STUDENTSKA RUBRIKA

Sortiranje podataka

Alfonzo Baumgartner* Stjepan Poljak†

Sažetak. *Ovaj rad prikazuje jedno od rješenja problema sortiranja podataka u jednodimenzionalnom polju (nizu) elemenata. U praksi se često pojavljuje potreba za sortiranjem podataka te se zbog toga traži što efikasniji i brži algoritam. U ovom radu detaljno je prikazan jedan od najboljih – merge-sort algoritam. Napravljene su i praktične izvedbe algoritama za sortiranje koje su testirane na različitim skupovima podataka.*

Ključne riječi: *algoritmi za sortiranje, merge-sort, rekurzivni algoritmi*

Data sorting

Abstract. *In this paper one solution of the problem of sorting an one-dimensional vector (array) of data is shown in this paper. A need for data sorting very often arises in the practice and therefore we are constantly searching for more efficient and faster sorting algorithms. One of the best sorting algorithms – merge-sort is shown in this paper in detail. We made two of sorting algorithms and tested them on different types of data.*

Key words: *sorting algorithms, merge-sort, recursive algorithms*

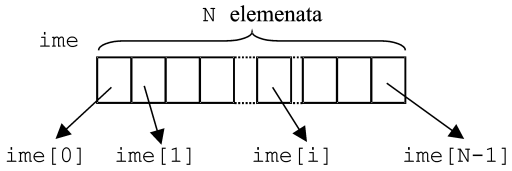
1. Uvod

1.1. Nizovi kao strukture podataka

Jednodimenzionalno polje (niz) je linearna struktura podataka u koju možemo smjestiti više podataka istog tipa. Primjerice u programskom jeziku C, indeksiranje niza počinje od 0, tj. prvi element u nizu je na mjestu 0, a zadnji na mjestu $N-1$, gdje je N duljina niza.

*Elektrotehnički Fakultet, Sveučilište u Osijeku, Kneza Trpimira 2b, HR-31 000 Osijek, e-mail: baumgart@etfos.hr

†Odjel za matematiku, Sveučilište u Osijeku, Gajev trg 6, HR-31 000 Osijek, e-mail: spoljak@hotmail.com



Slika 1. *Shematski prikaz niza u programskom jeziku C*

Ako želimo koristiti niz, potrebna je i naredba deklaracije koja izgleda ovako:

tip ime[N];

gdje su:
tip – *tip podataka* niza, jednostavni (int, char, float...) ili izvedeni;
ime – *ime* niza, zadaje se kao i ime obične varijable (skalara);
N – konstanta, tj. prirodni broj koji označava *duljinu* niza.
Općenito, niz je potpuno definiran s gore navedenom trojkom: *tipom*, *imenom* i svojom *duljinom*.

U ovom radu, niz je osnovna struktura podataka. Ako su elementi niza međusobno usporedivi, onda se takav niz elemenata može *sortirati*. Sortirati znači poredati elemente niza od najmanjeg do najvećeg, tj. "uzlazno" ili od najvećeg do najmanjeg, tj. "silazno". Dva spomenuta načina sortiranja su dualni, a to znači ako riješimo jedan problem, vrlo se lako dolazi i do rješenja drugoga. Zato ćemo mi u ovom radu promatrati samo problem "uzlaznog" sortiranja.

1.2. Rekurzivne funkcije

Kako bi objasnili jedan od najboljih algoritama za sortiranje danas, moramo se najprije upoznati s pojmom rekurzivnosti: Svaka funkcija koja unutar svoje definicije poziva samu sebe naziva se *rekurzivna funkcija*. Kako ne bi došlo do neograničenog broja poziva, tj. kopija takve funkcije u memoriji uvijek mora postojati i neki nerekurzivni put izlaska iz takve funkcije. *Rekurzivnim stablom* možemo prikazati rad rekurzivne funkcije kako bi si lakše predočili kako jedan rekurzivni algoritam zapravo funkcionira. Većina programskih jezika, pa tako i primjerice C, podržava rekurzije kod funkcija. Dan je i primjer jedne vrlo jednostavne rekurzivne funkcije.

Primjer 1. *Napisati rekurzivnu C funkciju koja će ispisati niz od N cjelobrojnih podataka.*

Rješenje:

```
void Ispis( int V[], int i, int N)
{
  if (i>=N || i<0) return;
  printf( "%d\n", V[i] );
  Ispis( V, i+1, N );
}
```

Ova funkcija sastoji se od tri naredbe. Prva naredba predstavlja nerekurzivni izlazak iz funkcije i osigurava ispravan rad. Druga naredba ispisuje i-ti element niza

na ekran, dok je treća naredba rekurzivni poziv iste te funkcije i zbog toga je *Ispis* rekurzivna funkcija.
Zadatak za vježbu: što se mijenja u radu ove funkcije ako zamijenimo mjesta drugoj i trećoj naredbi?

2. Algoritmi za sortiranje

U znanosti je problem sortiranja još uvijek aktualan i traže se bilo kakve mogućnosti ubrzanja, tako da se nedavno došlo do otkrića algoritma "intro-sort" koji je kombinacija dvaju već poznatih algoritma za sortiranje, a sam je brži od svakog pojedinačno. U najnižu klasu spadaju najjednostavniji, ali i najsporiji algoritmi kao što su to primjerice algoritami "bubble-sort" ili "selection-sort", dok u najvišu klasu spadaju algoritmi poput "quick-sort" i "merge-sort". Cilj ovog rada jest ukazati na razlike ovih dviju klasa algoritama.

Kod svih ovih algoritama ulazni parametar je isti – niz V od N elemenata, a problem je sortirati elemente niza "uzlazno".

2.1. Selection-sort algoritam

Ideja jednog od najjednostavnijih algoritama "selection-sort" se sastoji u sljedećem: u zadanom nizu pronaći svaki element koji je manji od onog koji se nalazi na prvom mjestu i ako je manji zamijeniti ga s tim elementom koji je na prvom mjestu. Tako ćemo na kraju dobiti najmanji element na prvom mjestu. Postupak se tako dalje ponavlja za preostale elemente od drugog do zadnjeg mjesta. Znači dobivamo poredan niz od najmanjeg elementa na prvom mjestu, dok će na zadnjem mjestu ostati najveći element.

Vro je jednostavno ostvariti ovakav algoritam u bilo kojem programskom jeziku. Evo jednog rješenja u programskom jeziku C:

```
for (i=0; i<N-1; i++)
  for (j=i+1; j<N; j++)
    if (V[i]>V[j]) { t=V[i]; V[i]=V[j]; V[j]=t; }
```

Primjedba: lako je dobiti i algoritam koji sortira "silazno", dovoljno bi bilo u trećem retku znak '>' prebaciti u '<'.

Analiza rada: Sada trebamo odrediti složenost ovoga algoritma, tj. naći nekakvu mjeru njegove brzine i odrediti o čemu ta brzina ovisi. Jasno je da duljina niza N ima presudan značaj. Manji značaj ima i tip elemenata niza, jer za neke složenije tipove podataka usporedba je puno kompliciranija nego za neke jednostavnije tipove podataka. Nadalje, u ovoj jednostavnoj analizi, gledat ćemo samo utjecaj duljine niza N.

U prvoj iteraciji vanjske petlje po brojaču i, imamo $N-1$ ponavljanja unutarnje petlje po brojaču j. Svaki sljedeći put imamo jedno ponavljanje manje. Sada lagano možemo izračunati broj ponavljanja U_s:

$$U_s = (N-1) + (N-2) + \cdots + 3 + 2 + 1 = \frac{(N-1)N}{2} = \frac{N^2 - N}{2}.$$

Ovu jednakost je lako dokazati ako zbrajamo prvi član sume sa zadnjim, drugi s predzadnjim itd.

Ne smijemo nikako zanemariti onaj N^2 koji smo dobili u brojniku rezultata. Vidjet ćemo i na praktičnim primjerima na kraju ovog rada kakav je značaj toga kvadrata na brzinu izvođenja sortiranja. Naposlijetku, zaključimo da je algoritam "selection-sort" tzv. "kvadratne složenosti", tj. nazovimo tako ovu klasu sporijih algoritama za sortiranje. Vidjet ćemo da ni to što se od tih N^2 oduzima N i to sve dijeli s dva ne pomaže puno, tj. i dalje imamo 'dosta spor' algoritam.

2.2. Merge-sort algoritam

Upotrijebljena je dobro poznata ideja "podijeli pa vladaj" na sljedeći način: Niz V se podijeli na lijevi i desni dio otprilike po polovici, nakon toga se rekurzivno sortira posebno lijevi i posebno desni dio, te se nakon toga niz ponovno spaja u jedan sortirani niz. Spajanje (engl. merge) je sada jednostavno jer je lijevi i desni dio već sortiran. Rad algoritma se može opisati ovako: niz se prepolovi, pa se nakon toga i te polovice prepolove, sve dok se ne dođe do niza duljine 1. Polovice duljine 1 se dalje ne raspolavljaju jer one već jesu sortirani nizovi, postupak spajanja će napraviti ostatak posla.

Osnovna ideja se može malo detaljnije opisati sljedećim algoritmom:

Sort(V,d,g)

- **ako je** $g - d <= 1$ **kraj**
- $s = \lfloor \frac{d+g}{2} \rfloor$
- Formiraj $VL = \{V_d, \ldots, V_s\}$
- Formiraj $VD = \{V_{s+1}, \ldots, V_g\}$
- Sort(VL,d,s)
- Sort(VD,s+1,g)
- V=Merge(VL,VD)

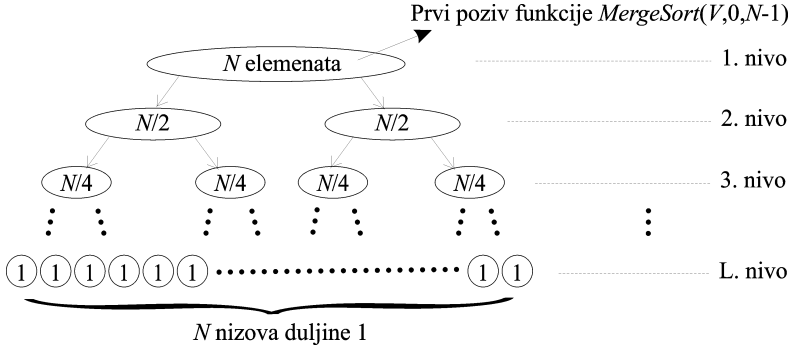
Sort je rekurzivna funkcija koja sortira niz V sa zadanom donjom granicom d i gornjom granicom g. Prvu točku možemo opisati kao nerekurzivni izlazak iz funkcije, od druge do četvrte točke imamo razdvajanje niza V na lijevi VL i desni VD dio. U petoj i šestoj točki imamo dva rekurzivna poziva iste funkcije samo s malo drugačijim parametrima. U sedmoj točki funkcije *Sort* nije detaljno opisan postupak spajanja lijevog i desnog dijela niza koji su već sortirani natrag u V. Postupak spajanja detaljno je opisan u sljedećem primjeru izvedbe "merge-sort" algoritma u programskom jeziku C:

```
void MergeSort( tippod V[], int d, int g )
{
  if (d==g) return; // Nerekurzivni izlazak iz funkcije
  int s = (g-d)/2;
  int n1=s+1, n2, i;
  if ((g-d)%2==1) n2=n1; else n2=n1-1;
```

```
  tippod *VL = new tippod[n1];
  tippod *VD = new tippod[n2];
// Rastavi niz V na lijevi i desni dio VL i VD
  for (i=0; i<n1; i++) VL[i] = V[d+i];
  for (i=0; i<n2; i++) VD[i] = V[d+i+s+1];
  MergeSort( VL, 0, n1-1 ); // Rekurzivni poziv (Sortiraj lijevi dio)
  MergeSort( VD, 0, n2-1 ); // Rekurzivni poziv (Sortiraj desni dio)

// Spoji VL i VD u jedan V. (Merge)
  i=0;
  int k, i1=0, i2=0;
  for (k=0; k<n1+n2; k++)
  {
    if (i1<n1 && i2<n2) {
      if (VL[i1]<VD[i2]) V[dg+i++]=VL[i1++];
          else V[dg+i++]=VD[i2++];
        }
    else {
      if (i1<n1) V[dg+i++]=VL[i1++];
          else V[dg+i++]=VD[i2++];
        }
  }
  delete [] VL;
  delete [] VD;
}
```

Analiza rada: Pokušajmo sada opet u terminima duljine niza N odrediti složenost, tj. brzinu rada ovakvog algoritma. Najprije moramo reći da dio koji spaja (merge) lijevi i desni dio ukupne duljine M napravi to u točno M ponavljanja. To se očigledno može vidjeti iz gore navedenog primjera u C-u gdje se petlja ponavlja točno $n1+n2$ puta, gdje su $n1$ i $n2$ redom duljine lijevog i desnog dijela bilo kojeg podniza zadanog niza V. Međutim, kako odrediti ukupni broj ponavljanja kada imamo rekurzivnu funkciju?! To je općenito teži problem, ali u ovom slučaju dosta će nam pomoći rekurzivno stablo prikazano na *Slici 2.*



Slika 2. *Prikaz rekurzivnog stabla koje generira rekurzivna funkcija MergeSort prilikom izvođenja*

Rekurzivno stablo nam pomaže tako što možemo utvrditi da se na svakom nivou stabla uvijek nalazi ukupno N elemenata niza bez obzira da li je niz V razbijen na 2 ili više dijelova, pa sve do zadnjeg nivoa gdje se mogu naći samo nizovi duljine 1.

Već smo rekli da se svaka dva dijela moraju spajati u jedan veći i koliko to traje. To znači da po svakom nivou imamo najviše N ponavljanja jer svi podnizovi skupa imaju najviše N elemenata po nivou. Broj nivoa označen je s L, pa imamo ukupni broj ponavljanja U_m za "merge-sort" algoritam od

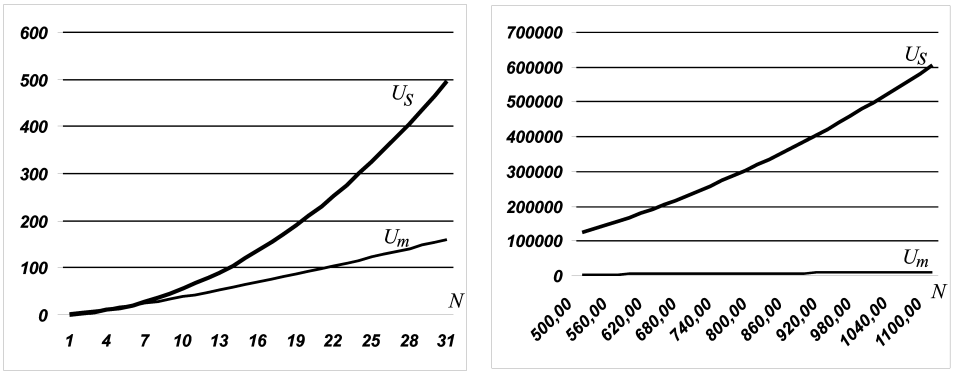
$$U_m = N \cdot L.$$

Ostaje nam još samo utvrditi ovisi li L o ukupnom broju elemenata u nizu N. Lako se pokaže da ovisi i egzaktno se može izračunati da je uvijek $L = \lceil \log_2 N \rceil$, pa tako dobivamo konačno

$$U_m = N \cdot \lceil \log_2 N \rceil.$$

Ova analiza rada je točna za svaki N koji je potencija broja 2. Za ostale N analiza je malo kompliciranija, ali se dobije isti rezultat. Ovakvu klasu algoritama nazovimo "$N \log N$".

Usporedimo sada ukupne brojeve ponavljanja za merge sort U_m i za selection sort U_s. Lako se vidi da su oni sličnog reda veličine samo za dosta male duljine nizova N, međutim, za sve veće vrijednosti broja N (tj. za sve dulje nizove), razlika postaje sve veća u korist U_m (tj. merge sort je brži). Odnos ovih dviju veličina najbolje pokazuje *Slika 3.*



Slika 3. *Ovisnost ukupnog broja ponavljanja o duljini niza N*

Sa *Slike 3* jasno je vidljivo da su opisana dva algoritma usporediva samo za vrlo male vrijednosti N. Znači, teorijski rezultat jest da za svaki veći N uvijek trebamo koristiti "merge-sort" algoritam, jer je višestruko brži.

3. Eksperimentalni rezultati

Na različitim tipovima podataka testirani su algoritmi "selection-sort" i "merge-sort". Mjereno je vrijeme[1] izvođenja za različite duljine nizova N i dobiveni su rezultati prikazani u Tablicama 1 i 2.

[1] Testiranje je izvedeno na računalu s procesorom Intel Pentium 4, 2.40GHz i RAM memorijom od 256MB.

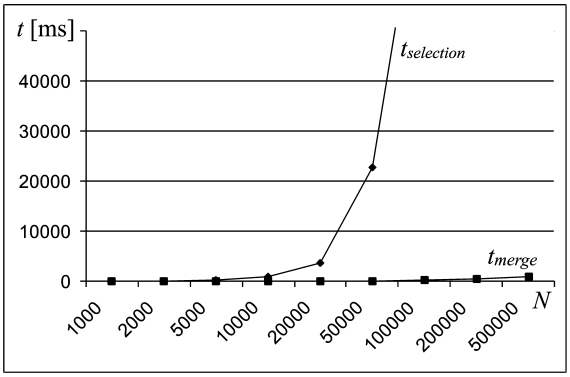
N	1000	2000	5000	10000	20000	50000	100000	200000	500000
$t_{selection}$ [ms]	16	47	219	922	3672	22656	86094	299297	1425812
t_{merge} [ms]	< 1	< 1	15	15	31	78	171	344	922

Tablica 1. Rezultati sortiranja realnih brojeva tipa float

N	1000	2000	5000	10000	20000	50000	100000	200000	500000
$t_{selection}$ [ms]	31	125	719	2453	8391	50297	189140	686500	4286922
t_{merge} [ms]	< 1	15	15	16	32	109	234	516	1469

Tablica 2. Rezultati sortiranja nizova znakova (riječi)

Na *Slici 4* prikazani su dobiveni rezultati za sortiranje realnih brojeva (*Tablica 1*) grafički. Vidljivo je da ovi eksperimentalni rezultati odgovaraju teorijskim koji su analizirani u prethodnoj točki s prikazom na *Slici 3*.



Slika 4. *Izmjereni rezultati izvršavanja algoritama za sortiranje na nizovima realnih brojeva različite duljine*

4. Zaključak

Iako je "selection-sort" algoritam naizgled puno jednostavniji, "merge-sort" je puno efikasniji što je pokazano u ovom radu i teorijski i eksperimentalno. No, ipak trebamo naposlijetku utvrditi da "merge-sort" ipak ima malo uvećane memorijske zahtjeve za razliku od "selection-sort" koji koristi samo zadani niz V. Algoritam "merge-sort" osim zbog svoje rekurzivnosti (sjetimo se da svaki rekurzivni poziv stvara novu kopiju te funkcije u memoriji sa svim njenim internim varijablama) unutar same funkcije stvara još jedan niz osim onog koji je došao kao parametar. Taj uvećani memorijski zahtjev ipak nije toliko značajan kolika je razlika u brzini izvođenja što se dobro vidi iz eksperimentalnih rezultata pogotovu za veće duljine niza N.

Ova dva algoritma nisu jedini korišteni u praksi algoritmi za sortiranje, ali su jako reprezentativni, jer svi ostali zapravo spadaju u jednu od dvije klase složenosti ovdje prezentirane ("kvadratna" i "$N \log N$" složenost) ili im je brzina negdje "između". Među ostalima, tu su još od poznatijih "bubble-sort", "quick-sort", "heap-sort", "insert-sort".

Najbrži danas korišten algoritam za sortiranje je "introspective-sort". Taj hibridni algoritam zapravo je kombinacija dvaju već poznatih "quick-sort" i "heap-sort". Naime, kada program detektira da se nekoliko puta za redom nailazi na loš raspored elemenata za "quick-sort" i da bi u tom slučaju došlo do "kvadratnog" očekivanog vremena izvođenja, on se prebacuje na "heap-sort" koji je tada bitno bolji. Više o tome i sam algoritam može se pogledati u [2].

Naposlijetku, spomenimo da podaci ne moraju biti uvijek smješteni u nizovima linearno u memoriji (znači jedan iza drugoga) već mogu biti smješteni i u tzv. vezanim listama, a baš o tome načinu smještanja podataka također ovisi odabir najboljeg algoritma za sortiranje. Ovdje navedeni "merge-sort" algoritam zapravo postiže još bolje rezultate kada su podaci smješteni u vezanim listama, a ne u nizovima (jednodimenzionalnim poljima). Što su vezane liste i kako se podaci smještaju u njih nadilazi tematiku ovoga rada pa ih nećemo detaljnije objašnjavati.

Literatura

[1] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, C. STEIN, *Introduction to Algorithms, 2nd edition*, MIT Press, Massachusetts, 2001, 123–164.

[2] D. R. MUSSER, *Introspective Sorting and Selection Algorithms*, Software – Practice and Experience **(8)**(1997), 983–993.

[3] D. KNUTH, *The Art of Computer Programming, Sorting and Searching, Second Edition*, Reading, Massachusetts: Addison-Wesley, 1998.