"Merge" = spajanje ("uređena unija") 2 sortirana niza u jedan sortirani niz

$$\left.\begin{matrix} a_1 \le a_2 \le \ldots \le a_m \\ b_1 \le b_2 \le \ldots \le b_n \end{matrix}\right\} \rightarrow c_1 \le c_2 \le \ldots \le c_{m+n}$$

Primjer:

niz (a): 1 3 4 8 10
niz (b): 2 5 6

– usporedim 2 prva = najmanja člana;
– manji od njih je najmanji u nizu (c)

$1 < 2 \Rightarrow c_1 = 1$

zbog $c_1 = a_1$, član a_1 smo "potrošili" ("prebacili" u niz (c)), pa gledamo sljedeći član u (a).
Tj. tamo gdje "potrošim" član – idem na sljedeći član (ako ga ima)

niz (a): 1 3 4 8 10 niz (c):
niz (b): 2 5 6 1 2

niz (a): 1 3 4 8 10 1 2 3
niz (b): 2 5 6

niz (a): 1 3 4 8 10 1 2 3 4
niz (b): 2 5 6

niz (a): 1 3 4 8 10 1 2 3 4 5
niz (b): 2 5 6

niz (a): 1 3 4 8 10 1 2 3 4 5 6
niz (b): 2 5 6

niz (a): 1 3 4 $\underline{8 \quad 10}$ 1 2 3 4 5 6 8 10
niz (b): $\underbrace{2 \quad 5 \quad 6}$

sve "prebacili" $\Rightarrow$ prebaci ostatak od (a) NA KRAJ u (c).

Algoritam:

ap = 1 } indeksi ("pokazivači") na prva sljedeća
bp = 1 } "nepotrošena" mjesta u (a), odn. (b)
cp = 1 } indeks prvog sljedećeg slobodnog mjesta u (c).

sve dok je (ap ≤ m) i (bp ≤ n) ponavljaj
{tj. sve dok ne stignemo do kraja jednog od ta 2 niza}
- ako je a[ap] ≤ b[bp] onda
  - c[cp] = a[ap] } iz (a) u (c)
  - ap = ap + 1 } i "pomak" u (a)
- inače
  - c[cp] = b[bp] } iz (b) u (c)
  - bp = bp + 1 } i "pomak" u (b)
- cp = cp + 1 ← pomak u (c)

sve dok je (ap ≤ m) ponavljaj } kopiraj "kraj" od (a) u (c)
- c[cp] = a[ap]
- ap = ap + 1
- cp = cp + 1

sve dok je (bp ≤ n) ponavljaj } kopiraj "kraj" od (b) u (c).
- c[cp] = b[bp]
- bp = bp + 1
- cp = cp + 1

Uočiti: zbog testa u prvoj petlji (ap≤m) i (bp≤n) izvršava se (najviše) jedna od zadnje dvije petlje.

Nema puno smisla pitati "kako" je stala prva petlja, jer je test ugrađen u drugu i treću:

ako je ap ≤ m onda ~~prva petlja~~
  prva petlja ← pita (ap ≤ m) na početku!
inače
  druga petlja

U jeziku C ovaj algoritam ima izrazito zgodan i kratak zapis, ako koristimo "inkrement" operaciju

$$xp{+}{+}$$

$\Longleftrightarrow$ iskoristi xp i povećaj ga za 1.

Blok od 3 naredbe dolika (na pr.)

$$\left.\begin{array}{l} c[cp] = a[ap] \\ ap = ap + 1 \\ cp = cp + 1 \end{array}\right\} \xrightarrow[\text{kratko u C-u}]{} \boxed{c[cp{+}{+}] = a[ap{+}{+}]}.$$

<u>Napomena</u>: Kad su podaci spremljeni u strukturi <u>polja</u> (pristup elementima preko indeksa), onda je merge nizova (a) i (b) puno lakše napraviti <u>kopiranjem</u> u novi niz (c).

Pokušajte napraviti cijelu stvar bez kopiranja, koristeći samo zamjene elemenata, s tim da je na početku:

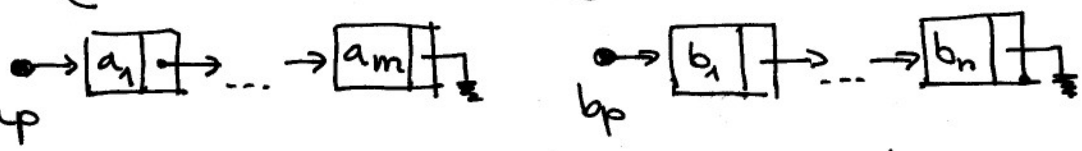
$$a_1 \; \ldots\ldots \; a_m \quad b_1 \; \ldots\ldots \; b_n$$

recimo u polju c:

$$\underbrace{c[1] \; \ldots \; c[m]}_{\text{uređen } \nearrow} \quad \underbrace{c[m+1] \; \ldots \; c[m+n]}_{\text{uređen } \nearrow}$$

i zamjenama treba dobiti da je cijeli niz uređen $\nearrow$.

– S druge strane, ako su podaci iz nizova (a) i (b) spremljeni (zadani) kao (dvije) vezane liste



onda je merge puno elegantniji – <u>nema kopiranja</u>, samo pažljivo mijenjamo veze (pointere!).

Sad se možemo vratiti na "merge-sort". Ideja za sortiranje niza

$$x_1, x_2, \ldots, x_n$$

na principu "podijeli pa vladaj" je:

1. – naoti indeks "srednjeg" člana
$$k = \lfloor n/2 \rfloor = n \text{ div } 2$$
i "podijeli" niz na 2 podjednako duga dijela
$$\underbrace{x_1, \ldots, x_k}_{1.\text{ dio}}, \quad \underbrace{x_{k+1}, \ldots, x_n}_{2.\text{ dio}}$$
(slično binarnom pretraživanju!)
[Usput: ISPLATI se uzeti <u>podjednako</u> duge dijelove, tj. $k = \lfloor n/2 \rfloor$.
Probajte to <u>dokazati</u> – na kraju, kad napravimo cijeli algoritam!]

2. – sortiraj $x_1, \ldots, x_k$
– sortiraj $x_{k+1}, \ldots, x_n$

3. – spoji (sortirane) nizove $\underbrace{x_1, \ldots, x_k}_{(a)}$ i $\underbrace{x_{k+1}, \ldots, x_n}_{(b)}$
Ovo je operacija merge: (a) i (b)
Za nju nam treba <u>dodatni prostor</u> za kopiranje, pa kod realizacije treba malo paziti, jer želimo da sortirani cijeli – spojeni niz "završi" na ISTIM mjestima
$$x_1, \ldots, x_n.$$

No, prije realizacije 3. koraka, treba odlučiti kako ćemo napraviti <u>2. korak</u> – tj. sortirati ona 2 kraća niza.

Prirodno: iskoristiti ISTI ovaj algoritam, samo na raznim podnizovima – tj. na raznim "intervalima" indeksa

$$1..n \longrightarrow 1..k,\ k+1..n$$

Taj postupak se zove REKURZIJA (ili rekurzivni algoritam):

- za rješenje "velikog" problema koristimo ISTI algoritam na "manjim" problemima – sve dok se problem ne svede na trivijalno mali problem kojeg znamo direktno riješiti!

⇒ ovo "pozivanje samog sebe" ("jedenje za rep")
- ide na sve manjim potproblemima
- MORA BITI KONAČNO – tj. negdje MORA STATI.

Prva stvar koju moramo riješiti – kako STATI, tj. kako prekinuti rekurziju!

Dakle, što je – u našem slučaju – onaj "trivijalno mali" problem, kojeg znamo direktno riješiti?

Na pr., niz od 2 člana je lako sortirati – direktno! Očito, treba nam $1 \times$ usporedba i najviše jedna zamjena članova.

Međutim, idemo "do kraja" – sjetite se da je niz od <u>jednog jedinog</u> člana uvijek <u>sortiran</u>!

Tj. za niz od jednog člana NEMAMO ŠTO RADITI!

(Prijevod: stvar zaista postaje trivijalna!
Toliko trivijalna da uopće nema posla.)

To znači da ona 3 koraka: (1) raspolovi ($k = \lfloor n/2 \rfloor$)
(2) sortiraj podnizove
(3) spoji

radimo samo ako je duljina niza barem 2.

To bi za cijeli niz značilo $n \geq 2$, ali polako!
Isti kriterij treba propisno ugraditi i za podnizove, a to još NISMO napravili.

Da bismo dobili <u>korektan</u> rekurzivni algoritam, čitav posao moramo <u>korektno "parametrizirati"</u>.

Nije dovoljno gledati tzv. vanjski poziv algoritma koji kaže:

"sortiraj niz $x_1, \ldots, x_n$, duljine n".

Treba vidjeti što algoritam radi "negdje duboko u sredini posla", i kako to ZGODNO ZAPISATI.

Kad radimo na podnizovima, onda:

- duljine (naravno) padaju,
- tzv. RADNI DIO niza <u>nije</u> smješten na početku (tj. ne mora biti $x_1, \ldots, x_k$).
- analogno, ne mora biti ni na kraju ($x_{k+1}, \ldots, x_n$)
- ove dvije mogućnosti dobijemo na najvišem nivou rekurzije, kad "napadamo" čitav niz.

Negdje "u pola posla" naprosto radimo na <u>NEKOM</u> komadu niza, koji NIJE RAZBACAN, nego se proteže preko BLOKA UZASTOPNIH INDEKSA

$$x_\ell, \ldots, x_m.$$

[Dovoljno je znati početni indeks ℓ i krajnji indeks m, pa da znamo sve što treba.

Duljina tog podniza je $m - \ell + 1$, ali nam duljina UOPĆE NEĆE TREBATI, jer:

$$\text{duljina} \geq 2 \iff m - \ell + 1 \geq 2$$

ili $\quad m \geq \ell + 1.$

No, m i ℓ su cijeli brojevi, pa je

$$\ell + 1 \leq m \iff \boxed{\ell < m}$$

(što je odmah očito iz skice podniza!)

Dakle, negdje "u pola posla", naš algoritam mora sortirati podniz

$$x_l, \ldots, x_m$$

a taj je korektno određen indeksima $\underline{l, m}$.
I to je tražena "parametrizacija" za rekurziju!
(Opet, sjetite se binarnog traženja. Tamo, također, pamtimo lijevi i desni indeks, kao i ovdje.
Tj. tko želi, može napisati binarno pretraživanje kao rekurzivni algoritam)

Naravno, ostaje još pitanje:
- kako se ovako parametrizirani algoritmi zapisuju u standardnim programskim jezicima?

Odgovor: jednostavno – kao <u>potprogrami</u>
i to <u>s parametrima</u>.

Općenito – imamo 2 vrste potprograma:
- procedure (subroutine)
- funkcije (u C-u – sve su funkcije)

Zajedničko: algoritam s imenom i parametrima
Razlika : funkcija uvijek vraća neku vrijednost (nekog tipa).

<u>Primjer</u>: funkcija za sinus ima oblik:

<u>zaglavlje</u>: $\underbrace{\text{(tip vrijednosti)}}_{\text{recimo, double}}\ \underbrace{\text{sin}}_{\text{ime funkcije}}\ (\underbrace{\text{popis parametara}}_{\text{double x}})$

i onda idu naredbe koje za zadanu vrijednost x (to je LOKALNA VARIJABLA u tijelu funkcije) računaju vrijednost koju treba.

Za vraćanje vrijednosti i IZLAZ iz funkcije postoji naredba

$$\text{return } \underbrace{\ldots}_{\text{vrijednost}}$$

Poziv funkcije: najčešće unutar izraza u sklopu neke naredbe:

$$y = 5 * \sin(\underline{2.73})$$

ovdje zadajem vrijednost parametra za koju želim vrijednost funkcije.

- Onaj "x" iz zaglavlja funkcije se obično zove FORMALNI PARAMETAR (ili ARGUMENT).
  Formalni – zato što će STVARNU VRIJEDNOST dobiti tek kod poziva funkcije.
- Onaj "2.73" je STVARNI PARAMETAR (argument).
- Prijenos parametra 2.73 u potprogram za sin odgovara tome da se lokalnoj varijabli $\underline{x}$ dodijeli vrijednost 2.73, tj.

$$x = 2.73$$

i onda se izvršavaju naredbe funkcije (odn. potprograma), do naredbe za vraćanje.
- Pri povratku se cijeli poziv funkcije

$$\sin(2.73)$$

zamjenjuje onom IZRAČUNATOM VRIJEDNOŠĆU (iz naredbe return) i dalje se računa

$$y = 5 * \text{ta vrijednost}$$

- Potpuno isti princip vrijedi i za potprograme koji nisu funkcije – postoji posebna naredba za poziv potprograma!

- A gdje je spremljen onaj x iz potprograma!?

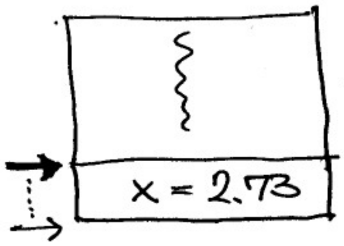
Njega nema, sve dok ne pozovem potprogram (jer je LOKALNI OBJEKT).

Pri pozivu (ulazu) – "oživi", tako da DINAMIČKI dobije odgovarajući prostor (memoriju) i ADRESU u posebnom bloku memorije za takve lokalne objekte u potprogramima.

NAZIV: (runtime) STACK (stog).

Na taj stog se "SLAŽU" lokalni objekti pri ULAZU u potprogram. Tamo žive, dok radi potprogram. Pri IZLAZU se BRIŠU

ULAZ – oživi x i spremi vrijednost



IZLAZ – briše taj x sa stoga.

Naravno, uvijek se pamti prva slobodna lokacija na stogu (tzv. STACK POINTER), koji šeta "gore – dolje".

- Ovaj mehanizam uredno realizira prebacivanje vrijednosti U potprogram (tzv. ULAZNI parametri).
- Međutim, moguće je napraviti i obratno, tj. kroz parametar VRATITI neku vrijednost IZ potprograma (tzv. IZLAZNI parametar).

(To je različito od vraćanja vrijednosti funkcije!)

Kako se to radi?

- Prvo, kad se vratim iz potprograma, ta IZLAZNA vrijednost će se negdje spremiti i ta adresa (prostor) mora biti rezerviran prije ULASKA.

U C-u: moram (kao stvarni parametar) u pozivu zadati POINTER (pokazivač), tj. poslati upravo tu adresu na koju će se spremiti izlazna vrijednost.

[Tj. na STACK se sprema ta adresa]
(Ovo je mehanizam prijenosa parametara PO VRIJEDNOSTI — lokalni objekti UVIJEK dobiju VRIJEDNOST IZVANA, pa makar to bila i ADRESA).

U Pascalu i Fortranu mogu napisati i IME varijable, samo moram navesti da je to IZLAZNI parametar. Tada prevodilac sam realizira prijenos adrese u potprogram. (Tj. mehanizam je zapravo ISTI, samo je ZAPIS drugačiji).

—·—

Vratimo se mergesortu. Naš mergesort je potprogram s parametrima koji zadaju onaj komad niza kojeg treba sortirati:

$$x_\ell, \ldots, x_m.$$

Moram zadati: koji niz (x — prenosi se adresa prvog elementa!)

i indekse ℓ, m.

Zapis: $\text{mergesort}(\underset{\substack{\uparrow\\ \text{adresa}}}{x}, \underbrace{\ell, m}_{\text{indeksi}})$

<u>Algoritam</u> mergesort (x, ℓ, m): {sortira $x_\ell, \ldots, x_m$}
ako je $\ell < m$ onda
$k = \lfloor (\ell+m)/2 \rfloor$
mergesort (x, ℓ, k) {sort: $x_\ell, \ldots, x_k$}
mergesort $(x, k+1, m)$ {sort: $x_{k+1}, \ldots, x_m$}
spoji (x, ℓ, k, m)

Potprogram spoji (x, ℓ, k, m) treba spojiti sortirane nizove

$$x_\ell, \ldots, x_k \qquad x_{k+1}, \ldots, x_m$$

u sortirani niz $x_\ell, \ldots, x_m$.

Realizacija je kao ranije:

$$a_1, \ldots, a_m \iff x_\ell, \ldots, x_k$$
$$b_1, \ldots, b_n \iff x_{k+1}, \ldots, x_m$$

s tim da kod spajanja kopiramo u <u>pomoćni niz</u>

$$c_1, \ldots, c_{m-\ell+1}$$

a onda to vratimo natrag u $x_\ell, \ldots, x_m$.

<u>Algoritam</u> spoji (x, l, k, m):

```
ap = l
bp = k+1
cp = 1
sve dok je (ap ≤ k) i (bp ≤ m) ponavljaj
  [ako je x[ap] ≤ x[bp] onda
  |    [c[cp] = x[ap]
  |    [ap = ap + 1
  |inače
  |    [c[cp] = x[bp]
  |    [bp = bp + 1
  [cp = cp + 1
sve dok je (ap ≤ k) ponavljaj
  [c[cp] = x[ap]
  |ap = ap + 1
  [cp = cp + 1
sve dok je (bp ≤ m) ponavljaj
  [c[cp] = x[bp]
  |bp = bp + 1
  [cp = cp + 1          ←  cp uredno drži
                           zadnji indeks
                           u c!
                           (znamo
                           cp = m-l+1)
za i = 1 do cp ponavljaj
  [x[l+i-1] = c[i]
```

Binarno pretraživanje:

$x_1 \le x_2 \le \dots \le x_n$ i pitam da li je a u nizu

nasao := laz

$d \leftarrow 1$; $g \leftarrow n$; i nisam nasao:

sve dok je $d \le g$ ponavljaj

  $i \leftarrow \lfloor (d+g)/2 \rfloor$;

  ako je $a = x_i$ onda

    našli smo ga, vrati indeks i, izađi iz petlje

  inače ako je $a < x_i$ onda $g \leftarrow i$

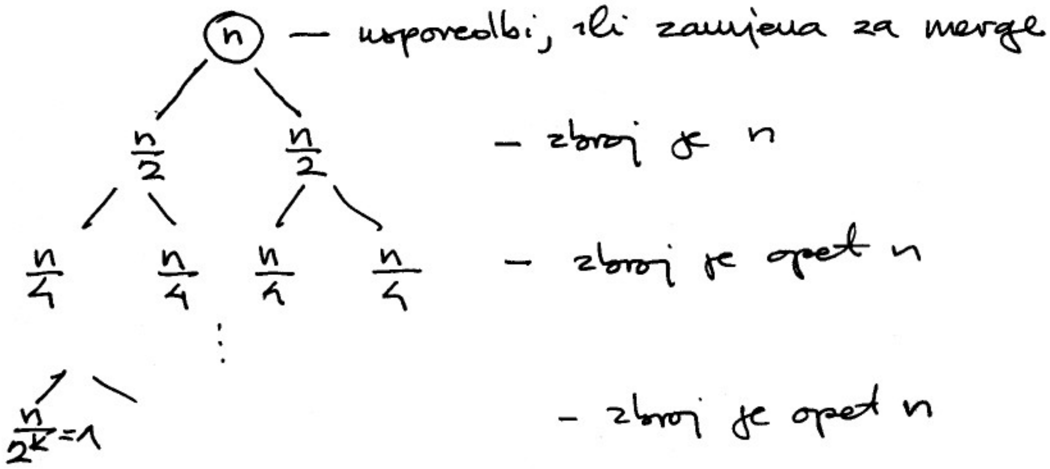
    inače $d \leftarrow i$

slož: $n \to \frac{n}{2} \to \frac{n}{4} \to \dots \to \frac{n}{2^k} = 1 \Rightarrow k = \log_2 n$

Mergesort: sortiraj 2 polovine $\underbrace{x_1 - x_{\lfloor \frac{n}{2} \rfloor}}, \underbrace{x_{\lfloor \frac{n}{2} \rfloor + 1}, -, x_n}$

merge = spoji - sortirano:

Vrijeme: rastav po bin. stablu:



i imam okruglo $k = \log_2 n$ nivoa, na svakom je zbroj $= n$

$\Rightarrow$ $n \cdot \log_2 n$ operacija (usp/zamjena).

Usporedba n^2 i $n \log_2 n$

omjer $\frac{n^2}{n \log_2 n} = \frac{n}{\log_2 n}$, za $n = 1024$ $\frac{1024}{10} = 102.4$ PUTA!