Programiranje 1
13. predavanje

Saša Singer

singer@math.hr

web.math.hr/~singer

PMF – Matematički odjel, 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

Termini “bitnih” događaja (zasad vrlo neslužbeno):

- 2. kolokvij — utorak, 18. 1. 2011., u 12 sati.
- Popravni kolokvij — utorak, 1. 2. 2011., u 12 sati.

Upisi ocjena i usmeni (po želji):

- pogledati obavijest na rezultatima kolokvija!

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.

Naknadni upis ocjene = naknadni potpis = molba, a to košta!

Informacije — nastavak

Bitno: Aplikacija za “zadaće” se

- zaključava s početkom drugog kolokvija.

Nakon toga,

- nema više novih “računa” (prijava),
- ni predaje zadataka.

U tom trenu vrijedi:

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

i nema iznimaka!

Informacije — Praktični kolokvij — termini

Praktični kolokvij (treći krug) ide odmah sljedeći tjedan,

- predvidivo: subota, 18. 12., u praktikumu 1.

Popis s terminima će biti na zidu, desno od moje sobe, početkom tjedna.

Informacije — Praktični kolokvij — općenito

Napomena: Nedolazak na bilo koji PK treba opravdati ispričnicom (nastavniku),

- inače gubite pravo na popravak!

Rok za predaju ispričnica za nedolazak na PK i sl.:

- posljednje predavanje ili konzultacije prije kraja semestra (praznika).
- Konkretno, za moju grupu: petak, 17. 12., do 14 sati.

Sve nakon toga se rješava molbama!

Pretraživanje nizova (ponavljanje)

Sadržaj

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

Pretraživanje nizova (ponavljanje)

Problem pretraživanja u općem obliku:

- 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 neki indeks i takav da je $\texttt{elt} = x_i$.

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 mjerimo brojem usporedbi

- “jednak”, odnosno, “različit”.

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.

Zapis:

$$T(n) = \mathcal{O}(n).$$

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

Točno značenje zapisa složenosti

Točno matematičko značenje zapisa

$$T(n) = \mathcal{O}(f(n))$$

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

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.

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

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

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)".

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)

Može se napraviti i varijanta sa samo jednom usporedbom u svakom raspolavljanju (probajte sami).

Zapis za trajanje:

$$T(n) = \mathcal{O}(\log n).$$

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

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.

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 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: "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 $i = 0$ do $i < n - 1$ ponavljaj:
  - U nesređenom dijelu niza (indeksi od i do $n - 1$) nađi najmanji element.
  - Najmanji element zamijeni s prvim elementom nesređenog dijela niza.
  - Ovim korakom 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.

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

Sortiranje nizova izborom ekstrema (nastavak)

“Građevni blok”: ako je `ind_min` $\neq i$ vrši se

- zamjena prvog elementa nesređenog dijela i minimalnog elementa,

korištenjem pomoćne varijable `temp`.

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

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;
}
```

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema zadano polje.

42	12	55	94	18	44	67

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema zadano polje.

42	12	55	94	18	44	67

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

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema 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. Sortirajte izborom ekstrema 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. Sortirajte izborom ekstrema 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. Sortirajte izborom ekstrema 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. Sortirajte izborom ekstrema 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. Sortirajte izborom ekstrema zadano polje.

42	12	55	94	18	44	67

`i = 0`

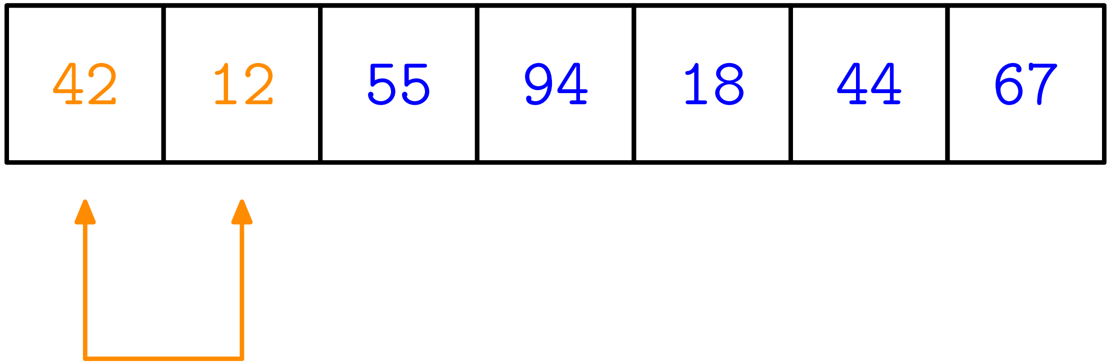
`j = 6`

`x_min < x[6] = 67`

`ind_min = 1`

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema zadano polje.




```
temp = x[i]

x[i] = x[ind_min]

x[ind_min] = temp
```

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema zadano polje.

12	42	55	94	18	44	67

i = 1

x_min = x[1] = 42

ind_min = 1

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema 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. Sortirajte izborom ekstrema zadano polje.

12	42	55	94	18	44	67

`i = 1`

`j = 3`

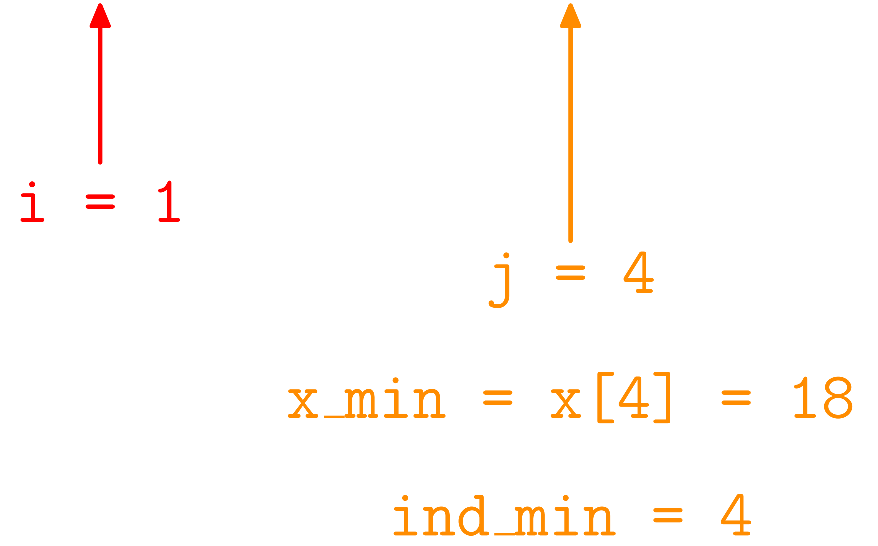
`x_min < x[3] = 94`

`ind_min = 1`

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema zadano polje.

12	42	55	94	18	44	67



Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema zadano polje.

12	42	55	94	18	44	67

`i = 1`

`j = 5`

`x_min < x[5] = 44`

`ind_min = 4`

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema 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. Sortirajte izborom ekstrema 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. Sortirajte izborom ekstrema zadano polje.

12	18	55	94	42	44	67

`i = 2`

`x_min = x[2] = 55`

`ind_min = 2`

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema zadano polje.

12	18	55	94	42	44	67

`i = 2`

`j = 3`

`x_min < x[3] = 94`

`ind_min = 2`

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema zadano polje.

12	18	55	94	42	44	67

`i = 2`

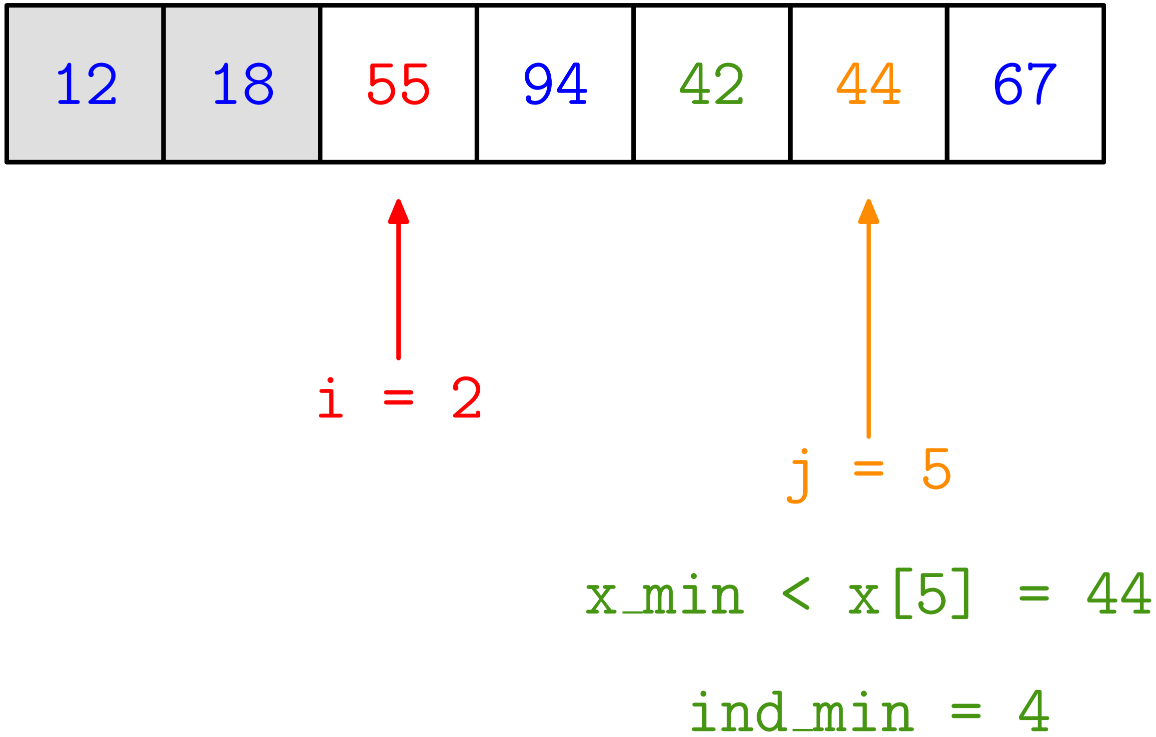
`j = 4`

`x_min = x[4] = 42`

`ind_min = 4`

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema zadano polje.



Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema zadano polje.

12	18	55	94	42	44	67

`i = 2`

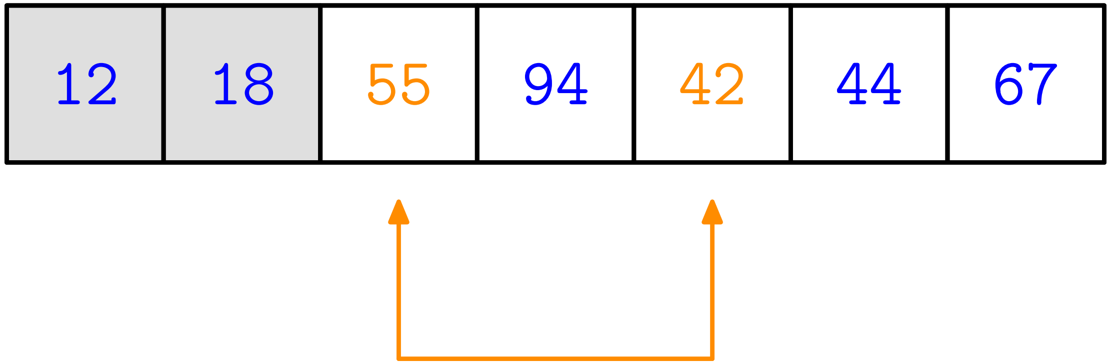
`j = 6`

`x_min < x[6] = 67`

`ind_min = 4`

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema zadano polje.




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

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema zadano polje.

12	18	42	94	55	44	67

`i = 3`

`x_min = x[3] = 94`

`ind_min = 3`

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema 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. Sortirajte izborom ekstrema zadano polje.

12	18	42	94	55	44	67

`i = 3`

`j = 5`

`x_min = x[5] = 44`

`ind_min = 5`

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema 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. Sortirajte izborom ekstrema 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. Sortirajte izborom ekstrema zadano polje.

12	18	42	44	55	94	67

`i = 4`

`x_min = x[4] = 55`

`ind_min = 4`

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema zadano polje.

12	18	42	44	55	94	67

i = 4

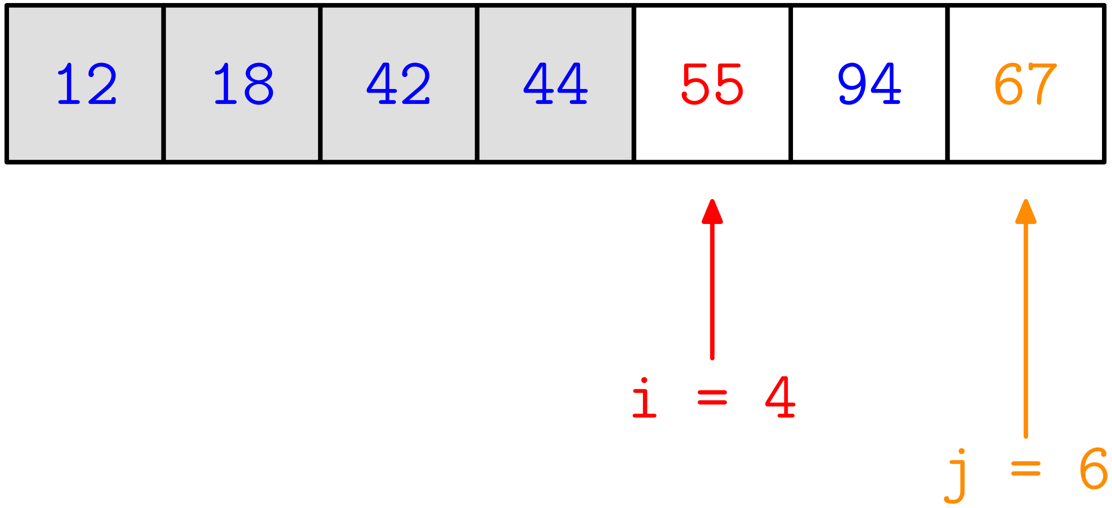
j = 5

x_min < x[5] = 94

ind_min = 4

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema zadano polje.



`x_min < x[6] = 67`

`ind_min = 4`

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema zadano polje.

12	18	42	44	55	94	67

i = 5

x_min = x[5] = 94

ind_min = 5

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema 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. Sortirajte izborom ekstrema 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. Sortirajte izborom ekstrema zadano polje.

12	18	42	44	55	67	94

i = 6

Sortiranje izborom ekstrema — primjer

Primjer. Sortirajte izborom ekstrema zadano polje.

12	18	42	44	55	67	94

Sortiranje nizova izborom ekstrema (nastavak)

Prva varijanta (prošla funkcija):

- kod traženja ekstrema pamtimo:
  - vrijednost ekstrema,
  - indeks elementa na kojem se ekstrem dostiže.

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.

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 (raznim algoritmima)?

- Možemo mjeriti vrijeme.
- Možemo uspoređivati broj operacija koje program obavlja.

Taj broj operacija je jedna od mjera složenosti algoritma.

Primijetite da kod sortiranja imamo dvije bitno različite 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 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.

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.

Zaključak: za trajanje algoritma vrijedi

$$T(n) = \mathcal{O}(n^2).$$

Sortiranje nizova izborom ekstrema (nastavak)

Dosad smo uvijek sortirali dovođenjem najmanjeg elementa na početak. Isti efekt imat će i dovođenje najvećeg na kraj.

Ideja:

- Dovedi najveći element niza $x_0, x_1, \ldots, x_{n-1}$ na njegovo 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.

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:

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

- broj zamjena je manji ili jednak:

$$n - 1 = \mathcal{O}(n).$$

Za silazno sortiranje niza treba okrenuti

- uloge ekstrema — najmanji $\leftrightarrow$ najveći, ili
- smjer "skraćivanja" — sprijeda $\leftrightarrow$ straga.

Sortiranje zamjenama susjeda
Bubble sort

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).
- Ako dva susjedna člana niza x_j i x_{j+1} nisu u dobrom poretku, zamijeni im mjesto (vrijednost).
- 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.

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.

12	42	55	94	18	44	67

`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.

12	42	18	55	44	67	94

```
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

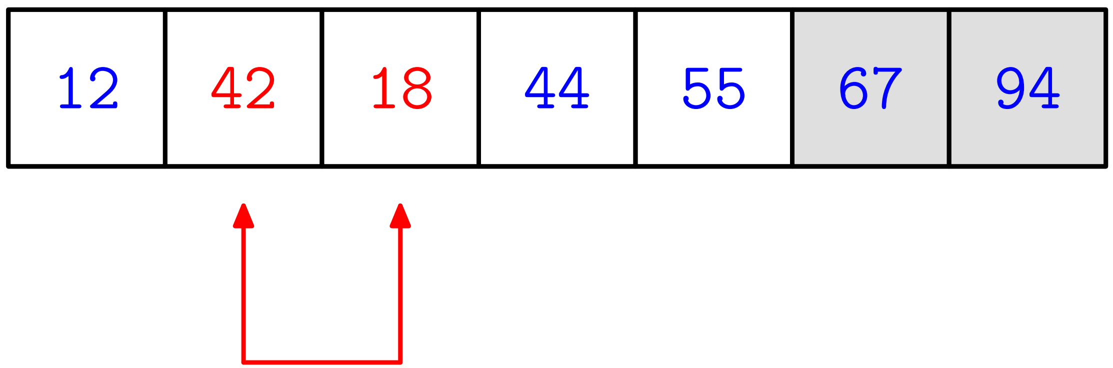
Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	18	44	55	67	94

```
zamjena = 0
```

Sortiranje zamjenama susjeda — primjer

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.



`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 = 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 (njegovo pravo mjesto).
- I drugi, veći elementi počeli su "putovati" prema kraju niza.

Zaključak: nakon prvog koraka

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

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. Dakle, ukupan broj usporedbi je

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

- Broj zamjena može drastično varirati od 0 (ako je niz već sortiran) do najviše

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

To će se dogoditi ako je niz naopako sortiran.

Složenost sortiranja zamjenama susjeda (nast.)

Složenost — sažetak:

- broj usporedbi:

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

- broj zamjena je manji ili jednak:

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

Za silazno sortiranje niza treba okrenuti

- znak usporedbe — manji $\leftrightarrow$ veći, ili
- smjer prolaza i "skraćivanja" — unaprijed $\leftrightarrow$ unatrag.

Sortiranje zamjenama susjeda (nastavak)

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

Može li se algoritam poboljšati?

- Možemo pamtiti koliko smo zamjena napravili u trenutnom prolazu nizom. Ako nismo napravili nijednu — možemo stati.
  - Dovoljno je: “ima — nema” zamjena u tom prolazu.
- 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).

Sortiranje zamjenama susjeda (nastavak)

```
void bubble_sort(int x[], int n) {
    int i, j, temp, zamjena;
    i = n - 1;
    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 (nastavak)

Zaključak:

- sortiranje zamjenama susjeda se ne isplati za opće nizove,
- možda se i može isplatiti za “skoro sortirane nizove”, ali to nije lako definirati što je.

Još o sortiranju

Slični algoritmi sortiranja:

- U Bubble sortu uvijek napredujemo od početka niza. Ako jednom krenemo od početka, pa zatim unatrag, pa opet naprijed, itd.,
  - dobit ćemo tzv. (na engl.) Shaker sort.

  (u doslovnom prijevodu "streseni sort").
- Možemo sortirati na "kontrolirane" udaljenosti, recimo, susjede, pa malo dalje članove, pa još dalje, ...
  - Takav sort zove se (engl.) Shell sort.

  To nije "školjkasti sort", već je ime dobio po autoru: Donald L. Shell, 1959. godine.

  Analiza složenosti mu je komplicirana, ali algoritam može biti brži od kvadratnog.

Još o sortiranju (nastavak)

Donja ograda za složenost uspoređivanjem:

- Za sortiranje zamjenama elemenata unutar istog niza korištenjem usporedbi članova, može se pokazati da je

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

(${c}$ je pozitivna konstanta),

- tj. broj usporedbi je reda veličine barem $n \log n$, što može biti bitno brže nego n^2 usporedbi (za velike n).

Postoje i brži algoritmi sortiranja

- na pr. Radix sort ima linearnu složenost,

ali za specijalne vrste podataka!

Još o sortiranju (nastavak)

Algoritmi 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,
  - ali u najgorem slučaju i dalje mu je složenost reda veličine n^2.

  Koristi se zbog dobre prosječne brzine i dio je standardne C biblioteke. Radimo ga na Prog2.
- Heapsort — autor John W. J. Williams, 1964. godine.
  - ima i prosječnu i najgoru složenost reda veličine $n \log n$,
  - ali je u prosjeku sporiji od Quicksorta.

  Detaljniji opis na SPA.

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 dva niza polovične duljine.
- U najgorem slučaju, ako smo izabrali "krivi" x_k, morat ćemo sortirati niz duljine $n - 1$.

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.

Bitni dio rješenja: kod sortiranja uspoređujemo

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

a ne same vrijednosti elemenata (x-ove).

Rješenje je u `zad_1.c`.

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`.

Za kraj

Najljepše želje svima

za nadolazeće blagdane!