Prošli puta smo naparavili osnovni algoritam za sortiranje niza - tzv. sortiranje traženjem ekstrema.

Za uzlazno (↗) sortiranje niza, "vanjska" petlja je išla unaprijed i dovodila jedan po jedan član na njegovo pravo mjesto.

Unaprijed ⇒ bi članovi rasli, tj. tražili smo najmanji ("ekstrem") u nesređenom dijelu i njega stavili na početak tog dijela.

Sličnu ideju možemo iskoristiti i "unatrag". Tada najveći član u nesređenom dijelu dovodimo na kraj tog dijela niza. Nesređeni dio se "skraćuje" straga.

Prvi prolaz: $x_1, -, x_n$; dovedi najvećeg na x_n; ostaje za srediti $x_1, -, x_{n-1}$.

Cijeli algoritam:

```
za i = n do 2 s korakom -1 ponavljaj
    j = i
    za k = 1 do i-1 ponavljaj
        ako je x_k > x_j onda
            j = k
    ako je i <> j onda
        temp = x_i
        x_i = x_j
        x_j = temp
```

skraćeni zapis zamjene: $x_i \leftrightarrow x_j$

Složenost je (naravno) ista kao i za prvu varijantu:

$$\text{broj usporedbi} = \frac{n(n-1)}{2} \approx \frac{n^2}{2} = O(n^2)$$

$$\text{broj zamjena} \leq n-1, \text{ tj. } = O(n).$$

Važno: zamjene radimo samo kad zaista moramo!

Sljedeći standardni algoritam za sortiranje je tzv. "BUBBLE" SORT (bubble = mjehurić).

Za razliku od prošlog algoritma, ovaj je baziran na zamjenama elemenata (članova) niza – ali; ne bilo kojih, nego uvijek susjednih.

Njega možemo nazvati "sortiranje zamjenama susjeda".

Najlakše ga je ilustrirati na "malom" primjeru za n=4

start: 3 9 -2 7 – usporedimo prva 2;
– ako su u "krivom" poretku, zamijenimo ih

(≤) (ok)
(tj. nema zamjene)

3 9 -2 7 – usporedimo sljedeći par (x_2, x_3)

(>) ⇒ zamjena

3 -2 9 7 – usporedimo sljedeći (i zadnji) par (x_3, x_4)

(>) ⇒ zamjena

imamo:

3 -2 7 9

Ovo je kraj prvog prolaza kroz niz (išli smo unaprijed!)

Algoritam za 1. prolaz: (i=1 – v. malo kasnije)

za j = 1 do n-1 ponavljaj
[ako je $x_j > x_{j+1}$ onda
[$x_j \leftrightarrow x_{j+1}$

tj. ako su x_j i x_{j+1} u krivom poretku, zamijeni ih!

Zato što startamo sprijeda (idemo unaprijed), ovim zamjenama tjeramo najveći član niza prema kraju niza. Oprez – ne samo njega!

Može se dogoditi da još ponekim velikim članovima "okrenemo" poredak i potjeramo ih prema kraju (dakle - putuju kao mjehurići - recimo u libeli, ili šampanjcu, prema kraju - bliže svom pravom mjestu).

Uočiti: na kraju 1. prolaza, sigurno je najveći član stigao na kraj niza.
(Dokaz!? - razmislite da li mogao ostati na nekom drugom mjestu!)

Dakle, smanjimo niz skratiti za 1 - i to zadnji član i ponoviti postupak na nizu $x_1, -, x_{n-1}$.

U našem primjeru - na kraju 1. prolaza imamo:

3 -2 7 | 9 → na svom mjestu.

2. prolaz - opet idemo unaprijed, par po par i okrećemo ih, ako su u krivom poretku.
Dakle:

3 -2 7 9
(>) → zamjena

-2 3 7 9
(≤) → ok.

Algoritam za 2. prolaz ($i=2$) ima petlju "kraću za 1":

za $j = 1$ do $n-2$ ponavljaj
[ako je $x_j > x_{j+1}$ onda
[$x_j \leftrightarrow x_{j+1}$;

Sada je i 7 sigurno na svom mjestu ($x_3 = x_{n-1}$), pa skratimo niz za još jedno mjesto i radimo na $x_1, -, x_{n-2}$.

Treći prolaz:

-2 3 | 7 9
(≤) → ok.

i nakon trećeg prolaza kroz niz od 2 člana (x_1, x_2) i x_2 stiže na svoje mjesto.
(3)

Onda i x_1 (-2) mora biti na svom mjestu!

Uočiti: ukupni broj prolaza je $n-1$

$i=1$ radi na $x_1, -, x_n$
$i=2$ radi na $x_1, -, x_{n-1}$
$\vdots$
$i=n-1$ radi na x_1, x_2.

} općenito $x_1, -, x_{n-i+1}$

Dakle, prva varijanta algoritma "Bubble" je:

za $i=1$ do $n-1$ ponavljaj
[za $j=1$ do $n-i$ ponavljaj
[ako je $x_j > x_{j+1}$ onda
[$[x_j \Leftrightarrow x_{j+1}$;

Napomena: j ide do $\boxed{n-i}$, tako da u zadnjem prolazu usporedi x_{n-i} i x_{n-i+1}, što je zadnji par susjeda u radnom dijelu $x_1, -, x_{n-i+1}$.

Algoritam je vrlo kratak (4 reda) i zato ga mnogi "vole", jer se lako pamti (→ nemojte "pamtiti" nego "mislite").

Međutim, algoritam je LOŠ, SPOR, NE VALJA, ... i nemojte ga koristiti u ovom kratkom obliku!

Zašto ne valja? Pogledajmo složenost.

Broj usporedbi:
$n-1$ za $i=1$
$n-2$ za $i=2$
... $n-i$ u i-tom prolazu $(x_1,x_2), (x_2,x_3), \dots, (x_{n-i}, x_{n-i+1})$
$\vdots$
1 u zadnjem prolazu!

Ukupno:

$$\text{broj usporedbi} = (n-1)+(n-2)+\dots+2+1 = \frac{(n-1)\cdot n}{2}$$

$$\approx \frac{n^2}{2} = O(n^2),$$

dakle ISTO kao i kod sortiranja traženjem ekstrema!

I što je tu loše? Tu – <u>ništa</u>.

A broj zamjena? Taj, dosta očito, može <u>drastično</u> varirati, jer <u>svaka</u> usporedba <u>može</u> (ali ne mora) rezultirati zamjenom.

Dakle, broj zamjena može (najviše) biti <u>jednak</u> broju usporedbi, pa imamo ocjenu (gornju ogradu)

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

Naravno, ostaje pitanje da li se to zaista može dogoditi.

Idemo vidjeti! To bi značilo (bar u prvom prolazu) da su <u>svaka</u> 2 susjeda u <u>krivom</u> poretku (naopaka!), tj.

$$x_j > x_{j+1}, \text{ za } j = 1, \dots, n-1$$

tj, kad urednije zapišemo (redom)

$$x_1 > x_2 > \dots > x_{n-1} > x_n.$$

A to se, naravno, MOŽE dogoditi – dobijemo na ulazu NAOPAKO SORTIRANO POLJE!

Tada: – prvi prolaz ima točno $n-1$ zamjena, i (što je još gore)
– nakon tih $n-1$ zamjena dobijemo niz

$$\underbrace{x_2 \; x_3 \; \dots \; x_{n-1} \; x_n}_{> \; > \; \dots \; >} \; | \; x_1$$

opet su SVI naopako

(zamjene su
$x_1 \leftrightarrow x_2$
$x_2 (= x_1') \leftrightarrow x_3$
$x_3 (= x_1'') \leftrightarrow x_4$
$\vdots$
$x_{n-1} (= x_1^{(n-2)}) \leftrightarrow x_n$)

$\Rightarrow$ (zaključak indukcijom!) $\Rightarrow$ u <u>svakom</u> prolazu su <u>svi</u> naopako $\Rightarrow$ svaka usporedba rezultira zamjenom.

Dakle, tzv. <u>najgori</u> slučaj za broj zamjena $= \frac{(n-1)n}{2}$ se može dosjetiti (dogoditi) i to na <u>naopako</u> sortiranom nizu

U najgorem slučaju – broj zamjena je, također, kvadratno ovisan o n, i

zato je (ovakav) bubble-sort loš!

Preciznije, može biti loš!

Ovo ↑ je dovoljan razlog da ga (u ovom obliku) ne treba koristiti u praksi.

Ima i "bolji" oblik. Do njega dolazimo tako da pogledamo kad ima malo zamjena, odnosno, kad ih uopće nema!

Da, stvarno – a kad ih nema?

Ako ih nema, onda ih NE SMIJE biti ni u prvom prolazu, a to znači da je

$$x_j \leq x_{j+1} \quad \text{za svaki } j = 1, \dots, n-1,$$

tj., raspisano:

$$x_1 \leq x_2 \leq \dots \leq x_{n-1} \leq x_n.$$

Ako je ulazni niz već sortiran, onda nema zamjena! Naravno, onda ih nema ni u jednom od sljedećih prolaza, $i = 2, \dots, n-1$, jer su i "prediji komadi" niza, također, sortirani.

I tu je prostor – ideja za uštedu, odn. ubrzanje.

Ako u nekom prolazu (za neki i) dobijemo da nema zamjena, taj dio niza je sortiran i možemo odmah prekinuti posao (jer je i ostatak IZA tog komada korektno sortiran.

Što je još zgodnije, zamjene NE treba brojati! Dovoljna nam je logička vrijednost koja kaže da li smo u danom prolazu napravili zamjenu (bar jednu) ili ne.

I sad samo malo opreza. Prvo ćemo srediti obrat za jedan prolaz (neku unutrašnjost od i).

Na početku prolaza treba postaviti

zamjena = laž

(jer u tom prolazu još zasad NISMO napravili niti jednu zamjenu).

Zatim, u petlji po j za taj prolaz, ako naiđemo par susjeda u krivom poretku, tj.

ako je $x_j > x_{j+1}$

onda ih "okrećemo" $x_j \leftrightarrow x_{j+1}$ i postavljamo "zastavicu" (engl. "flag") zamjena = istina

(jer smo upravo napravili zamjenu).

Time smo "sredili" svaki pojedini prolaz. Ostaje nam još "dovesti u red" vanjsku petlju koja "vrti" prolaze.

Ono što hoćemo = prekinuti prolaze u trenu kad (nakon prolaza!) ostane
zamjena = laž.

Obratno rečeno — u sljedeći prolaz idemo ako je
zamjena = istina.

Naravno, još moramo korektno postaviti broj prolaza i.

Na početku je (recimo) $i = 1$ ("broj" sljedećeg prolaza, tako da petlja po j i dalje ide od 1 do $n-i$)

U prolaz idemo ako je $i \leq n-1$, a na kraju prolaza povećamo i za 1.

Još samo "spojimo" uvjete za i i zamjenu:

sljedeći prolaz $\Leftrightarrow$ ($i \leq n-1$) and zamjena = istina

mogu izbaciti

Zadnja stvar – treba korektno postaviti vrijednost logičke varijable zamjena na početku, kad stavljamo i $i = 1$.

Malo blesavo, ali treba staviti

zamjena = istina

jer želimo (moramo) ući u prvi prolaz, koji će onda postaviti zamjena na ono što treba.

Algoritam "Bubble 2":

```
zamjena = istina
i = 1
sve dok je (i ≤ n-1) i (zamjena = istina) ponavljaj
    zamjena = laz
    za j = 1 do n-i ponavljaj
        ako je x_j > x_{j+1} onda
            x_j ↔ x_{j+1}
            zamjena = istina
    i = i+1
```

Algoritam više nije tako jednostavan!
U najgorem slučaju – složenost ostaje ista (kvadratni broj usporedbi i zamjena)

Dakle, za *sortiranje "bilo kakvog" (općeg ili slučajnog) niza – i dalje se NE ISPLATI, obzirom na sortiranje traženjem ekstrema.

Ukratko – "ne služi ničemu!"

Nije baš sasvim točno. Ako znamo unaprijed da će završiti u malom broju prolaza, onda ima smisla. Tj. za sasvim posebne nizove, može biti i dobar. A koji su to nizovi?

Odgovor nije sasvim jednostavan!
Ugrubo – oni koji su "skoro" sortirani (nije baš precizno)
(malim brojem zamjena susjeda postaju sortirani).

– Uputa – bubble sort ima smisla za drugačije oblike spremanja nizova (ne u polje!)
Recimo – za strukture u kojima su zamjene susjeda "brze", a zamjene "udaljenih" članova "spore" (– nešto poput traka – sekvencijalnih medija). [ne mijedi – RAM, tj. pristup svakom članu NE TRAJE PODJEDNAKO]

– Varijante:
– "Shaker"-sort: umjesto svaki puta "unaprijed", prolazi alterniraju po smjeru
unaprijed, unatrag
(kao miješanje – "trešnja" koktela u Shakeru – gore/dolje)

– "sortiranje na "kontrolirane" udaljenosti susjedi, pa malo dalji, pa još dalji... članovi.
To se zove Shell-sort (v. Knuth 3. dio)
i analiza složenosti je kompliciranija, ali alg. može biti i brži od kvadratnog.

– Napomena: za sortiranje zamjenama elemenata (članova) unutar istog niza (bez dodatne memorije), i to na osnovu USPOREDBI članova
vrijedi:
broj usporedbi $\geq c \cdot n \cdot \log n$
tj. broj usporedbi je reda veličine barem $n \cdot \log n$
(što, ako dostignemo, je bitno BRŽE od n^2, recimo za $n = 10^6$ ili 10^9).

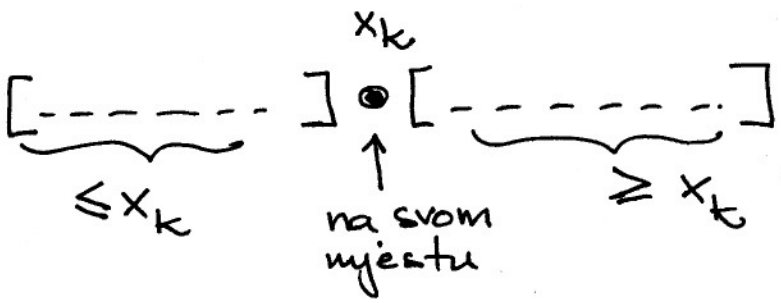
Algoritmi za sortiranje nizova koji se stvarno koriste u praksi:

Quicksort – prosječna složenost $O(n \log n)$ za "slučajne" – dobro razbacane (neuređene) nizove
– najgori slučaj – i dalje kvadratno u n
– zbog dobre prosječne brzine – najčešće se koristi – standardni dio C-biblioteke.

Heapsort – $O(n \log n)$ i u najgorem slučaju (v. SPA)
– u prosjeku je SPORIJI od Quicksorta.

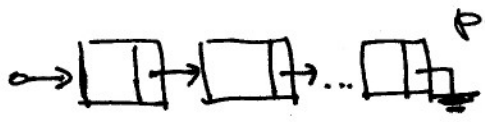
Podloga za Quicksort – "podijeli pa vladaj" (divide and conquer)

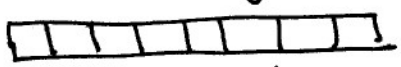
– dovedi neki član na njegovo pravo mjesto
– ispred njega su manji od njega (ali NE sortirano)
– iza njega su veći od njega (opet NE sortirano)
⇒ sortiraj ta 2 manja polja (smanjenje veličine NE za 1, već, IDEALNO na POLA)



– Postoje i algoritmi za sortiranje kod kojih je:
– dozvoljena dodatna memorija ili su specijalno prilagođeni drugačijim strukturama podataka za "pamćenje nizova"

na pr. – mergesort (sekvencijalne strukture – vezane liste, datoteke → ureduj pointerima





traka – nejednak pristup

Mergesort - malo kasnije! (dozvoljeno kopiranje na poljima!)

- Postoje i sortovi koji nisu bazirani na relacijskom uspoređivanju podataka ($\leq$ za 2 objekta, s odgovorom - istina/laž), već se koristi "proširenu" uređaj "podobjekata" (tj. nešto poput leksikografskog uređaja za: stringove = nizove znakova ... a<b<.... brojeve = nizove znamenki 0<1...)

Takvi sortovi ("radix-sort") mogu biti i brži od klasičnih - čak linearno u n, tj. $O(n)$.

— . —

Uvod u merge-sort = merge (bez sorta)

Merge = spoji i to uredeno.

Što to znači?

Zadana su 2 sortirana niza:

$$a_1 \leq a_2 \leq \ldots \leq a_m \quad (m \text{ članova})$$

i

$$b_1 \leq b_2 \leq \ldots \leq b_n \quad (n \text{ članova})$$

Ta 2 niza treba "spojiti" u treći niz, duljine $m+n$, koji sadrži točno sve članove zadana 2 niza, i još je sam opet sortiran

$$c_1 \leq c_2 \leq \ldots \leq c_{m+n}.$$

(Da ne kompliciramo s oznakama, što znači da c sadrži članove od a i b, zamislimo skupovno

" $\{c_1, -, c_{m+n}\} = \{a_1, -, a_m\} \cup \{b_1, -, b_n\}$ "

uz odgovarajuće brojanje elemenata!)

Prvo - motivacija - tj. koja je korist od takvog algoritma za spajanje?

Zamislimo da želimo napraviti algoritam sortiranja po principu "podijeli pa vladaj". To znači nešto otprilike:

- podijeli niz u (recimo) 2 podniza (recimo, podjednakih duljina)
- sortiraj svakog od njih (kako znaš i umiješ)
- i sad dođe "merge":
  spoji natrag ta 2 sortirana podniza u jedan sortirani niz.

Dakle, idemo prvo napraviti ovaj zadnji dio algoritma.