

Hitro urejanje (Quick sort)

- Ideja: razbijmo seznam na manjše kose, ki jih posebej uredimo
- Deli in vladaj
 - Težek problem razbijemo na več lažjih (trivialnih), jih rešimo in rešitve na koncu združimo v končno rešitev

4	3	6	7	8	1	5	2
---	---	---	---	---	---	---	---

Hitro urejanje (Quick sort)

- Kako seznam vrednosti razbijemo na več lažjih problemov?
 - Izberemo neko vrednost (primerjalni element, pivot)
 - Manjše vrednosti v seznamu premaknemo levo, večje pa desno od primerjalnega/delilnega elementa
 - Posebej uredimo levi in desni del
 - Delilni element je že na pravem mestu

4	3	6	7	8	1	5	2
---	---	---	---	---	---	---	---

Hitro uredi – konceptualen primer

4	3	6	7	8	1	5	2
---	---	---	---	---	---	---	---

- Zaporedje razdeli na dva dela glede na vrednost enega izmed elementov (prvi element = primerjalni element)





<u>4</u>	3	6	7	8	1	5	2
----------	---	---	---	---	---	---	---

- Ostale vrednosti premakni levo ali desno glede na primerjalni element



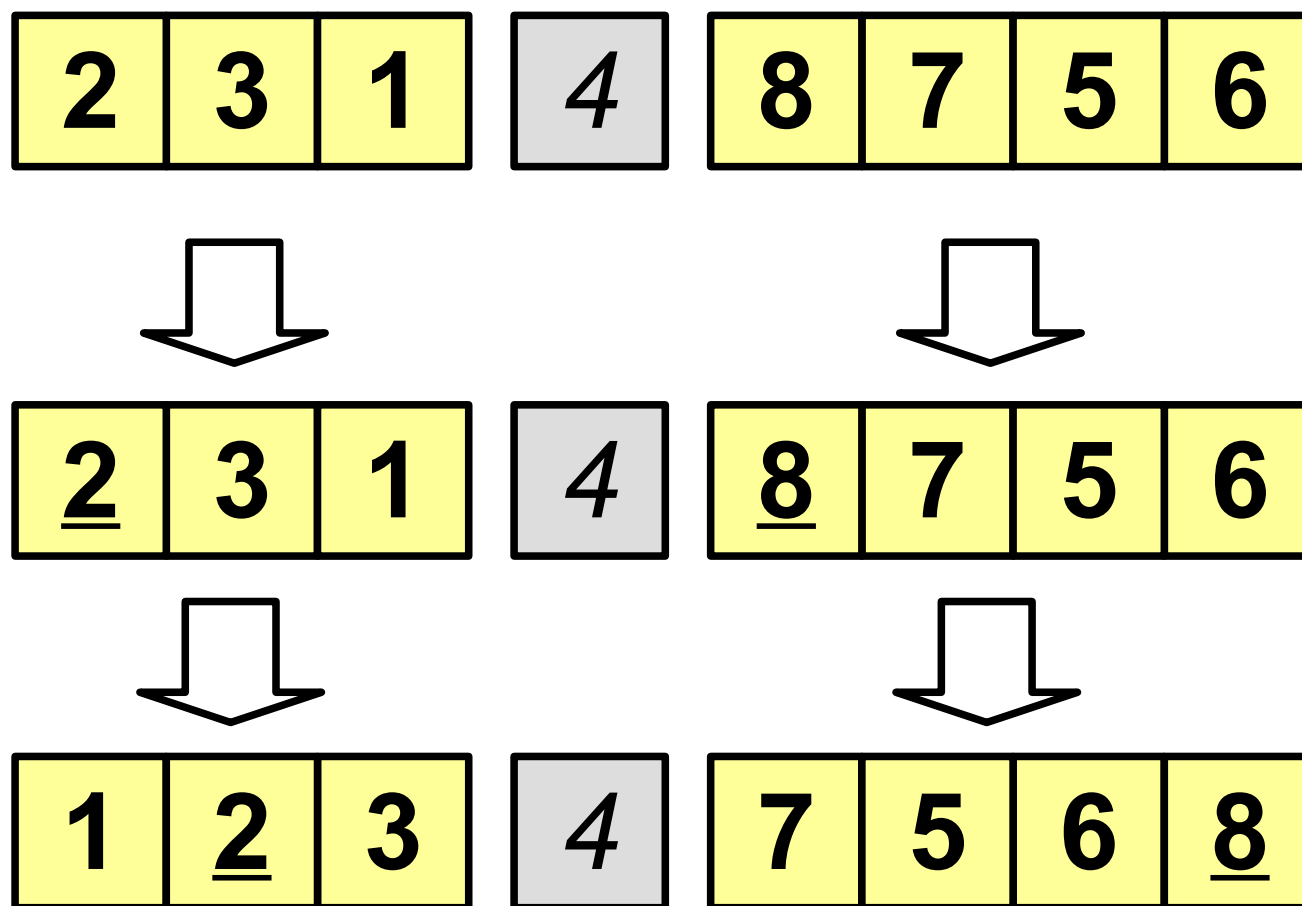
2	3	1	<u>4</u>	8	7	5	6
---	---	---	----------	---	---	---	---

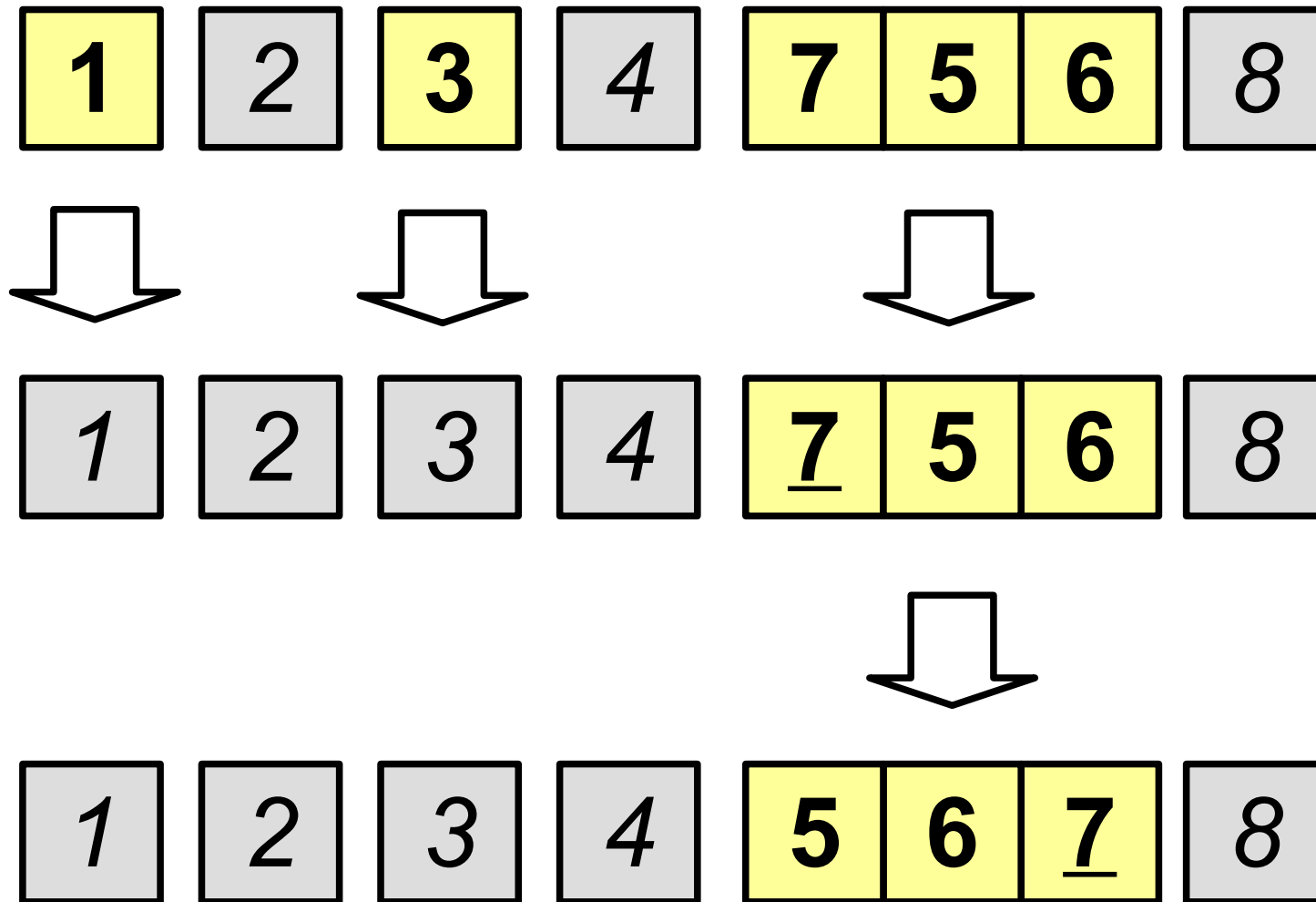
- Problem smo razdelili na dva lažja (manjša) problema

Hitro uredi – konceptualen primer

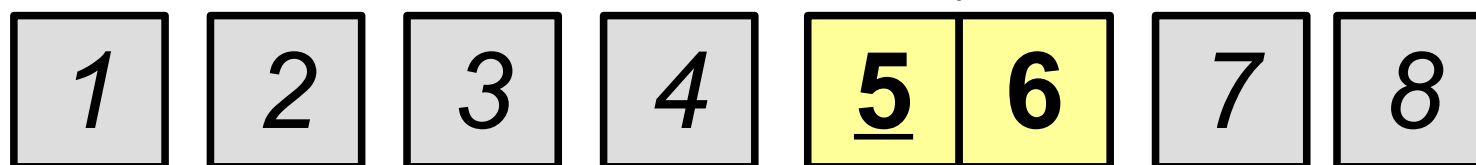
- Postopek od prej ponovimo na levem in desnem delu



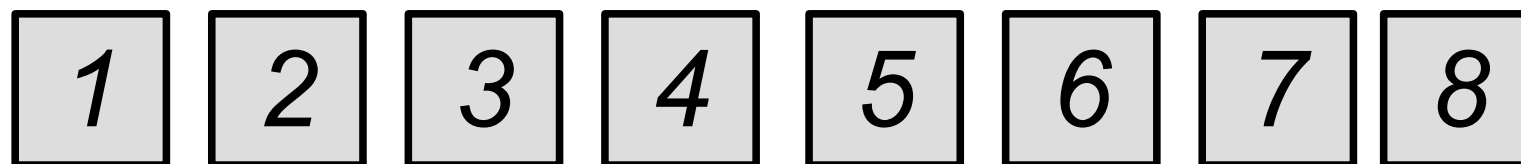
Hitro uredi – konceptualen primer



Hitro uredi – konceptualen primer



Končni rezultat



Hitro urejanje polja

- V C++ uporabljamo polja, ki jih je težje premikati in združevati
- Potrebujemo primernejši pristop
 - Premikali bomo vrednosti znotraj polja
 - Primerjalni element bomo postavili na primerno mesto

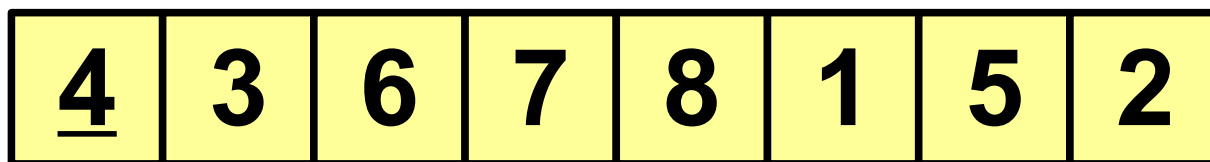
Delitev polja

- Potrebno postaviti primerjalni element na pravo mesto
- Vrednosti, ki so manjše, postavimo levo in vrednosti, ki so večje, postavimo desno

<u>4</u>	3	6	7	8	1	5	2
----------	---	---	---	---	---	---	---

Delitev polja

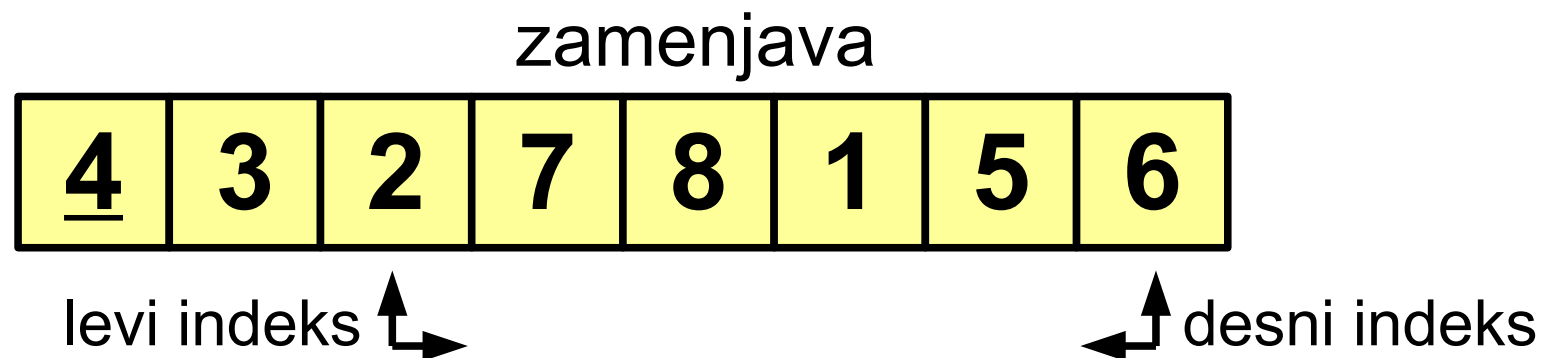
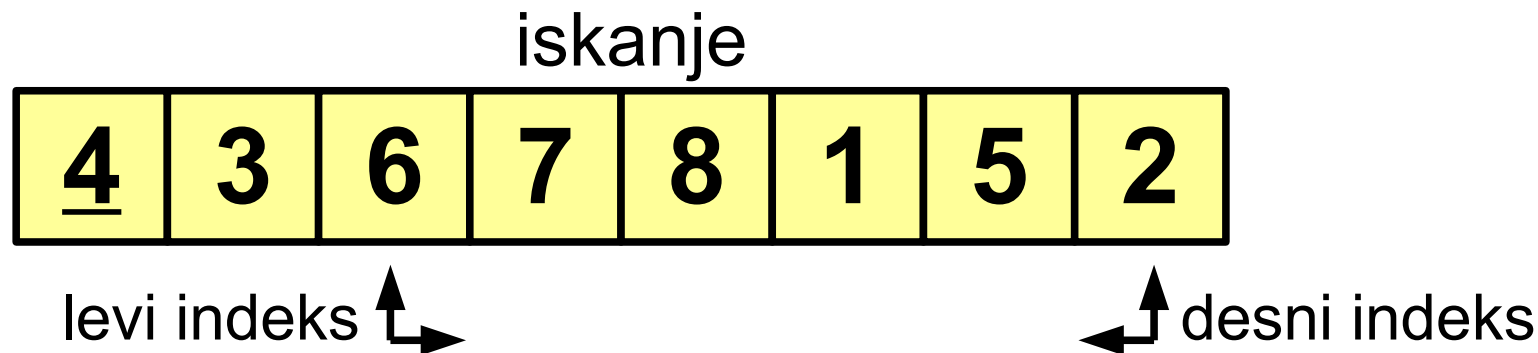
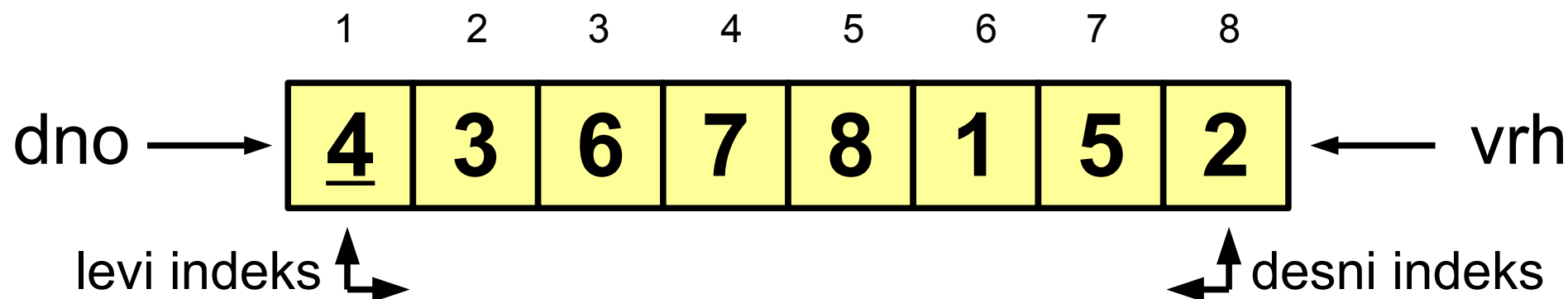
- Primerjalni element naj bo prvi element
- Uvedemo levi indeks
 - Pomikamo ga desno in iščemo večje elemente od primerjalnega elementa
- Uvedemo desni indeks
 - Pomikamo ga levo in iščemo manjše elemente od primerjalnega elementa
- Najdene vrednosti zamenjujemo → kar je levo od levega indeksa bo manjše od primerjalnega elementa...
- Na koncu primerjalni element postavimo na pravo mesto (desni indeks)



levi indeks ↗

↖ desni indeks

Delitev polja - primer



Delitev polja - primer

<u>4</u>	3	2	7	8	1	5	6
----------	---	---	---	---	---	---	---

levi indeks ↗

↖ desni indeks

iskanje

<u>4</u>	3	2	7	8	1	5	6
----------	---	---	---	---	---	---	---

levi indeks ↗

↖ desni indeks

zamenjava

<u>4</u>	3	2	1	8	7	5	6
----------	---	---	---	---	---	---	---

levi indeks ↗

↖ desni indeks

Delitev polja - primer

<u>4</u>	3	2	1	8	7	5	6
----------	---	---	---	---	---	---	---

levi indeks ↗ ↖ desni indeks

iskanje

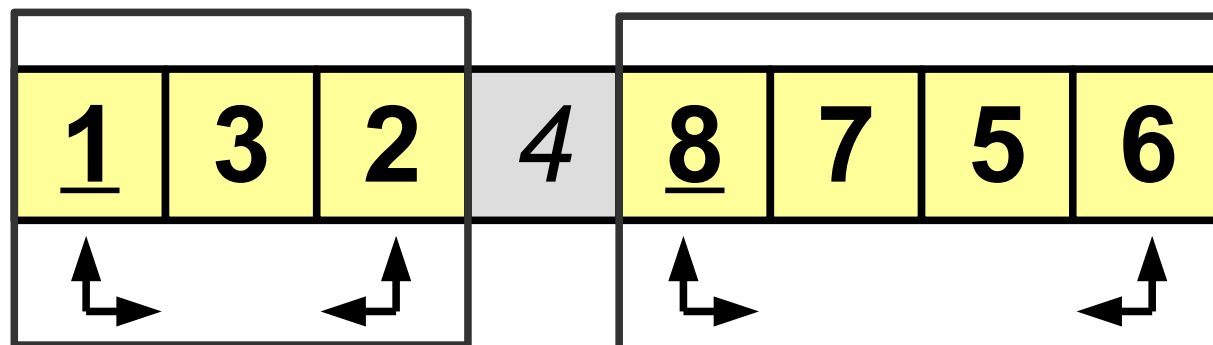
<u>4</u>	3	2	1	8	7	5	6
----------	---	---	---	---	---	---	---

levi indeks ↗ ↖ desni indeks

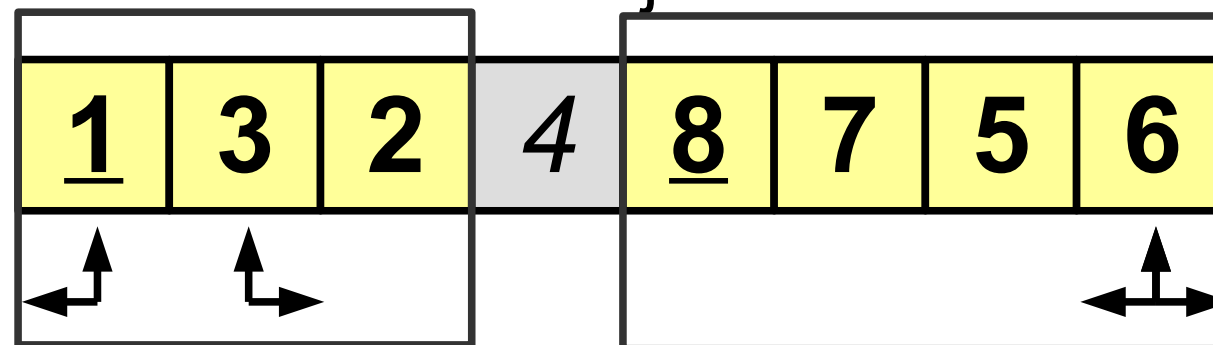
1	3	2	<u>4</u>	8	7	5	6

postopek ponovimo

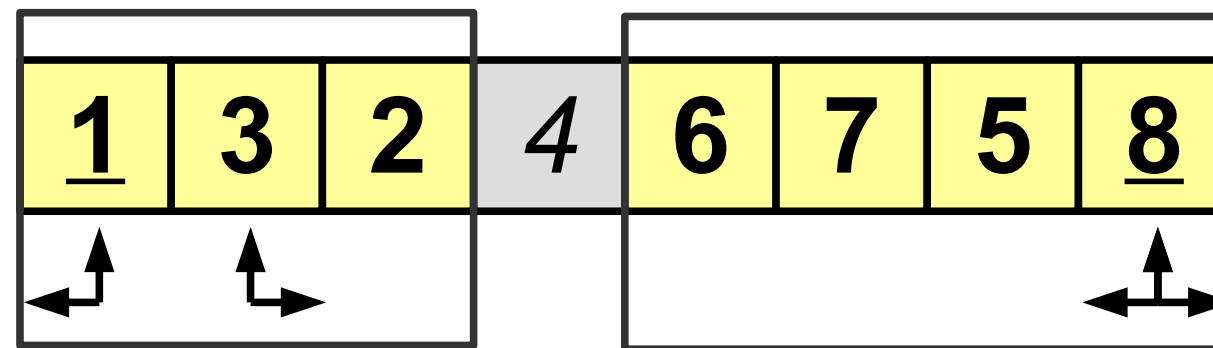
postopek ponovimo

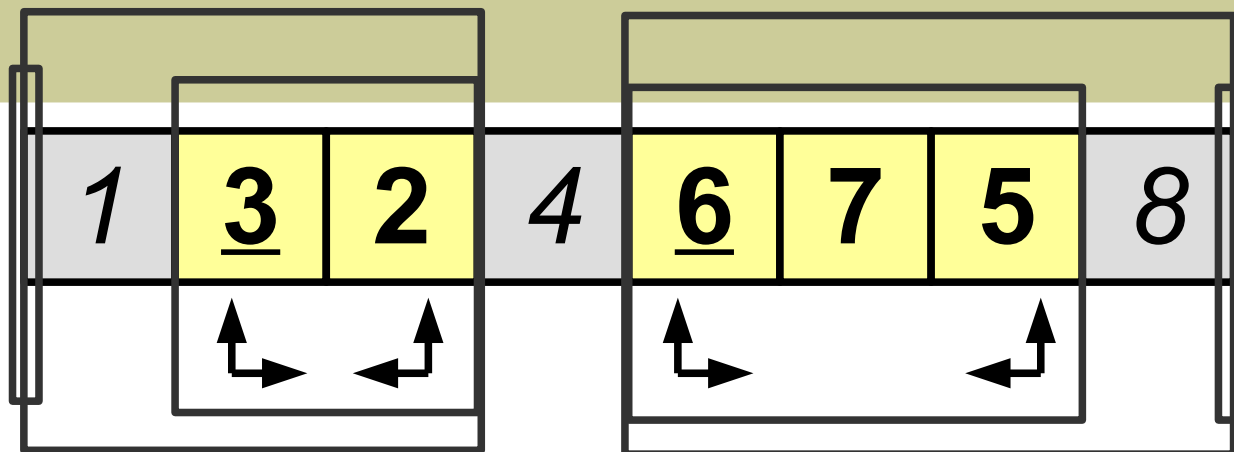


iskanje

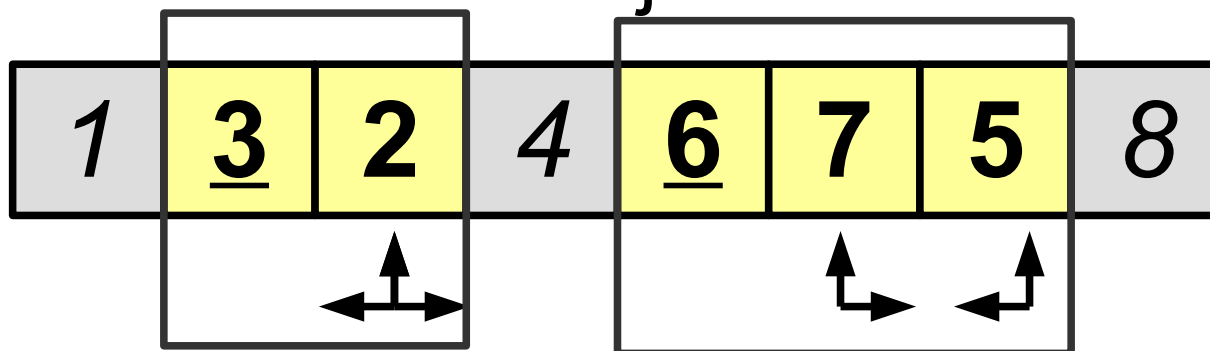


zamenjava

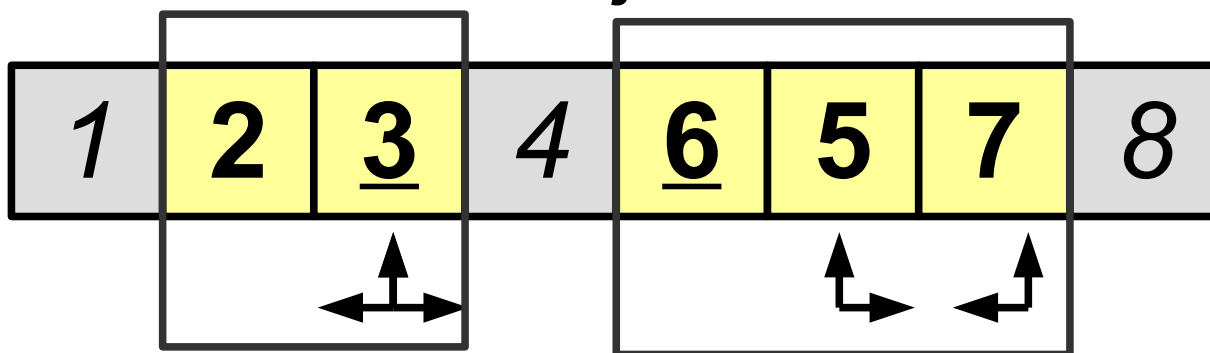


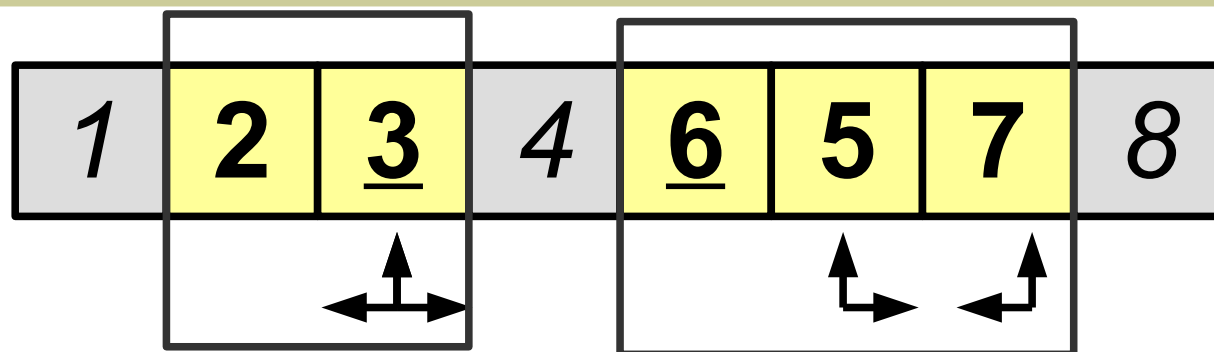


iskanje

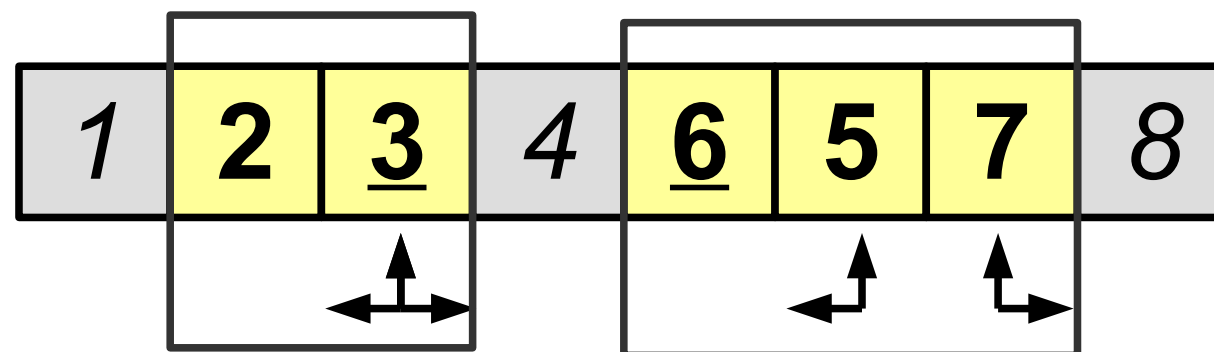


zamenjava

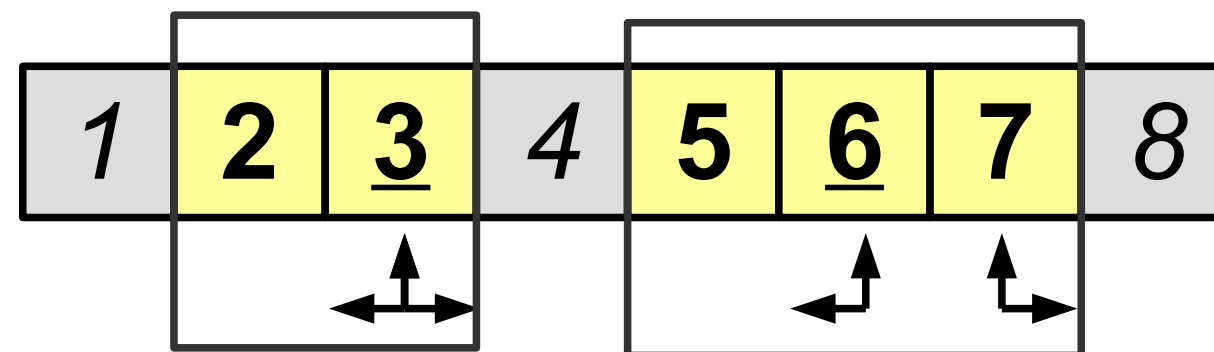


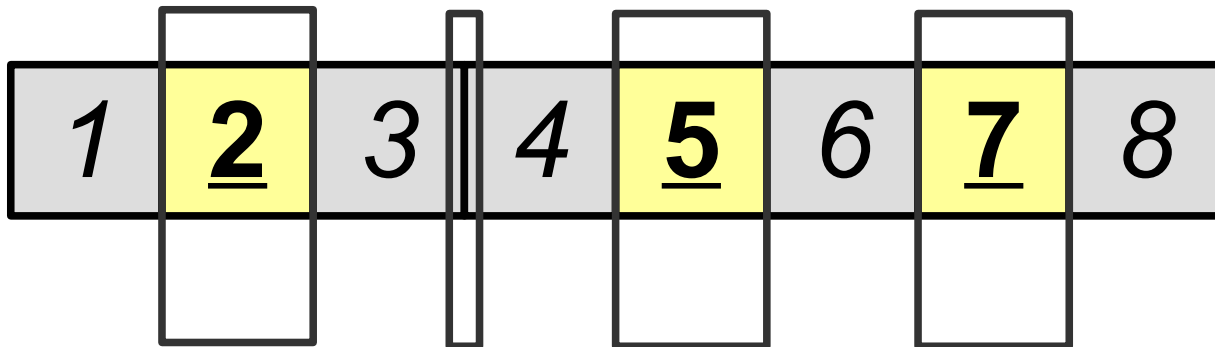
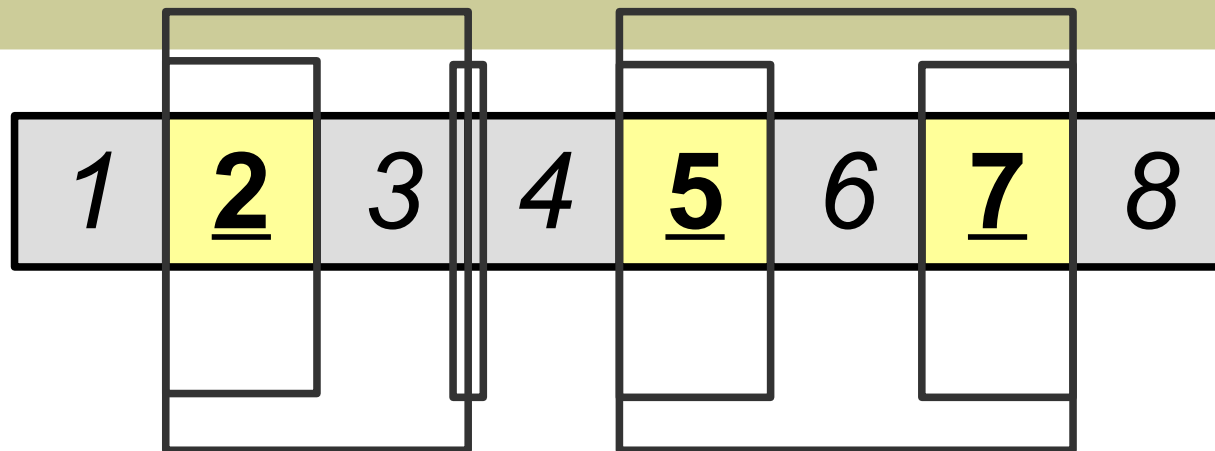


iskanje



zamenjava





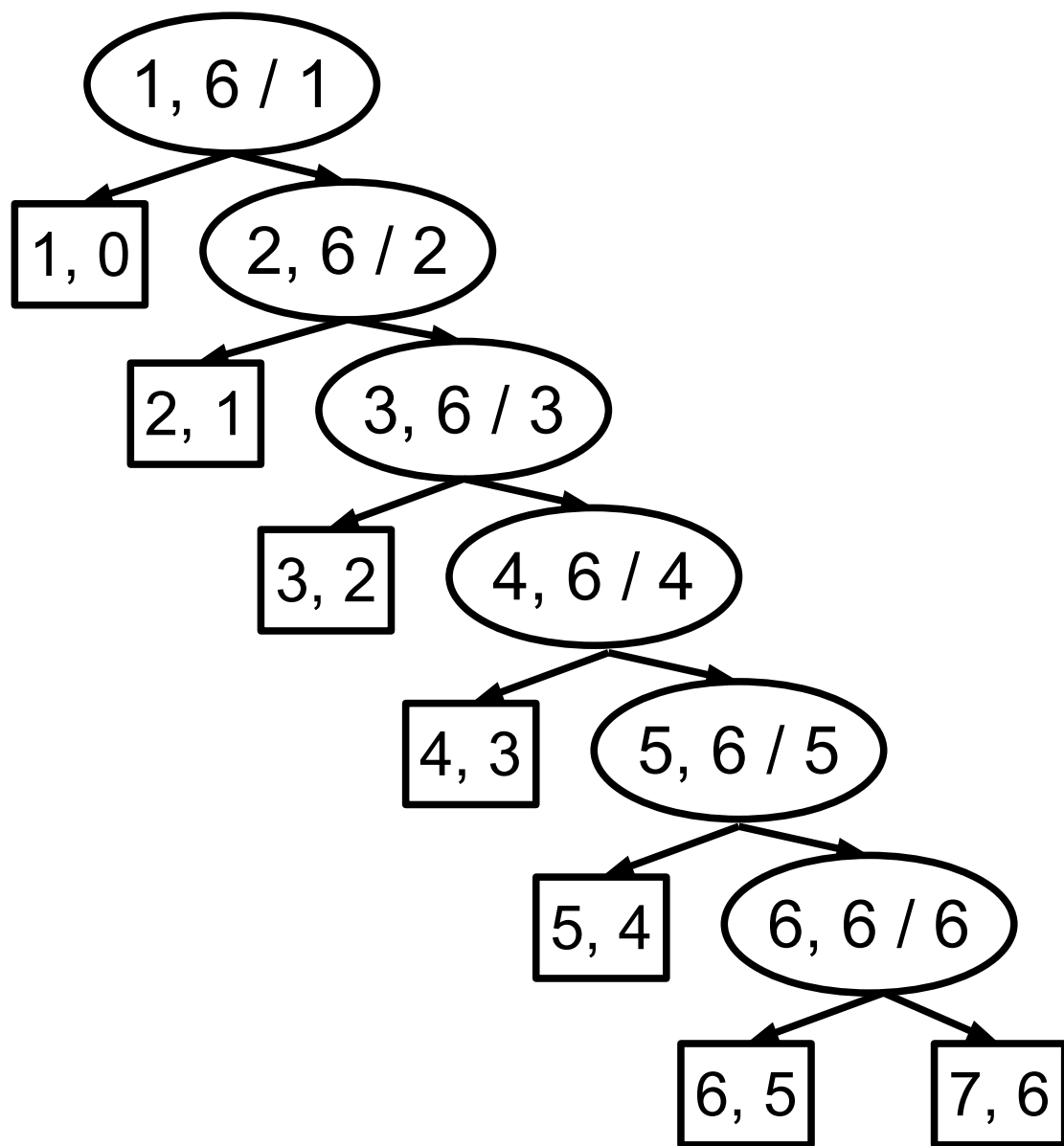
1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---


```
function HITRO_URED(a, dno, vrh)
begin
    if dno < vrh then
        begin
            j := DELI(a, dno, vrh);
            HITRO_URED(a, dno, j-1);
            HITRO_URED(a, j+1, vrh);
        end;
    end
```

```
function DELI(a, dno, vrh)
begin
    pe := a[dno];      // pivot
    l := dno;          // levi indeks
    d := vrh;          // desni indeks
    while indeksa l in d se nista prekrižala do
        begin
            pomikaj l desno, dokler  $a[l] \leq pe$  in  $l < vrh$ 
            pomikaj d levo, dokler  $a[d] \geq pe$  in  $d > dno$ 

            if indeksa l in d se nista prekrižala then
                Zamenjanj elementa  $a[l]$  in  $a[d]$ ;
            end;
            Zamenjaj elementa  $a[dno]$  in  $a[d]$ ;
        return d;
    end
```

Robni primer – že urejeno zaporedje

[illegible]

Robni primer – že urejeno zaporedje

- Drevo rekurzivnih klicev se lahko izrodi
 - Urejanje že urejenega zaporedja
 - Dobimo kvadratno časovno zahtevnost $O(n^2)$
 - Kam se shrani n spremenljivk od rekurzije?
- Rešitve: uporaba mediane
 - Delilni element naj bo sredinski → sredinski element pred delitvijo postavimo na prvo mesto

```
...  
m := dno + (vrh-dno)/2;  
Zamenjaj elementa a[dno] in a[m];  
...
```

Zahteve naloge

- Implementirajte algoritem hitro uredi z mediano, brez mediane in aplikacijo:
 - Opcije 1, 2, 3: Uporabnik določi dolžino novega zaporedja
 - Preverjanje zaporedja (Opcija 5): desni element ne sme biti manjši od levega
 - Po urejanju (opcije 6, 7, 8) se mora izpisati čas urejanja
 - Opcija 8): Poljuben algoritem urejanja (npr.: urejanje z izbiranjem, mehurčki, ...)
 - Zahtevana zmožnost hitrega urejanja z mediano 1M podatkov (urejenih in naključnih)!

Hitro uredi – izbira:

- 1) Generiraj naključno zaporedje
- 2) Generiraj naraščajoče zaporedje
- 3) Generiraj padajoče zaporedje
- 4) Izpis zaporedja
- 5) Preveri ali je zaporedje urejeno
- 6) Uredi s hitrim urejanjem brez mediane
- 7) Uredi s hitrim urejanjem z mediano
- 8) Uredi z drugim algoritmom za urejanjem
- 9) Konec

Izbira:

Zahteve naloge - testiranje

- Program testirajte in vaše ugotovitve zapišite kot komentar v izvorno kodo ali v tekstovno datoteko
- Izmerite čase urejanja za naključna in tudi naraščajoča zaporedja dolžin vsaj 10000, 20000, 30000, 40000, 50000 elementov
 - Hitro uredi z mediano
 - Hitro uredi brez mediane
 - Drugi poljubni algoritem urejanja
- Zapišite vseh 30 časov in narišite grafe (čas urejanja glede na dolžino zaporedja). Kot komentar zapišite ocenjeno časovno zahtevnost (1. predavanja).
 - $O(n^2)$, $O(n \log(n))$
- Za testiranje lahko uporabite tudi več različnih dolžin zaporedij

Zahteve naloge - testiranje

- V tekstovno datoteko zapišite tudi
 - Kaj se zgodi pri hitrem urejanju brez mediane 1000000 naraščajočih elementov
 - Če urejanje potrebuje več kot 1 minuto, ga lahko prekinete
 - Ali se hitro urejanje brez mediane padajočega zaporedja obnaša podobno kot pri naraščajočem zaporedju?
 - Kaj se zgodi, če na začetku znotraj funkcije `HITRO_UREDI` definirate dodano polje (`double x[10000];x[0]=0;`) in uredite 10000 naraščajočih elementov s hitro uredi brez mediane? Tokrat prevedite kot **debug/razhroščevanje**.
 - Kakšen je vzrok?
 - Kaj če urejate samo 10 elementov?

Zahteve naloge

Implementirajte algoritem hitro uredi z mediano, ki bo uredil zaporedje števil dolgo do 1.000.000 podatkov. Števila v zaporedju so lahko naključna (izbira 1), lahko pa so urejena v naraščajočem (izbira 2) ali pa padajočem vrstnem redu (izbira 3). Ob zagonu programa se mora zagnati meni, ki je prikazan na prejšnjih prosojnicah.

Ob izbiri menijske postavke za generiranje katerega od zaporedij, je potrebno od uporabnika zahtevati vpis dolžine zaporedja, nato pa v polje vnesti ustrezna števila. Pri urejanju (izbira 6, 7 ali 8) je potrebno izmeriti čas urejanja, ki ga izpišete, ko je urejanje končano. Pri izbiri 8 je potrebno zaporedje urediti z drugim poljubnim algoritmom za urejanje. Uporabite lahko npr.: urejanje z izbiranjem. Program se konča, ko uporabnik izbere menijsko postavko *Konec*.

Testiranje

Pri testiranju, vaš program prevedite kot »release« oz. »izdaja«. Algoritem za hitro urejanje brez mediane, algoritem za urejanje z mediano in drug poljuben algoritem urejanja je potrebno testirati vsaj na zaporedjih dolžine 10000, 20000, 30000, 40000 in 50000 elementov. To storite za:

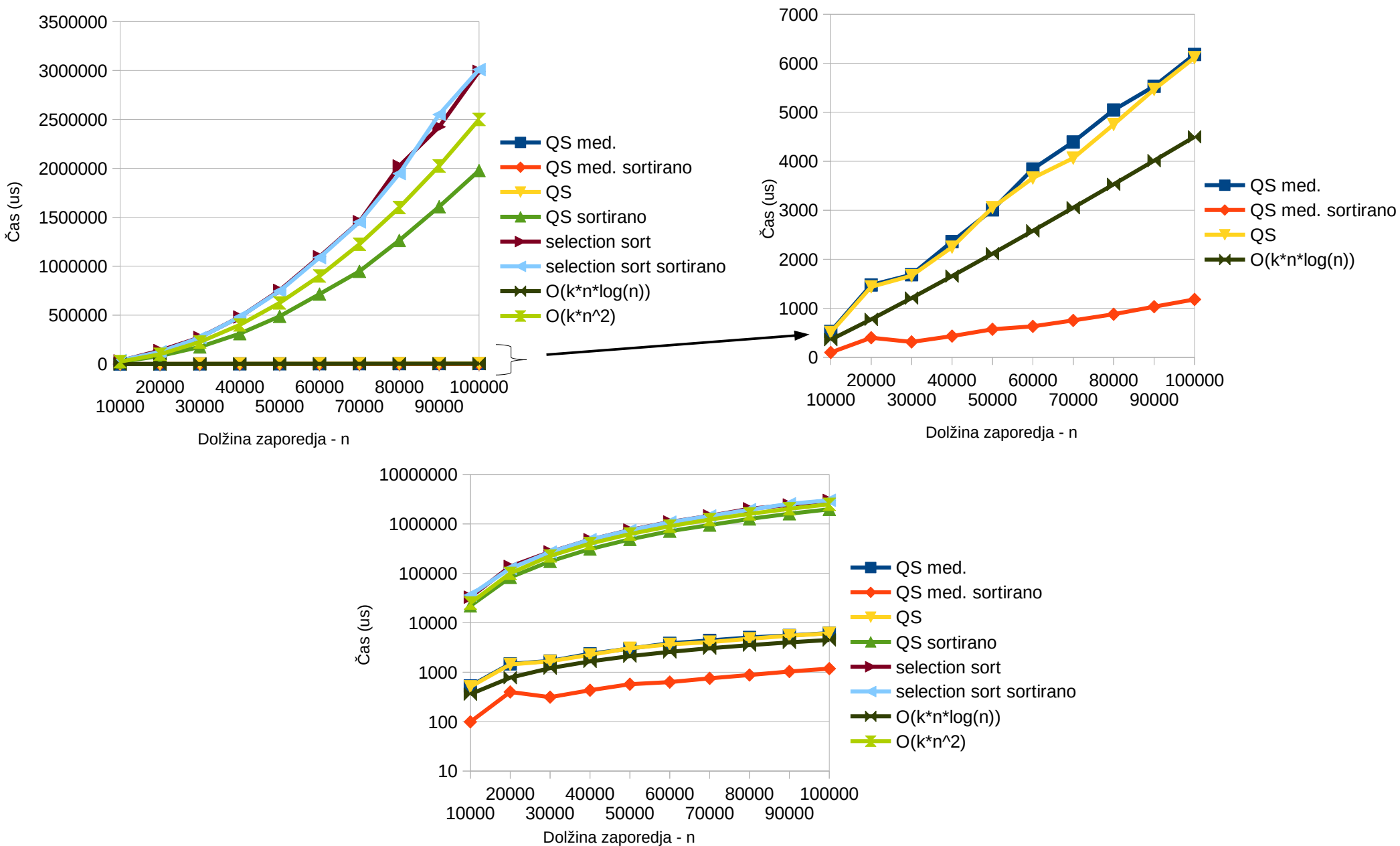
- naključno,
- naraščajoče zaporedje.

V izvorno datoteko kot komentar zapišite čase urejanja (skupno 30 časov) in vaše ugotovitve iz testiranj v nadaljevanju besedila. Izvorni kodi tudi priložite graf (slike) ali več grafov, ki prikazujejo čase urejanja vseh 6-ih kombinacij algoritmov glede na dolžino zaporedja. Na grafu naj os *x* predstavlja dolžino zaporedij, os *y* pa naj predstavlja čas urejanja zaporedja. Poskusite oceniti časovno zahtevnost vseh 6-ih kombinacij algoritmov.

Preverite ali se program res sesuje pri hitrem urejanje brez mediane 1000000 naraščajočih elementov (kar je tudi pričakovano). Ali se hitro urejanje brez mediane naraščajočega zaporedja obnaša podobno kot urejanje padajočega zaporedja?

Na koncu še testirajte, kaj se zgodi, če na začetku znotraj funkcije HITRO_UREDİ dodate vrstico: `double x[10000];x[0]=0;` in izvedete sortiranje 10.000 naraščajočih elementov s pomočjo hitro uredi brez mediane. Tokrat program prevedite kot »debug/ razhroščevanje«. Zakaj se program sesuje? Ali se kaj spremeni, če urejate samo 10 elementov? Ugotovitve zapišite v izvorno kodo kot komentar.

Čas trajanja urejanja (primer)



Preverjanje urejenosti zaporedja

```
function PREVERI(polje, dno, vrh)
begin
    for i:=dno to vrh-1 do
        if polje[i] > polje[i+1] then
            return Napačno urejeno zaporedje;
    return Zaporedje urejeno;
end
```

Urejanje z izbiranjem

```
function UREDI_Z_IZBIRANJEM(polje, velikostPolja)
begin
  for i:=1 to velikostPolja - 1 do
    begin
      minIndex := i;

      // iskanje min elementa desno od i
      for j:=i+1 to velikostPolja
        if polje[j] < polje[minIndex] then
          minIndex := j;

      if minIndex <> i then
        zamenjaj (polje[i], polje[minIndex]);
    end
  end
```

Merjenje časa izvajanja

```
#include <time.h>

...
clock_t start = clock();
qSort(0, 99999);
clock_t finish = clock();
double duration = (double) (finish - start) / CLOCKS_PER_SEC;
...
```

C++

```
#include <chrono>
using std::chrono::duration_cast;
using std::chrono::microseconds;
using std::chrono::steady_clock;
...
steady_clock::time_point start = steady_clock::now();
qSort(0, 99999);
steady_clock::time_point end = steady_clock::now();
std::cout << "Trajanje: " <<
    duration_cast<microseconds>(end - start).count()
    << "us";
```

C++11

Generiranje naključnih števil

C++

```
#include <cstdlib>
...
int v = rand() % 10000000;
...
```

C++11

```
#include <random>
std::mt19937 rng;
std::uniform_int_distribution<uint32_t> uint_dist(0,10000000);
...
int v = uint_dist(rng);
...
```

- Vrednost naloge: 6 točk
 - Drugi algoritem urejanja: 1 točka
 - Testiranje: 1,5 točke