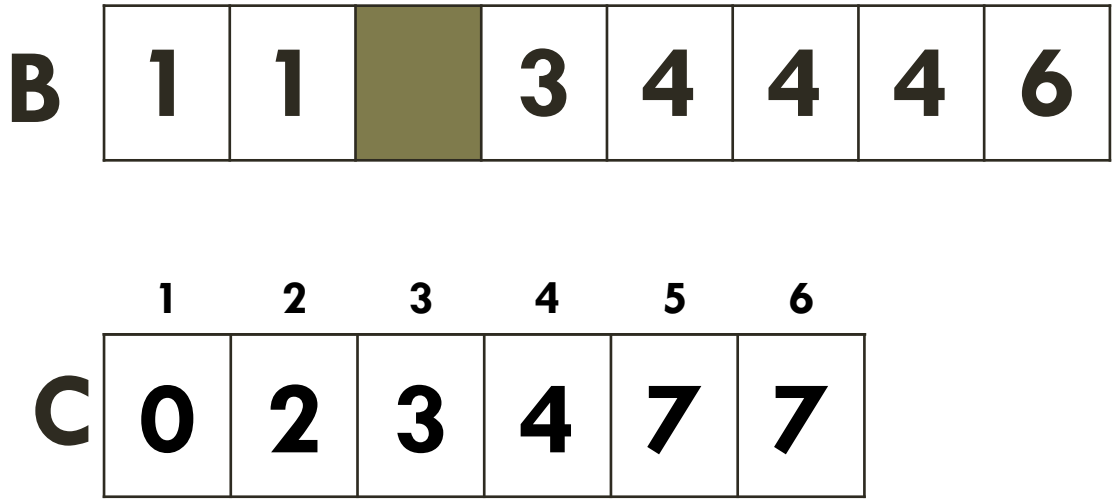

B
1
4
1 2 3 4 5 6
C
1 2 4 6 7 8

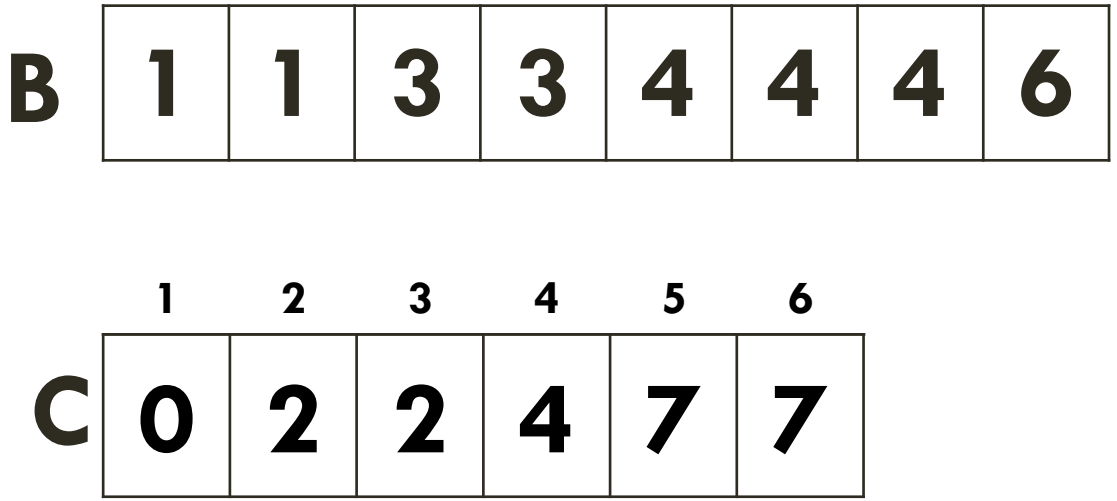

B
1
4 4
1 2 3 4 5 6
C
1 2 4 5 7 8


B
1
3
4 4
1 2 3 4 5 6
C
1 2 3 5 7 8


B
1 1 3 4 4
1 2 3 4 5 6
C
0 2 3 5 7 8


B
1 1 3 4 4 4
1 2 3 4 5 6
C
0 2 3 4 7 8


B
1 1 3 4 4 4 6
1 2 3 4 5 6
C
0 2 3 4 7 7


B
1 1 3 3 4 4 4 6
1 2 3 4 5 6
C
0 2 2 4 7 7

ALGORITAM

Algoritam soritra polje integera A[1 .. n] u rasponu od 1 do k.

```
counting_sort(A,k){
1       //prvo izracunaj C
2       // postavi C[i] = broj ponavljanja vrijednosti i u polju A
3       // prvo incijaliziraj C na 0
4       for i = 1 to k
5               C[i] = 0
6       n = A.last
7       for j = 1 to n
8               C[A[j]] = C[A[j]] + 1
9       // modificiraj C t.d. je C[i] = broj elementa ≤ i
10      for i = 2 to k
11              C[i] = C[i] + C[i-1]
12      // sortiraj A i spremaj rezultat u B
13      for i = n downto 1 {
14              B[C[A[i]]] = A[i]
15              C[A[i]] = C[A[i]] – 1
16      }
17      //kopiraj B u A
18      for j = 1 to n
19              A[j] = B[j]
20 }
```

SLOŽENOST SORTIRANJA PREBRAJANJEM

Koliko je vremena potrebno za izvršavanje algoritma?

Kod izračunavanja polja C za petlje **for** linije koda 4-5 (inicijalizacija polja C) i 10-11 (računanje broja elemenata manjih od i-tog) treba vremena $O(k)$.

Petlje **for** linije koda 7-8 (broj ponavljanja vrijednosti i u A), 13-16 (soritranje A i kopiranje u B), 18-19 (kopiranje B u A) traju $O(n)$ vremena.

Ukupno trajanje algoritma $O(k+n)$.

U praksi koristimo soritranje prebrajanjem kada je $k \leq n$ ili $k = O(n)$, tada je složenost algoritma $O(n)$.

STABILNOST SORTIRANJA PREBRAJANJEM

Soritranje prebrajanjem je stabilno, tj. elementi jednakih vrijednosti u ulaznom polju u jednakom su poretku i u izlaznom polju.

Dokaz stabilnosti:

Preptostavimo da su dva elementa , A[s] and A[s+1], u ulaznom polju takva da je $A[s]=A[s+1]$, $1 \leq s \leq n-1$.

Nakon izvršavanja glavne for petlje u soritranju prebrajanjem, imamo $B[p]=A[s+1]$ *i* $B[p-1]=A[s]$, $2 \leq p \leq n$. *A[s] i A[s+1] pojavljuju se u polju B u istom poretku kao i u ulaznom polju A. Zaključujemo, sortiranje prebrajanjem je stabilno.*

RADIX SORT

Problem: sortiranje velikih brojeva sa sortiranjem prebrajanjem (npr. $k = n^2$).

Rješenje: Radix Sort

Pretpostavke na ulazne podatke: pozitivni nenegativni brojevi koji imaju najviše *d* znamenaka.

Prvo se pomoću nekog **stabilnog** sortiranja (soritranje prebrajanjem) sortiraju najmanje značajne znamenke brojeva (znamenke jedinica).

Nakon toga analogno se sortiraju znamenke desetica te posljednje se nekim stabilnim sortiranjem soritraju najznačajnije znamenke.

PRIMJER: POLJE VELIČINE 6, D = 5

A	33101	26440	16341	2332	20101	801

Raspored znamenki jedinica je: 1 0 1 2 1 1
Soritranjem prebrajanjem dobiva se: 0 1 1 1 1 2

A	26440	33101	16341	20101	801	2332

Znamenke desetica: 4 0 4 0 0 3 → 0 0 0 3 4 4

A	33101	20101	801	2332	26440	16341

Znamenke stotica: 1 1 8 3 4 3 → 1 1 3 3 4 8

A	33101	20101	2332	16341	26440	801

Znamenke tisućica: 3 0 2 6 6 0 → 0 0 2 3 6 6

A	20101	801	2332	33101	16341	26440

Znamenke desettisućica: 2 0 0 3 1 2→ 0 0 1 2 2 3

A	801	2332	16341	20101	26440	33101

ALGORITAM

Algoritam soritra polje integera A[1 . . n], svaki element polja ima najviše *d* znamenaka.

radix_sort(A,d){

for i = 1 to d

do pomoću stabilnog sorta sortiraj polje A na znamenci i

}

Stabilna sortiranja: soritranje prebrajanjem, *merge sort,insertion sort*

SLOŽENOST RADIX SORTA

Analiza složenosti ovisi o stabilnom sortu koji se koristi u algoritmu.

Ako je svaka znamenka u rasponu od 1 do k i k nije prevelik, logično je uzeti sortiranje prebrajanjem.

Svaki prolaz u for petlji Radix Sorta ima složenost $\Theta(n + k)$.

Ukupno ima d prolaza pa je složenost algoritma $\Theta(d\ n + k\ d)$.

Kada je d konstanta i $k = O(n)$, Radix Sort ima linernu složenost.

Korektnost algoritma?

DOKAZ KOREKTNOSTI RADIX SORTA

Neka je *x* element ulaznog polja A[1 .. n]. Pokazujemo da nakon izvršavanja algoritma, svaki element čija je vrijednost veća od *x* slijedi nakon *x*.

Neka je $y = b_i 10^i + b_{i-1} 10^{i-1} + \ \ldots + b_0$, $b_i \ \neq 0$ prikaz nekog broja većeg od *x*.

Dalje, neka je $x = \ a_i 10^i + a_{i-1} 10^{i-1} + \ldots + a_0$.

Neka je k najveći indeks takav da je $b_k \neq a_k$. Obzirom da vrijedi $y > x$ i $b_k > a_k$ kada sortiranje prebrajanjem sortira na 10^k-tom mjestu, *y* se postavlja nakon *x*. U sljedećim pozivima sortiranja prebrajanjem, *x* i *y* imaju isti ključ i obzirom da je sortiranje prebrajanjem stabilno, *y* dolazi u sortiranom polju A nakon *x*.

- Smatra se da je broj bitova u jednom wordu $\theta\big(lg(n)\big)$. Radi konkretnosti, pretpostavimo da je $d * lg(n)$ broj bitova, gdje je d pozitivna konstanta.
- Tada, ako se svaki broj koji treba sortirati stane u jedan word, možemo broj smatrati d-znamenkastim u radix-n notaciji.
- Za primjer uzmimo da sortiramo milijun brojeva od kojih je svaki 64-bitni. Ako ih tretiramo kao 4-znamenkaste, radix-2^{16} brojeve, možemo ih soritrati u samo 4 prolaza Radix sortom, dok ako uzmemo neko sortiranje koje koristi uspoređivanje potrebno je $lg(n) = 20$ operacija po broju da bi se soritralo polje.
- Nedostatci: Radix sort koji koristi soritranje prebrajanjem kao stabilan sort zauzima puno memorije. Soritranje prebrajanjem koristi dodatna polja za sortiranje pa ako nije dostupno puno memorije, sortiranja koja koriste uspoređivanje davat će brže rezultate.

TESTIRANJA

Testiranja su provedena na računalu s karakteristikama:

- Procesor: Intel(R) Core(TM) i5-3230M 2.60 GHz
- RAM: 4,00 GB
- 64-bitni operacijski sustav

Programski jezik: C++

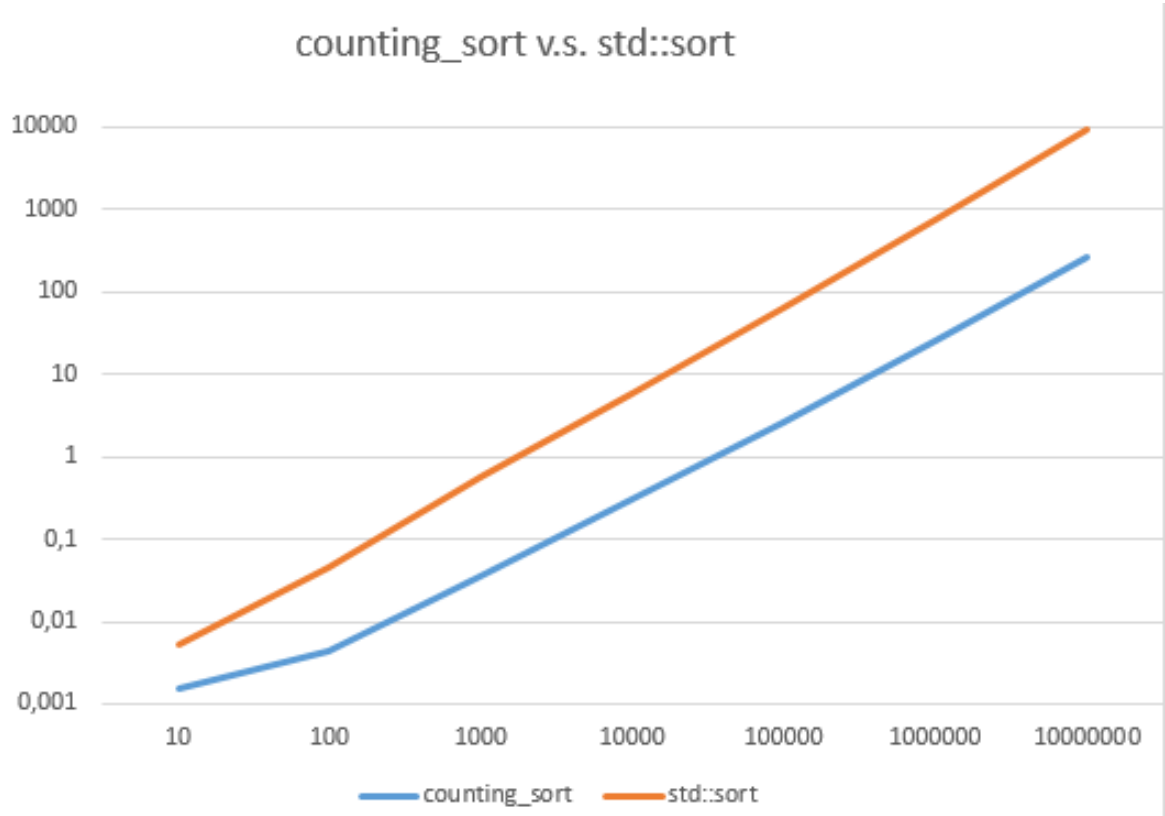
Rezultati testiranja prikazani su u tablicama u milisekundama. Testirnja su provedena na slučajno generiranim brojevima 100 puta.

Testirano je:

- usporedba između brzine sortiranja prebrajanjem i std::sort-a
- brzine sortiranja prebrajanjem za različite k
- uspredba brzine Radix sort-a sa std::sort-om kada su brojevi 4-znamenkasti i kada su 8-znamenaksti

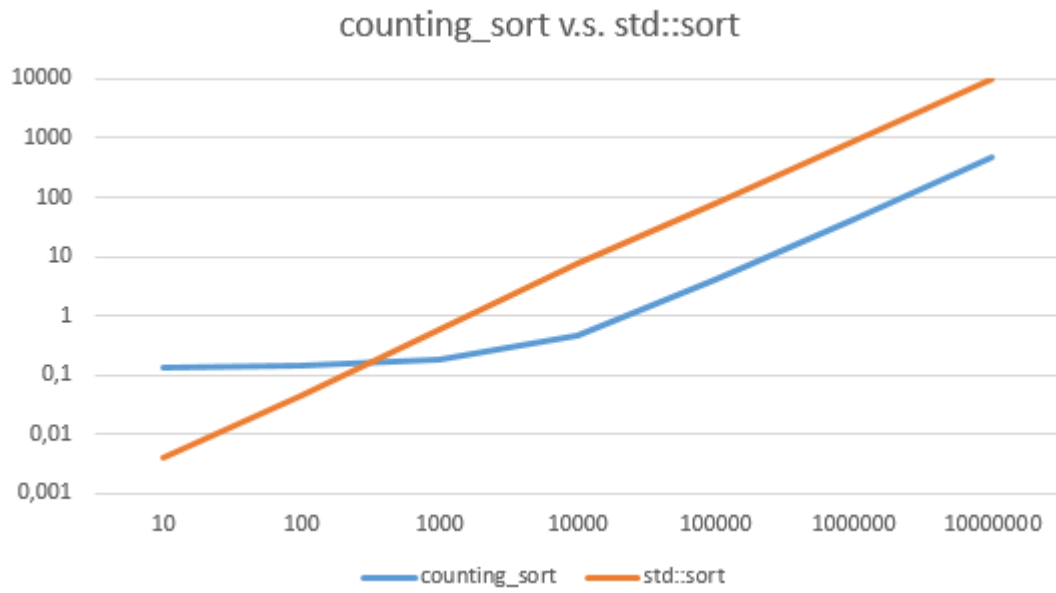
USPOREDBA: SORTIRANJE PREBRAJANJEM I STD:SORT (K=10)

n	counting_sort	std::sort	relativna brzina
10	0,00157501	0,00532108	3,3784
100	0,00433818	0,0452726	10,4359
1 000	0,034712	0,554853	15,9845
10 000	0,304786	5,92389	19,4362
100 000	2,5778	65,5202	25,4171
1 000 000	25,87	766,635	29,6341
10 000 000	259,338	9055,48	34,9177



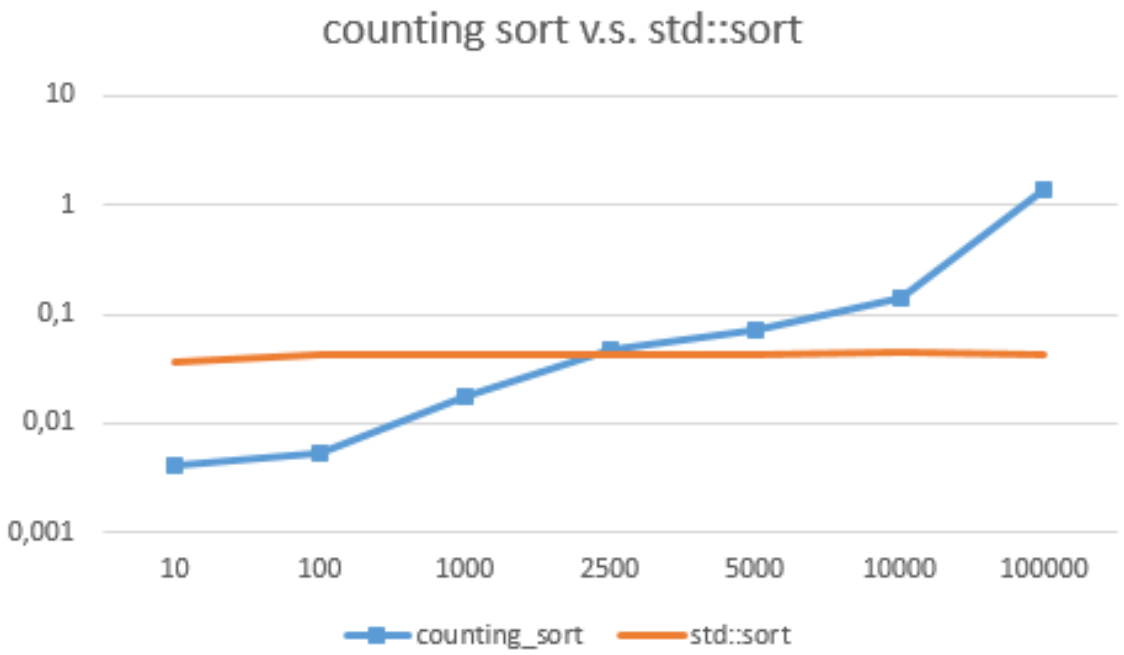
USPOREDBA: SORTIRANJE PREBRAJANJEM I STD:SORT (K=1000)

n	counting_sort	std::sort	relativna brzina
10	0,135155	0,003999	0,02958
100	0,139655	0,042561	0,30475
1000	0,174036	0,60185	3,45819
10 000	0,445838	7,69141	17,2515
100 000	4,12618	79,5014	19,2675
1 000 000	43,5635	889,951	20,4288
10 000 000	458,854	9797,68	21,3524



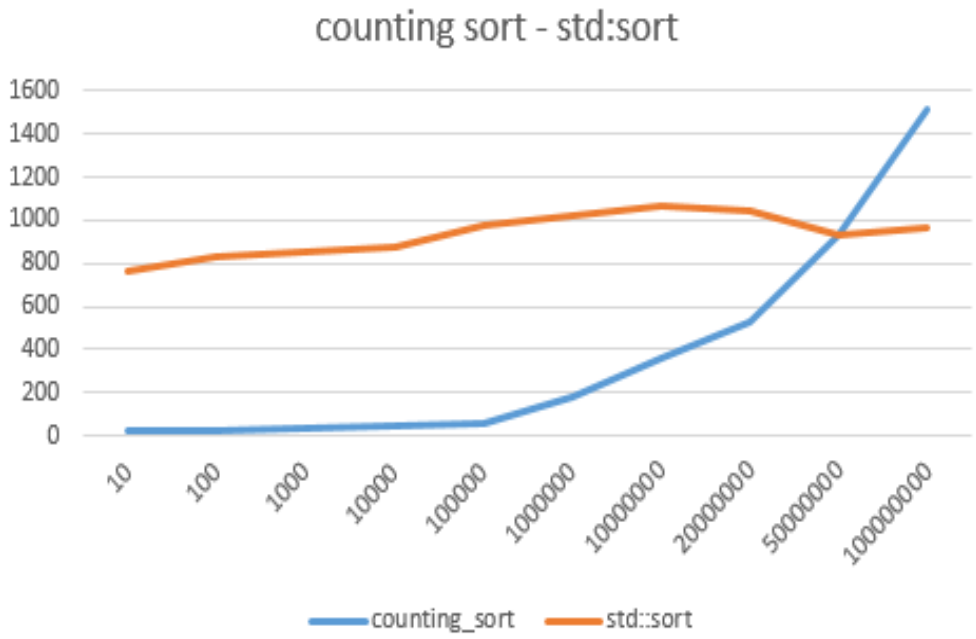
USPOREDBA: SORTIRANJE PREBRAJANJEM I STD:SORT (N=100)

k	counting_sort	std::sort	relativna brzina
10	0,00403	0,03675	9,1265
100	0,00529	0,04160	7,8709
1000	0,01786	0,04196	2,3490
2500	0,04658	0,04161	0,8933
5000	0,07055	0,04200	0,5953
10 000	0,13961	0,04415	0,3163
100 000	1,39974	0,04222	0,0302



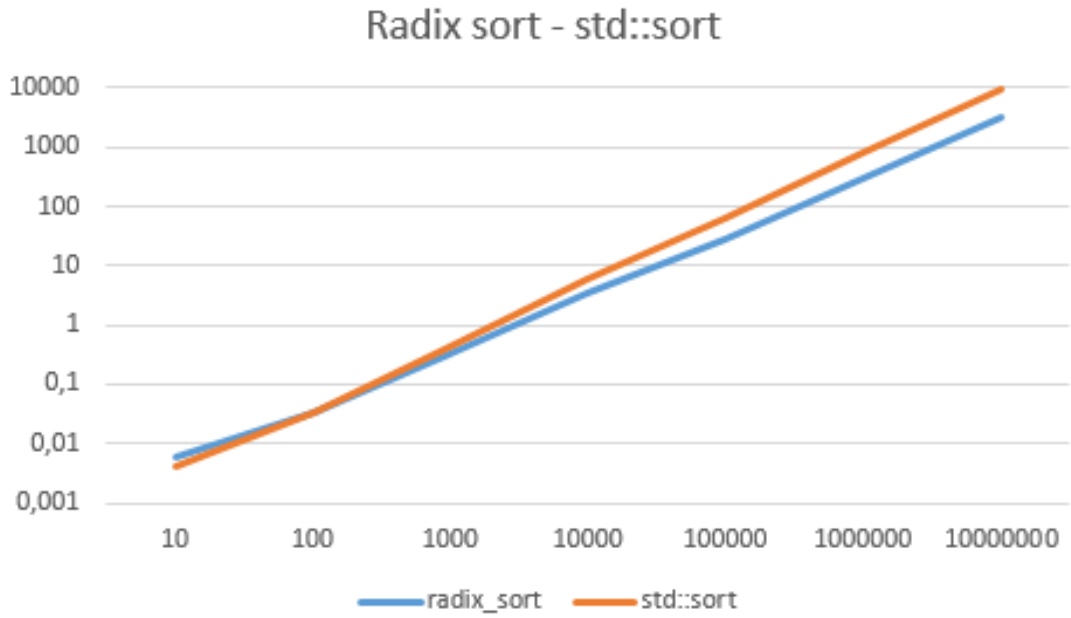
USPOREDBA: SORTIRANJE PREBRAJANJEM I STD:SORT (N=1 000 000)

k	counting_sort	std::sort	relativna brzina
10	25,3893	768,956	30,2866
100	26,8555	828,574	30,8530
1000	30,2761	858,464	28,3545
10 000	43,4015	875,423	20,1703
100 000	57,8513	982,776	16,9880
1 000 000	181,576	1027,45	5,6585
10 000 000	366,115	1065,8	2,9111
20 000 000	523,271	1048,64	2,0040
50 000 000	929,407	929,076	0,9996
100 000 000	1514,18	964,537	0,6370



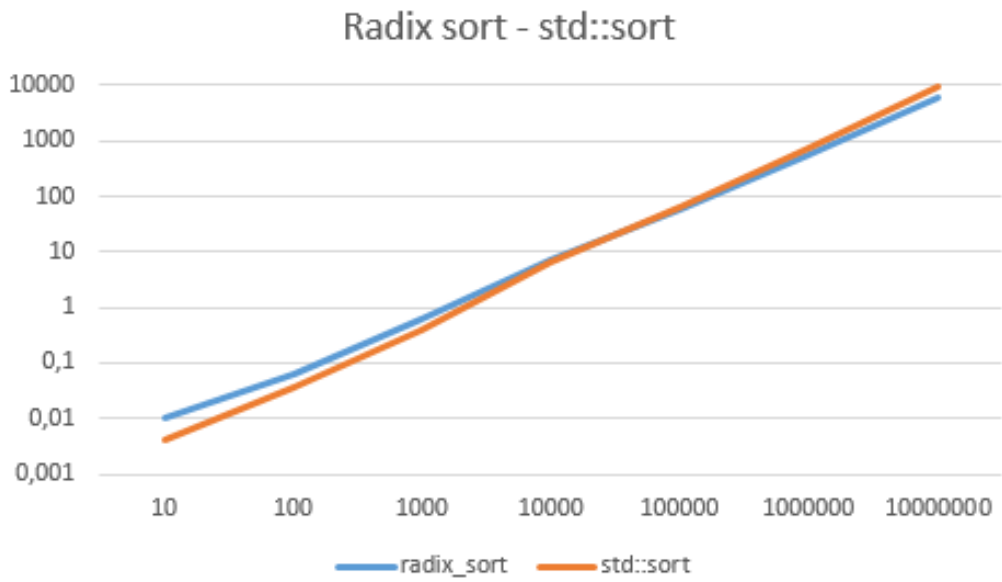
USPOREDBA: RADIX SORT I STD::SORT (D=4)

n	radix_sort	std::sort	relativna brzina
10	0,0061974	0,00405397	0,6541
100	0,0336572	0,0349778	1,0392
1000	0,318396	0,419292	1,3169
10 000	3,46022	5,92711	1,7129
100 000	29,2095	64,8247	2,2193
1 000 000	297,162	801,857	2,6984
10 000 000	3171,87	9954,93	3,1385



USPOREDBA: RADIX SORT I STD::SORT (D=8)

n	radix_sort	std::sort	relativna brzina
10	0,0098961	0,00404212	0,4085
100	0,0644608	0,0367344	0,5699
1000	0,610017	0,398919	0,6539
10 000	6,88679	6,53003	0,9482
100 000	57,91	64,5476	1,1146
1 000 000	576,267	766,631	1,3303
10 000 000	5863,5	9291,01	1,5846



LITERATURA

[1] R. Johnsonbaugh, M. Schaefer, *Algorithms*, Pearson Education International, Upper Saddle River, New Jersey, 2004.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Intrdouction to algorithms*, The MIT Press, 2000.

[3] https://probablydance.com/2016/12/02/investigating-radix-sort/ [zadnje pristupljeno: 16.1.2019]