Programiranje 1
13. predavanje

Saša Singer

singer@math.hr

web.math.pmf.unizg.hr/~singer

PMF – Matematički odsjek, Zagreb

Sadržaj predavanja

- Pretraživanje i sortiranje nizova (nastavak):
  - Pretraživanje — sekvencijalno i binarno (ponavlj.).
  - Sortiranje izborom ekstrema — Selection sort.
  - Razne varijante Selection sorta.
  - Složenost sortiranja — općenito.
  - Složenost Selection sorta.
  - Sortiranje zamjenama susj. elemenata — Bubble sort.
  - Poboljšana varijanta Bubble sorta.
  - Složenost Bubble sorta.
- Završni primjeri (ponavljanje za kolokvij):
  - Zadatak 1.
  - Zadatak 2.

Informacije

Zadnji petak, 26. 1., sam isto “tu” od 10 sati

- i možete me “pitati” (u uredu sam, ili ispred faksa).

Konzultacije od 12 sati, također, “rade” — slobodno dođete!

Informacije — kolokviji

Programiranje 1 je u kolokvijskom razredu F3.

- Drugi kolokvij: petak, 9. 2. 2018., u 15 sati.
- Popravni kolokvij: petak, 23. 2. 2018., u 15 sati.

Informacije — upis ocjena i usmeni

Upisi ocjena i usmeni (po želji/“kazni”):

- pogledati obavijest na rezultatima kolokvija!

Uobičajeno, dogovor za usmeni je u vrijeme uvida (ne kasnije).

Za upis ocjena, lijepo molim:

- doći u nekom od navedenih termina, bit će ih dovoljno!

Ako ne možete (bolesni i sl.) — javite se (mailom).

- Nemojte dolaziti izvan termina, i
- nemojte “zaboraviti” na ocjenu.

Postoji rok za predaju ocjena, zbog upisa.

Naknadni upis ocjene = naknadni potpis = molba!

Informacije — bodovi za zadaće

Ne zaboravite, “žive” su i domaće zadaće na adresi

`http://degiorgi.math.hr/prog1/ku/`

Dodatni bodovi “čekaju na vas”.

Bitno: Aplikacija za “zadaće” se

- zaključava s početkom drugog kolokvija.

Nakon toga,

- nema više novih prijava, ni predaje zadataka.

U tom trenu vrijedi:

- Tko je “unutra” i koliko je predao/la . . . , to je to,

i nema iznimaka!

Pretraživanje nizova (ponavljanje)

Sadržaj

- Pretraživanje nizova (ponavljanje):
  - Sekvencijalno pretraživanje.
  - Složenost sekvencijalnog pretraživanja.
  - Binarno pretraživanje sortiranog niza.
  - Složenost binarnog pretraživanja.

Pretraživanje nizova (ponavljanje)

Problem pretraživanja — opća formulacija:

- Treba provjeriti nalazi li se zadani element `elt` među članovima zadanog niza

$$x_0,\ x_1,\ \ldots\ ,\ x_{n-1}.$$

Drugim riječima, pitanje glasi:

- postoji li indeks $i \in \{0, \ldots, n-1\}$ takav da je `elt` $= x_i$.

Ako niz nije sortiran, tj. u nizu vlada "nered", koristimo

- sekvencijalno pretraživanje ("jedan po jedan").

Sekvencijalno pretraživanje — funkcija

Funkcija koja vraća odgovor:

```
int seq_search(int x[], int n, int elt)
{
    int i;

    for (i = 0; i < n; ++i)
        if (x[i] == elt)
            return 1;

    return 0;
}
```

Koristimo "skraćenu" pretragu, bez varijable `nasli`.

Sekvencijalno pretraživanje — složenost

Složenost pretraživanja mjerimo brojem usporedbi

- "jednak", odnosno, "različit" (jer nema uređaja),

i to samo u tipu za članove niza. Operacije na indeksima ne brojimo — njih ima podjednako kao i usporedbi.

U najgorem slučaju, moramo provjeriti sve članove niza, tj.

$$\text{broj usporedbi} \leq n.$$

Ova mjera složenosti je dobra procjena za trajanje izvršavanja, algoritma sekvencijalnog pretraživanja — oznaka $T(n)$.

Zapis za trajanje:

$$T(n) \in O(n).$$

Značenje: trajanje, u najgorem slučaju, linearno ovisi o n.

Točno značenje zapisa složenosti

Točno matematičko značenje zapisa

$$T(n) \in O(f(n))$$

za neke funkcije T i f (sa skupa $\mathbb{N}$ u skup $\mathbb{R}$):

Postoji konstanta $c \in \mathbb{R}$ i postoji $n_0 \in \mathbb{N}$, takvi da, za svaki $n \in \mathbb{N}$, vrijedi implikacija

$$n \geq n_0 \implies T(n) \leq c \cdot f(n).$$

Prijevod: T raste sporije od "neka konstanta puta f", za sve dovoljno velike n.

Napomena. Često se piše $T(n) = O(f(n))$, što nije korektno, jer ova "jednakost" nije simetrična!

Binarno pretraživanje

Ako je niz uzlazno ili silazno sortiran,

$$x_0 \leq x_1 \leq \cdots \leq x_{n-1} \quad \text{ili} \quad x_0 \geq x_1 \geq \cdots \geq x_{n-1},$$

potraga se može drastično ubrzati, tako da koristimo tzv.

- binarno pretraživanje — pretraživanje "raspolavljanjem".

Binarno pretraživanje — funkcija

Funkcija koja vraća odgovor ("skraćeni" oblik):

```
int binary_search(int x[], int n, int elt) {
    int l = 0, d = n - 1, i;
    while (l <= d) {
        i = (l + d) / 2;
        if (elt < x[i])
            d = i - 1;
        else if (elt > x[i])
            l = i + 1;
        else
            return 1;
    }
    return 0; }
```

Binarno pretraživanje — složenost

Koliko traje najdulja potraga (= ako element nismo našli)?

- nakon 1. podjele — duljina niza za potragu je $\leq \dfrac{n}{2}$
- nakon 2. podjele — duljina niza za potragu je $\leq \dfrac{n}{4}$
- nakon k-te podjele — duljina niza za potragu je $\leq \dfrac{n}{2^k}$.

Zadnji smo prolaz napravili kad je duljina pala strogo ispod 1

$$\frac{n}{2^k} < 1,$$

dakle, $n < 2^k$, odnosno, $k > \log_2 n$. Pritom, stajemo za najmanji takav k — tj., kad je $2^{k-1} \leq n < 2^k$.

Binarno pretraživanje — složenost (nastavak)

Složenost opet mjerimo brojem usporedbi, ali sada koristimo

- "manji (ili jednak)", odnosno, "veći (ili jednak)",

jer imamo uređaj među elementima i niz je sortiran po tom uređaju. Operacije na indeksima, opet, ne brojimo.

U najgorem slučaju, za broj raspolavljanja k vrijedi

$$2^{k-1} \leq n < 2^k,$$

ili

$$k \leq 1 + \lfloor \log_2 n \rfloor.$$

Svako raspolavljanje ima najviše 2 usporedbe, pa je

$$\text{broj usporedbi} \leq 2 \cdot (1 + \lfloor \log_2 n \rfloor).$$

Binarno pretraživanje — složenost (nastavak)

Zapis za trajanje:

$$T(n) \in O(\log n).$$

Značenje: trajanje, u najgorem slučaju, logaritamski ovisi o n.

Primjer. U sortiranom telefonskom imeniku s 10^6 osoba, dovoljno je samo 20 raspolavljanja!

Zaključak: Sortiramo zato da bismo brže tražili!

Sortiranje nizova

Sadržaj

- Sortiranje nizova (polja):
  - Sortiranje izborom ekstrema — Selection sort.
  - Razne varijante Selection sorta.
  - Složenost sortiranja — općenito.
  - Složenost Selection sorta.
  - Sortiranje zamjenama susjednih elemenata — Bubble sort.
  - Poboljšana varijanta Bubble sorta.
  - Složenost Bubble sorta.

Problem sortiranja nizova

Zadan je niz od n objekata

$$x_0,\ x_1,\ \ldots\ ,\ x_{n-1},$$

koje možemo uspoređivati relacijom uređaja $\leq$ ili $\geq$.

Problem sortiranja niza:

- Treba preurediti zadani niz — tako da članovi budu
  - uzlazno ($\nearrow$) poredani: $x_0 \leq x_1 \leq \cdots \leq x_{n-1}$, ili
  - silazno ($\searrow$) poredani: $x_0 \geq x_1 \geq \cdots \geq x_{n-1}$.

Za "preuređivanje" niza spremljenog kao polje koristimo

- zamjene članova niza, tj. njihovih vrijednosti: $x_i \leftrightarrow x_j$.

U nastavku, ako nije drugačije rečeno, pretpostavljamo da niz treba uzlazno sortirati.

Sortiranje nizova izborom ekstrema

Ideja: Koristimo usporedbe i zamjene elemenata u nizu.

- Dovedi najmanji element niza $x_0, x_1, \ldots, x_{n-1}$ na njegovo pravo mjesto.
- To mjesto je prvo u cijelom nizu, pa je (nakon zamjene), nova vrijednost elementa x_0, upravo, najmanji element niza.
- Postupak ponovi na skraćenom (nesređenom) nizu $x_1, \ldots, x_{n-1}$ (duljine za jedan manje, tj. $n - 1$).
- Niz se "skraćuje" sprijeda.
- To ponavljamo sve dok ne "stignemo" na niz sa samo jednim elementom (x_{n-1}) — taj je sigurno sortiran!

Naziv algoritma: "izbor" ekstrema $\longrightarrow$ Selection sort.

Sortiranje nizova izborom ekstrema (nastavak)

Na početku algoritma imamo nesređeni niz, tj. indeks prvog elementa u nesređenom dijelu je 0.

Algoritam uzlaznog sortiranja izborom ekstrema ima dva "građevna bloka":

- Za $i = 0$, sve dok je $i < n - 1$, ponavljaj:
  - U nesređenom dijelu niza (indeksi od i do $n - 1$) nađi najmanji element.
  - Najmanji element zamijeni s prvim elementom x_i nesređenog dijela niza (ako već nije na indeksu i).

  Nakon ovog koraka, nesređeni dio niza se smanjio za 1, tj. prvi element nesređenog dijela sad ima indeks $i + 1$.

Sortiranje nizova izborom ekstrema (nastavak)

"Građevni blok": u nesređenom dijelu niza (od i do $n-1$) nađi najmanji element (pogledati prošlo predavanje).

- Inicijalizacija: trenutno najmanji element u nesređenom dijelu je prvi element. Njegov indeks je $\texttt{ind\_min} = i$, a vrijednost $\texttt{x\_min} = x_i$.
- Za elemente s indeksima od $j = i + 1$ do $j = n - 1$ ispitaj je li $x_j < \texttt{x\_min}$.
- Ako je, zapamti novu minimalnu vrijednost $\texttt{x\_min} = x_j$ i novi indeks minimalnog elementa $\texttt{ind\_min} = j$.

Sortiranje nizova izborom ekstrema (nastavak)

"Građevni blok": ako nađeni minimalni element nije na prvom mjestu, tj. ako je $\texttt{ind\_min} \neq i$, vrši se

- zamjena prvog elementa nesređenog dijela i minimalnog elementa,

$$x_i \leftrightarrow x_{\texttt{ind\_min}},$$

korištenjem pomoćne varijable `temp` — u tri koraka:

- $\texttt{temp} = x_i$
- $x_i = x_{\texttt{ind\_min}}$
- $x_{\texttt{ind\_min}} = \texttt{temp}$.

Sortiranje izborom ekstrema — funkcija

```
void selection_sort(int x[], int n)
{
    int i, j, ind_min, x_min, temp;

    for (i = 0; i < n - 1; ++i) {
        ind_min = i;
        x_min = x[i];
        for (j = i + 1; j < n; ++j) {
            if (x[j] < x_min) {
                ind_min = j;
                x_min = x[j];
            }
        }
```

Sortiranje izborom ekstrema — funkcija (nast.)

```
        if (i != ind_min) {
            temp = x[i];
            x[i] = x[ind_min];
            x[ind_min] = temp;
        }
    }
    return;
}
```

Zbog `x_min = x[ind_min]`, zamjena se može napraviti i bez pomoćne varijable `temp`, sa samo dvije naredbe

```
            x[ind_min] = x[i];
            x[i] = x_min;        // x_min glumi temp.
```

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

42	12	55	94	18	44	67

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

42	12	55	94	18	44	67

```
i = 0
x_min = x[0] = 42
ind_min = 0
```

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

42	12	55	94	18	44	67

`i = 0`

`j = 1`

`x_min = x[1] = 12`

`ind_min = 1`

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

42	12	55	94	18	44	67

`i = 0`

`j = 2`

`x_min < x[2] = 55`

`ind_min = 1`

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

42	12	55	94	18	44	67

`i = 0`

`j = 3`

`x_min < x[3] = 94`

`ind_min = 1`

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

42	12	55	94	18	44	67

`i = 0`

`j = 4`

`x_min < x[4] = 18`

`ind_min = 1`

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

42	12	55	94	18	44	67

`i = 0`

`j = 5`

`x_min < x[5] = 44`

`ind_min = 1`

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

42	12	55	94	18	44	67

```
i = 0

j = 6

x_min < x[6] = 67

ind_min = 1
```

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

42	12	55	94	18	44	67

```
temp = x[i]
x[i] = x[ind_min]
x[ind_min] = temp
```

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	42	55	94	18	44	67

i = 1

x_min = x[1] = 42

ind_min = 1

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	42	55	94	18	44	67

`i = 1`

`j = 2`

`x_min < x[2] = 55`

`ind_min = 1`

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	42	55	94	18	44	67

i = 1

j = 3

x_min < x[3] = 94

ind_min = 1

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	42	55	94	18	44	67

`i = 1`

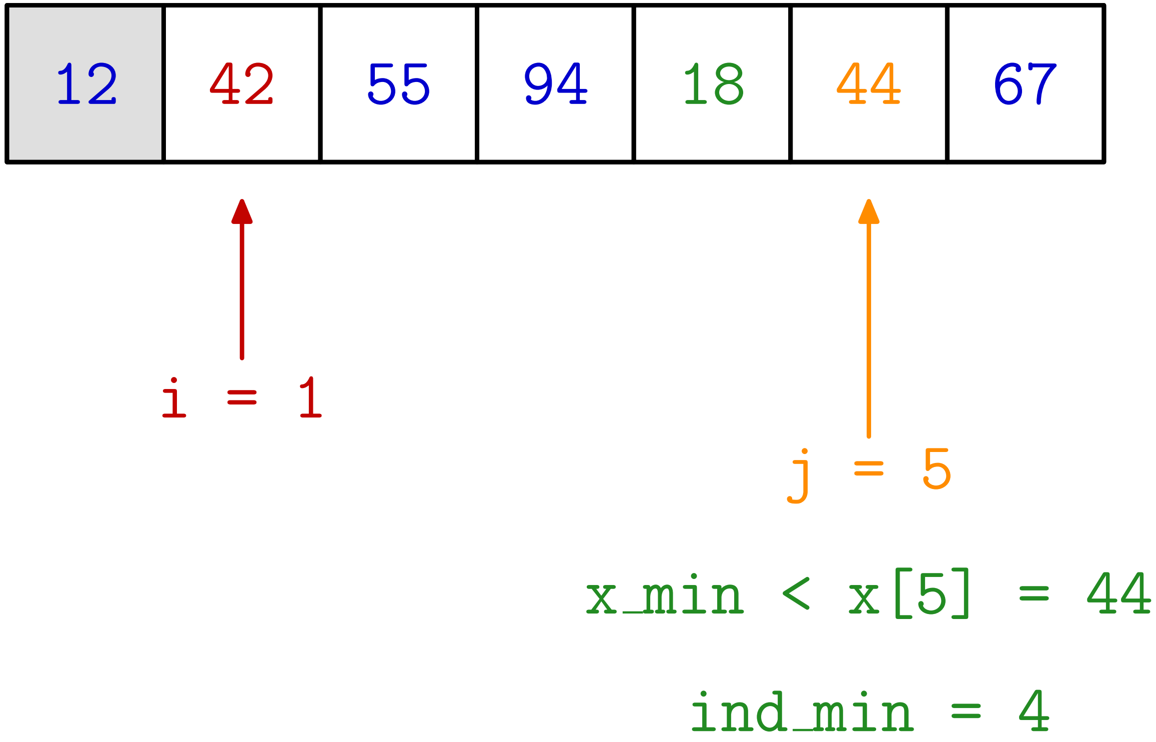
`j = 4`

`x_min = x[4] = 18`

`ind_min = 4`

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.



Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	42	55	94	18	44	67

`i = 1`

`j = 6`

`x_min < x[6] = 67`

`ind_min = 4`

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	42	55	94	18	44	67

```
temp = x[i]

x[i] = x[ind_min]

x[ind_min] = temp
```

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	18	55	94	42	44	67

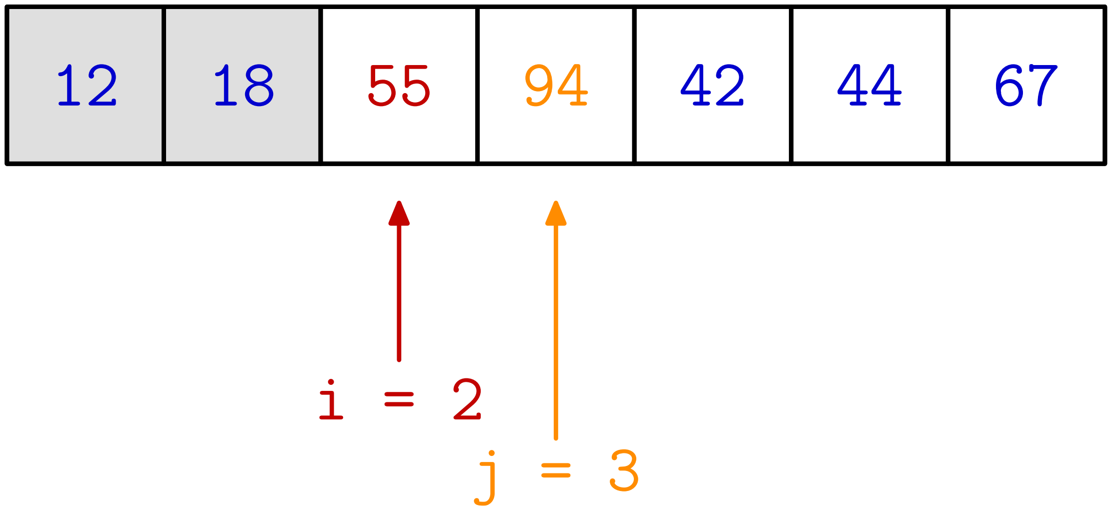
```
i = 2

x_min = x[2] = 55

ind_min = 2
```

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.



x_min < x[3] = 94

ind_min = 2

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	18	55	94	42	44	67

`i = 2`

`j = 4`

`x_min = x[4] = 42`

`ind_min = 4`

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	18	55	94	42	44	67

```
i = 2
```

```
j = 5
```

```
x_min < x[5] = 44

ind_min = 4
```

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	18	55	94	42	44	67

`i = 2`

`j = 6`

`x_min < x[6] = 67`

`ind_min = 4`

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	18	55	94	42	44	67

```
temp = x[i]

x[i] = x[ind_min]

x[ind_min] = temp
```

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	18	42	94	55	44	67

`i = 3`

`x_min = x[3] = 94`

`ind_min = 3`

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	18	42	94	55	44	67

`i = 3`

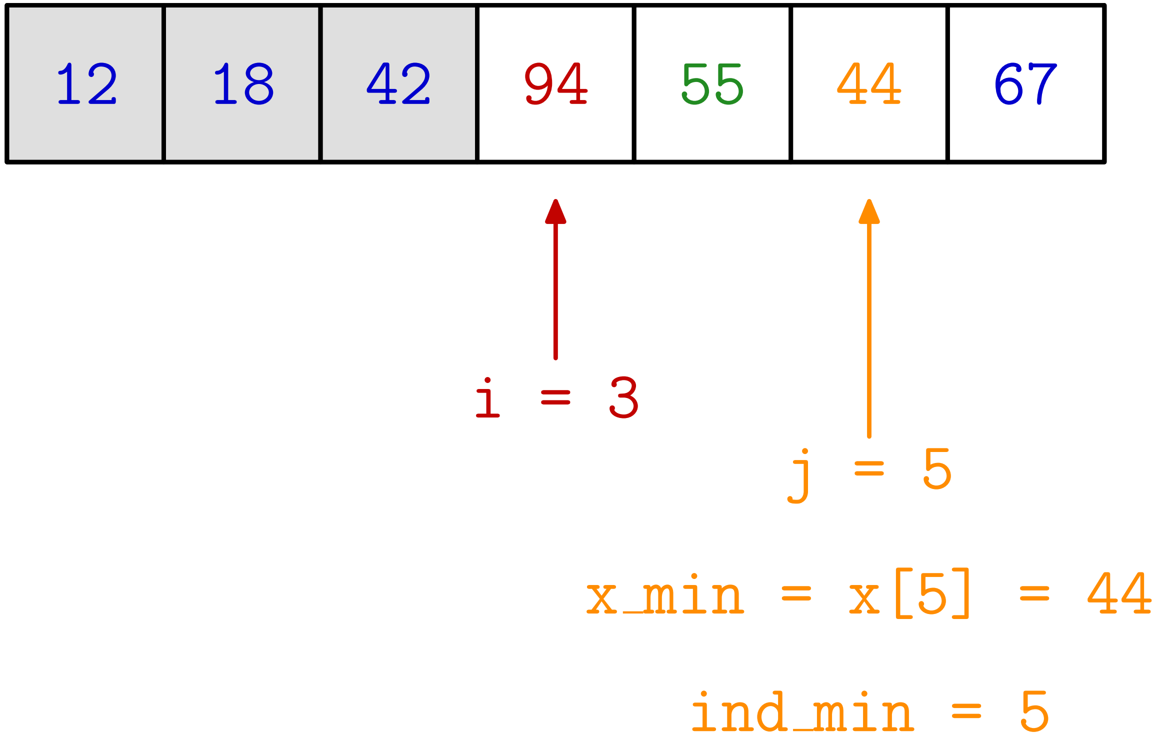
`j = 4`

`x_min = x[4] = 55`

`ind_min = 4`

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.



Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	18	42	94	55	44	67

`i = 3`

`j = 6`

`x_min < x[6] = 67`

`ind_min = 5`

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	18	42	94	55	44	67

```
temp = x[i]
x[i] = x[ind_min]
x[ind_min] = temp
```

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	18	42	44	55	94	67

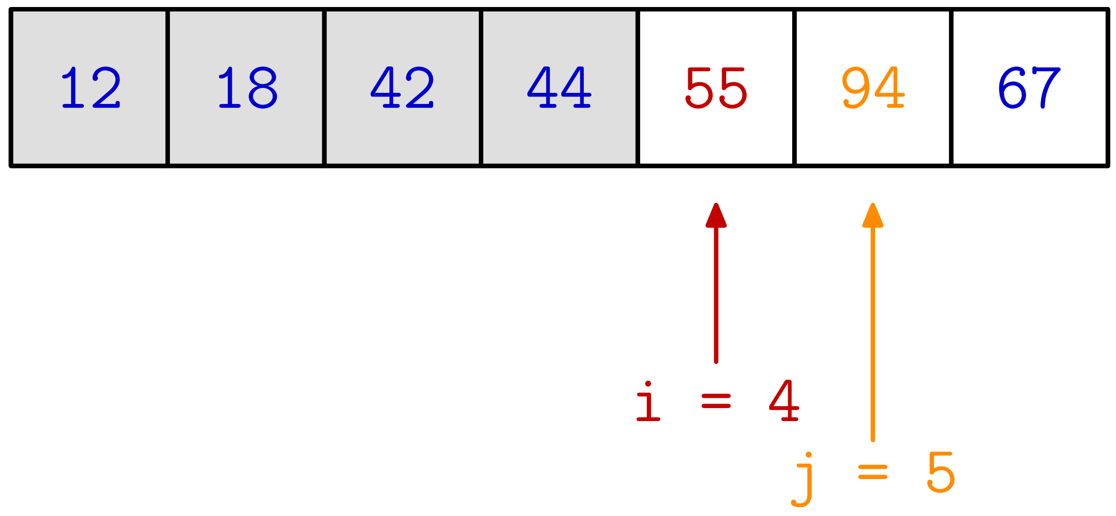
`i = 4`

`x_min = x[4] = 55`

`ind_min = 4`

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.



`x_min < x[5] = 94`

`ind_min = 4`

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	18	42	44	55	94	67

`i = 4`

`j = 6`

`x_min < x[6] = 67`

`ind_min = 4`

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	18	42	44	55	94	67

`i = 5`

`x_min = x[5] = 94`

`ind_min = 5`

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	18	42	44	55	94	67

i = 5

j = 6

x_min = x[6] = 67

ind_min = 6

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	18	42	44	55	94	67

```
temp = x[i]
x[i] = x[ind_min]
x[ind_min] = temp
```

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	18	42	44	55	67	94

i = 6

Sortiranje izborom ekstrema — primjer

Primjer. Izborom ekstrema sortirajte zadano polje.

12	18	42	44	55	67	94

Sortiranje izborom ekstrema — glavni program

```
int main(void) {
    int i, n;
    int x[] = {42, 12, 55, 94, 18, 44, 67};

    n = 7;
    selection_sort(x, n);

    printf("\n Sortirano polje x:\n");
    for (i = 0; i < n; ++i) {
        printf(" x[%d] = %d\n", i, x[i]);
    }
    return 0;
}
```

Sortiranje nizova izborom ekstrema (nastavak)

Prva varijanta — prošla funkcija:

- kod traženja ekstrema pamtimo:
  - vrijednost ekstrema (minimuma) `x_min`,
  - indeks elementa na kojem se ekstrem dostiže, `ind_min`.

Skraćena varijanta — po duljini kôda, ali ne mora biti i brža:

- očito je dovoljno pamtiti samo
  - indeks elementa na kojem se ekstrem dostiže,

  i to iskoristiti kod usporedbi, za indeksiranje članova niza.
- Trenutna vrijednost ekstrema je uvijek `x[ind_min]`.

Sortiranje nizova izborom ekstrema (nastavak)

```
void selection_sort(int x[], int n) {
    int i, j, ind_min, temp;
    for (i = 0; i < n - 1; ++i) {
        ind_min = i;
        for (j = i + 1; j < n; ++j)
            if (x[j] < x[ind_min])
                ind_min = j;
        if (i != ind_min) {
            temp = x[i];
            x[i] = x[ind_min];
            x[ind_min] = temp; }
    }
    return; }
```

Složenost sortiranja nizova

Kako ćemo uspoređivati koliko je brzo sortiranje niza raznim algoritmima?

- Možemo mjeriti vrijeme — teško i neprecizno, posebno za "kratke" nizove.
- Možemo uspoređivati broj operacija koje algoritam ili program obavlja.

Taj broj operacija je jedna od mjera složenosti algoritma.

Primijetite da kod sortiranja imamo dvije bitno različite elementarne operacije (koje ne moraju jednako trajati):

- uspoređivanje elemenata,
- zamjene elemenata.

Složenost sortiranja izborom ekstrema

Kod sortiranja izborom ekstrema vrijedi:

- broj usporedbi u svakom koraku jednak je duljini trenutnog niza umanjenoj za 1,
- jer se svaki element (osim prvog) uspoređuje s trenutno najmanjim.

Za sve korake zajedno, broj usporedbi je zbroj

$$(n-1)+(n-2)+\cdots+2+1=\frac{(n-1)\cdot n}{2}\approx\frac{n^2}{2}.$$

Dakle, broj usporedbi (sigurno) kvadratno ovisi o n.

To je loše — postoje i bolji algoritmi, s manje usporedbi.

Složenost sortiranja izborom ekstrema (nast.)

Nadalje,

- u svakom koraku vrši se najviše jedna zamjena nekog para elemenata (može je i ne biti, ako je najmanji na pravom mjestu).

Ukupan broj zamjena je najviše

$$n - 1.$$

Dakle, broj zamjena (najviše) linearno ovisi o n. To je dobro!

Zaključak: za trajanje algoritma vrijedi

$$T(n) \in O(n^2).$$

Sortiranje nizova izborom ekstrema (nastavak)

Dosad smo uvijek uzlazno sortirali

- dovođenjem najmanjeg elementa na početak.

Isti efekt imat će i

- dovođenje najvećeg na kraj nesređenog dijela niza.

Ideja:

- Dovedi najveći element niza $x_0, x_1, \ldots, x_{n-1}$ na njegovo pravo mjesto (to je zadnje u cijelom nizu).
- Postupak ponovi na skraćenom (nesređenom) nizu $x_0, \ldots, x_{n-2}$ (duljine za jedan manje, tj. $n - 1$).
- Niz se "skraćuje" straga.

Indeks i vanjske petlje sad "drži" zadnji nesređeni element, a petlja ide unatrag.

Sortiranje nizova izborom ekstrema (nastavak)

```
void selection_sort(int x[], int n) {
    int i, j, ind_max, temp;
    for (i = n - 1; i > 0; --i) {
        ind_max = i;
        for (j = 0; j < i; ++j)
            if (x[j] > x[ind_max])
                ind_max = j;
        if (i != ind_max) {
            temp = x[i];
            x[i] = x[ind_max];
            x[ind_max] = temp; }
    }
    return; }
```

Sortiranje nizova izborom ekstrema (nastavak)

Složenost — ista kao kod dovođenja najmanjeg na početak.

- Broj usporedbi je jednak

$$\frac{(n-1)\cdot n}{2} \approx \frac{n^2}{2} \in O(n^2).$$

- Broj zamjena je manji ili jednak

$$n - 1 \in O(n).$$

Napomena. Za silazno sortiranje niza treba okrenuti

- uloge ekstrema — najmanji $\leftrightarrow$ najveći, ili (ne oboje!)
- mjesto dovođenja ekstrema — početak $\leftrightarrow$ kraj, odnosno, smjer “skraćivanja” — sprijeda $\leftrightarrow$ straga.

Sortiranje zamjenama susjeda
Bubble sort

Uvod — provjera je li niz sortiran

Problem. Kako bismo provjerili je li zadani niz

$$x_0,\ x_1,\ \ldots\ ,\ x_{n-1}$$

već sortiran — na primjer, uzlazno?

Matematički rečeno, niz mora biti monotono rastući, gledano po indeksima, tj. za svaki par indeksa $j, k \in \{0, \ldots, n-1\}$ mora vrijediti

$$j < k \implies x_j \le x_k.$$

Međutim, algoritamski gledano, ne treba provjeriti sve parove indeksa — tih usporedbi bi bilo $n(n-1)/2$, što je previše.

- Dovoljno je provjeriti samo sve susjedne parove indeksa, tj. možemo uzeti $k = j + 1$. Onda mora vrijediti

$$x_j \le x_{j+1}, \quad \text{za sve} \quad j = 0, \ldots, n-2.$$

Provjera je li niz sortiran (nastavak)

Dakle, provjera "je li niz sortiran" odgovara ranijem predlošku za provjeru "svi članovi niza imaju neko svojstvo" — mora biti

$$(x_0 \leq x_1) \wedge (x_1 \leq x_2) \wedge \cdots \wedge (x_{n-2} \leq x_{n-1}).$$

Funkcija koja vraća odgovor, s najviše $n - 1$ usporedbi:

```
int sortiran(int x[], int n)
{
    int j;
    for (j = 0; j < n - 1; ++j)
        if (x[j] > x[j + 1])
            return 0;
    return 1;
}
```

Što učiniti ako niz nije sortiran?

Ako niz nije sortiran, onda postoji indeks $j \in \{0, \ldots, n-2\}$ takav da je

$$x_j > x_{j+1},$$

tj. susjedni elementi su u pogrešnom poretku. Funkcija za provjeru prekida rad, čim naiđe na prvo takvo mjesto.

Ako želimo sortirati niz, onda je prirodna ideja:

- zamijenti poredak takvih susjeda koji su u pogrešnom poretku, tj. "ispraviti" njihov poredak,
- i, zatim, nastaviti provjeru kroz cijeli niz, uz ispravak poretka svih "krivo poredanih" susjeda.

Napomena. Odmah, da svima bude jasno,

- opći niz nije moguće sortirati u jednom prolazu kroz niz!

Sortiranje zamjenama susjeda

Sortiranje zamjenama susjeda (engl. Bubble sort, bubble = mjehurić) bazira se na zamjenama susjednih elemenata u nizu.

Ideja:

- Idemo kroz niz od početka do kraja (tj. unaprijed $\longrightarrow$).
- Ako dva susjedna člana niza x_j i x_{j+1} nisu u dobrom poretku, zamijenimo im mjesto (vrijednost): $x_j \leftrightarrow x_{j+1}$.
- Kad stignemo do kraja niza (u prvom prolazu), ponovimo postupak.
- Nije jasno kada ćemo stati (jer se stalno vraćamo na početak!) — to ćemo analizirati nakon primjera.

Pratite najveći element i zamjene u svakom prolazu — ako ih nema, taj dio niza je sortiran!

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

42	12	55	94	18	44	67

`zamjena = 0`

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

42	12	55	94	18	44	67

`zamjena = 1`

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	55	94	18	44	67

`zamjena = 1`

Sortiranje zamjenama susjeda — primjer

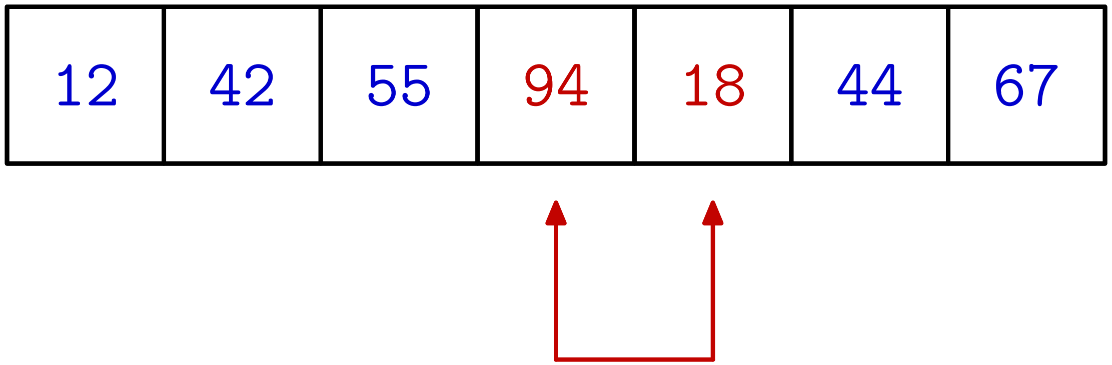
Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	55	94	18	44	67

`zamjena = 1`

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.



`zamjena = 1`

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	55	18	94	44	67

`zamjena = 1`

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	55	18	44	94	67

`zamjena = 1`

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	55	18	44	67	94

```
zamjena = 0
```

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	55	18	44	67	94

```
zamjena = 0
```

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	55	18	44	67	94

`zamjena = 0`

Sortiranje zamjenama susjeda — primjer

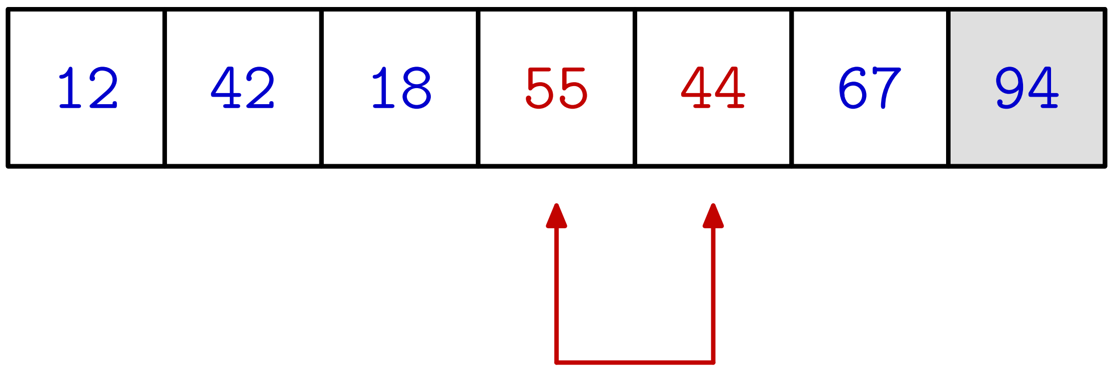
Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	55	18	44	67	94

`zamjena = 1`

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.



`zamjena = 1`

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	18	44	55	67	94

`zamjena = 1`

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	18	44	55	67	94

```
zamjena = 0
```

Sortiranje zamjenama susjeda — primjer

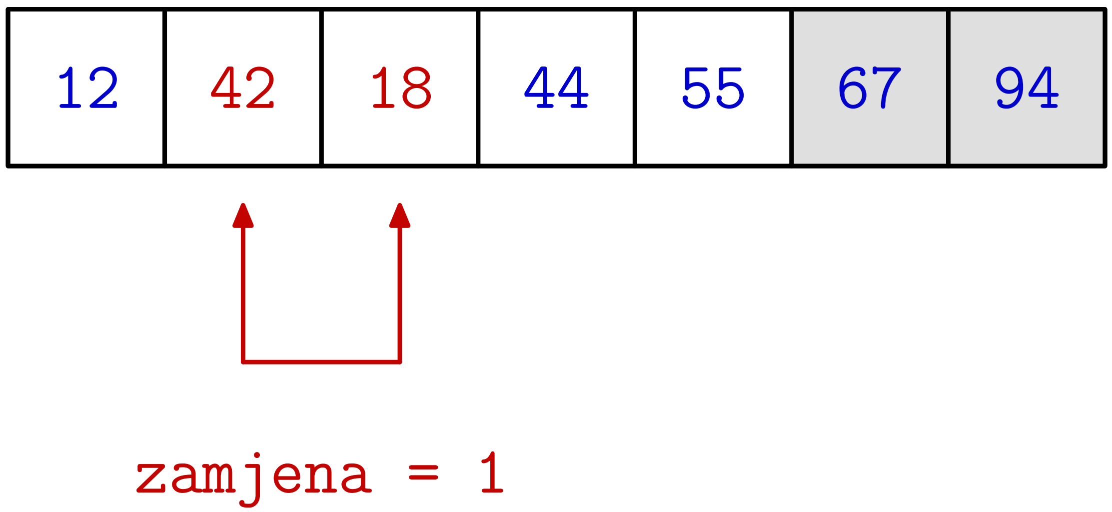
Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	18	44	55	67	94

```
zamjena = 0
```

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.



Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	18	42	44	55	67	94

`zamjena = 1`

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	18	42	44	55	67	94

`zamjena = 1`

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	18	42	44	55	67	94

```
zamjena = 0
```

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	18	42	44	55	67	94

`zamjena = 0`

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	18	42	44	55	67	94

`zamjena = 0`

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	18	42	44	55	67	94

`zamjena = 0`

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	18	42	44	55	67	94

Sortiranje zamjenama susjeda (nastavak)

Kada stajemo?

- Primijetimo da smo, u prvom prolazu, najveći element "odgurali" na kraj niza — na njegovo pravo mjesto.
- I drugi, veći elementi počeli su "putovati" prema kraju niza, međutim, ne moraju stići na svoje pravo mjesto.

Zaključak: nakon prvog koraka

- niz sigurno možemo skratiti za posljednji element x_{n-1}
- i nastaviti postupak sa skraćenim nizom $x_0, \ldots, x_{n-2}$.

Dakle, niz se skraćuje straga, isto kao kod "najveći na kraj".

- Stajemo nakon $n - 1$ prolaza, na jednočlanom nizu x_0.

Sortiranje zamjenama susjeda (nastavak)

```
void bubble_sort(int x[], int n)
{
    int i, j, temp;

    for (i = 1; i < n; ++i)
        for (j = 0; j < n - i; ++j)
            if (x[j] > x[j + 1]) {
                temp = x[j];
                x[j] = x[j + 1];
                x[j + 1] = temp;
            }
    return;
}
```

Složenost sortiranja zamjenama susjeda

Analiza složenosti algoritma:

- U prvom prolazu uspoređujemo $n - 1$ parova susjeda, u drugom $n - 2$, i tako redom . . . , do 1 u zadnjem koraku.

Dakle, ukupan broj usporedbi je, kao i kod izbora ekstrema,

$$(n-1) + (n-2) + \cdots + 2 + 1 = \frac{(n-1) \cdot n}{2} \approx \frac{n^2}{2}.$$

S druge strane, broj zamjena može drastično varirati:

- od 0 — ako je niz već sortiran, do najviše “svaka usporedba daje zamjenu” — ako je niz naopako sortiran.

Ukupan broj zamjena je najviše

$$(n-1) + (n-2) + \cdots + 2 + 1 = \frac{(n-1) \cdot n}{2} \approx \frac{n^2}{2}.$$

Složenost sortiranja zamjenama susjeda (nast.)

Složenost — sažetak:

- Broj usporedbi je jednak

$$\frac{(n-1)\cdot n}{2} \approx \frac{n^2}{2} \in O(n^2).$$

- Broj zamjena je manji ili jednak (i to je loše)

$$\frac{(n-1)\cdot n}{2} \approx \frac{n^2}{2} \in O(n^2).$$

Napomena. Promjena smjera prolaza u unatrag ($\longleftarrow$) dovodi najmanji na početak. Za silazno sortiranje niza treba samo

- okrenuti znak usporedbe — veći (`>`) $\leftrightarrow$ manji (`<`).

Sortiranje zamjenama susjeda (nastavak)

Dakle, ovakvo sortiranje zamjenama susjeda ne treba upotrebljavati, jer je loše — ima puno previše zamjena.

Može li se algoritam poboljšati?

- Možemo pamtiti koliko smo zamjena napravili u trenutnom prolazu nizom. Ako nismo napravili nijednu zamjenu — možemo stati, niz je sortiran.
  - Dovoljno je pamtiti: "ima — nema" zamjena u tom prolazu, kao u primjeru.
- Recimo, za ispravno sortiran niz, potreban je samo jedan prolaz da se ustanovi da je niz sortiran.
- U slučaju naopako (obratno) sortiranog niza, nismo uštedili ništa — nema spasa, treba svih $n - 1$ prolaza.

Sortiranje zamjenama susjeda (nastavak)

```
void bubble_sort(int x[], int n) {
    int i = n - 1, j, temp, zamjena;
    do {
        zamjena = 0;
        for (j = 0; j < i; ++j)
            if (x[j] > x[j + 1]) {
                temp = x[j];
                x[j] = x[j + 1];
                x[j + 1] = temp;
                zamjena = 1; }
        --i;    /* Smanji i za sljedeci prolaz. */
    } while (zamjena);
    return; }
```

Sortiranje zamjenama susjeda — zaključak

Zaključak: sortiranje zamjenama susjeda

- se ne isplati za opće nizove, jer ima previše zamjena.

Sjetite se da izbor ekstrema ima vrlo malo zamjena!

Možda se Bubble sort i može isplatiti za

- "skoro sortirane" nizove, ali nije lako definirati što to je,

osim baš kao "mali broj prolaza" u Bubble sortu.

Gruba ideja: to je sortirani niz, u kojeg je "slučajno" ubačen (na razna mjesta) neki mali broj novih članova.

No, za to postoji puno bolji algoritam: posebno sortirati nove članove, a onda sortirano spojiti dva sortirana niza — ova operacija se zove Merge (v. drugi semestar).

Još o sortiranju

Slični algoritmi sortiranja:

- U Bubble sortu uvijek napredujemo od početka niza.

  Ako jednom krenemo od početka, pa zatim unatrag, pa opet unaprijed, ..., dobit ćemo tzv. (na engleskom)

  - Shaker sort — doslovni prijevod: "streseni sort".

- Možemo sortirati na niz "padajućih udaljenosti": prvo daleke elemente, pa malo bliže, ..., i, na kraju, susjede. Na pr., 4 grupe (razmak 4), pa dvije (2) i susjedi (1).

  - Takav sort zove se Shell sort — nije "školjkasti sort", već je ime dobio po autoru: Donald L. Shell, 1959. g.

  Analiza složenosti mu je komplicirana. Ovisno o izboru niza udaljenosti, algoritam može biti brži od kvadratnog.

Još o sortiranju (nastavak)

Donja ograda za složenost sortiranja uspoređivanjem:

- Može se pokazati da za bilo koji algoritam sortiranja, koji koristi usporedbe parova članova (binarne relacije $<$, $>$), mora biti

$$\text{broj usporedbi} \geq c \cdot n \log n,$$

  gdje je c neka pozitivna konstanta,
- tj. broj usporedbi je reda veličine barem $n \log n$.

Dobra stvar: to je bitno brže nego n^2 usporedbi (za velike n).
Interesantno pitanje — koji algoritmi rade tako "brzo"?

Postoje i brži algoritmi sortiranja

- na pr. Radix sort ima linearnu složenost,

ali za specijalne vrste podataka!

Još o sortiranju (nastavak)

Algoritmi za sortiranje koji se koriste u praksi:

- QuickSort — autor C. A. R. (Tony) Hoare, 1962. godine.
  - Prosječna složenost mu je reda veličine $n \log n$ — za slučajne, dobro "razbacane" nizove.
  - U najgorem slučaju, složenost je reda veličine n^2.

  Koristi se zbog dobre prosječne brzine i dio je standardne C biblioteke. Radimo ga na početku Prog2.

- Heapsort — autor John W. J. Williams, 1964. godine.
  - Prosječna i najgora složenost su reda veličine $n \log n$,
  - ali je, u prosjeku, sporiji od Quicksorta.

  Detaljniji opis na SPA.

Još o sortiranju (nastavak)

U praksi se još koristi i

- MergeSort algoritam — sortiranje sortiranim spajanjem.
  - Najgora složenost je, također, reda veličine $n \log n$, tj. dostiže donju ogradu.
  - Lakše ga je napraviti na vezanoj listi, nego na polju, zbog dodatnih kopiranja kod polja.

  Radimo ga na Prog2, kad dođemo na vezane liste.

  - Autor je John von Neumann, 1945. godine.
  - To je prvi program napisan za računalo koje sprema i podatke i programe (von Neumannov model). Računalo se zvalo EDVAC.

Grubi opis QuickSorta

QuickSort se temelji na principu “podijeli, pa vladaj”.

- Uzmemo jedan element x_k iz niza i dovedemo ga na njegovo pravo mjesto u nizu.
- Lijevo od njega ostavimo elemente koji su manji ili jednaki njemu (u bilo kojem poretku).
- Desno od njega ostavimo elemente koji su veći od njega (u bilo kojem poretku).

- Ako smo dobro izabrali, tj. ako je pravo mjesto x_k blizu sredine niza, onda ćemo morati sortirati (rekurzivno)
  - dva manja niza, približno polovične duljine.

- U najgorem slučaju, ako smo izabrali “krivi” x_k, morat ćemo sortirati jedan niz duljine $n - 1$.

Sortiranje (sortiranim) umetanjem

Sortiranje umetanjem (engl. Insertion sort) bazira se na

- sortiranom ubacivanju sljedećeg elemenata x_i, u već sortirani početak niza $x_0, \ldots, x_{i-1}$, za $i = 1, \ldots, n-1$.

Sortirano ubacivanje se radi na sljedeći način:

- Kopiramo x_i u pomoćni element `temp`.
- U silaznoj petlji $j = i, \ldots, 1$, gledamo je li $x_{j-1} >$ `temp`.
- Ako je, pomaknemo x_{j-1} za jedno mjesto udesno, tj. kopiramo ga u x_j (i smanjimo j).
- Prvi put kad se dogodi da je $x_{j-1} \leq$ `temp`, stavimo $x_j =$ `temp` — to mu je pravo (sortirano) mjesto.
- Ako se to ne dogodi, onda je $j = 0$, pa opet napravimo isto, tj. stavimo $x_0 =$ `temp`.

Sortiranje umetanjem (nastavak)

```
void insertion_sort(int x[], int n) {
    int i, j, temp;

    for (i = 1; i < n; ++i) {
        temp = x[i];
        for (j = i; j >= 1 && x[j - 1] > temp; --j)
            x[j] = x[j - 1];
        x[j] = temp;
    }
    return;
}
```

Složenost: Broj usporedbi je $(n-1)n/2$, a broj pomaka je manji ili jednak od toga — no, pomaci su brži od zamjena.

Usporedba algoritama sortiranja (Intel C)

Vrijeme (u s) za sortiranje polja s $n = 10^5$ (10^6) elemenata:

Algoritam	Slučajno	Uzlazno	Silazno
`min_1`	1.612	1.612	1.611
`min_2`	2.155	2.147	2.124
`max_1`	1.362	1.361	1.362
`max_2`	2.162	2.147	2.149
`bubble_1`	11.664	2.163	5.365
`bubble_2`	11.955	0.000	5.362
`ins_1`	1.199	0.000	2.399
`qs_p2a`	0.099	2.539	2.295
`qs_std`	0.176	0.005	0.005

Ponavljanje za kolokvij (primjeri i zadaci)

Sadržaj

- Završni primjeri (ponavljanje za kolokvij):
  - Zadatak 1.
  - Zadatak 2.

Zadatak 1

Zadatak 1. Napisati funkciju kojoj su argumenti

- polje (niz) `a` prirodnih brojeva (nenegativnih) i
- cijeli broj `n`, koji zadaje broj elemenata u polju `a`.

Funkcija mora sortirati polje `a` — silazno po

- broju različitih neparnih djelitelja elemenata u polju.

U ovom primjeru, uređaj na polju `a` nije određen elementima samog polja, već je

- zadan vrijednostima funkcije f na elementima polja,
- gdje je $f(x) =$ broj različitih neparnih djelitelja od x.

Dakle, po definiciji, smatramo da je x veći ili jednak y, ako i samo ako je $f(x) \geq f(y)$.

Zadatak 1 (nastavak)

To znači da polje `a` treba sortirati (poredati) tako da vrijedi

$$f(\texttt{a[0]}) \geq f(\texttt{a[1]}) \geq \cdots \geq f(\texttt{a[n-1]}).$$

Bitni dio rješenja: kod sortiranja uspoređujemo

- vrijednosti funkcije od elemenata, tj. $f(x)$-ove,
- a ne same vrijednosti elemenata (x-ove).

Na primjer, $x > y$ prelazi u $f(x) > f(y)$.

Ako želimo brzo pronaći neki element u tako sortiranom polju,

- analogno treba postupiti i u binarnom pretraživanju, tj.
- svagdje uspoređujemo vrijednosti funkcije elemenata, a ne same elemente!

Zadatak 1 (nastavak)

Prvi korak: treba napisati funkciju `f` koja računa vrijednost $f(x)$, za zadani x (uzimamo da su x i $f(x)$ tipa `unsigned`).

Lakše i sporo rješenje (pogledati `zad_1.c`):

- modificiramo algoritam za sortiranje (ekstrem) tako da uspoređuje vrijednosti funkcije `f` na elementima polja.

Mana: puno puta računa funkciju `f` za isti element polja.

Teže i brže rješenje (pogledati `zad_1a.c`):

- prvo izračunamo sve vrijednosti funkcije i spremimo ih u novo polje `fa`, tako da je `fa[i] = f(a[i])`.
- Zatim, sortiramo polje `fa` silazno, a sve zamjene radimo "istovremeno" u oba polja: `fa` i `a`.

Jedina mana: treba nam dodatno polje `fa`.

Zadatak 2

Zadatak 2. Napisati funkciju kojoj su argumenti

- polje (niz) `a` cijelih brojeva, cijeli (nenegativni) broj `n`,
- i još dva cijela broja `x1` i `x2`.

Argumenti `a` i `n` zadaju polinom s neparnim potencijama oblika:

$$p(x) = \sum_{i=0}^{n} a_i\, x^{2i+1}.$$

Funkcija mora vratiti

- cijeli broj `x` iz intervala $[\mathtt{x1}, \mathtt{x2}]$ u kojem se dostiže najmanja vrijednost $p(\mathtt{x})$.

Ako takvih brojeva ima više, dovoljno je vratiti jednog. Ako takvih brojeva nema ($\mathtt{x1} > \mathtt{x2}$ na ulazu), treba vratiti $\mathtt{x1} - 1$.

Zadatak 2 (nastavak)

Bitni dio rješenja: računanje vrijednosti polinoma u točki x radimo modifikacijom Hornerovog algoritma

- za neparne potencije, prema formuli

$$p(x) = \sum_{i=0}^{n} a_i \, x^{2i+1} = x \cdot \sum_{i=0}^{n} a_i \, (x^2)^i = x \cdot \sum_{i=0}^{n} a_i \, y^i .$$

Dakle, stvari idu ovim redom:

- na početku, napravimo supstituciju $y = x^2$,
- napravimo Hornerov algoritam za polinom u varijabli y,
- dobivenu vrijednost, na kraju, pomnožimo s x.

Rješenje je u `zad_2.c`.