Vaja 1 – Hitro urejanje

1. Splošna predstavitev problema

V praksi se pogosto srečujemo z veliko količino podatkov, ki jih je potrebno urediti po nekem kriteriju. Ker je urejanje zgolj ena izmed operacij, mora biti le – to hitro. Zaradi tega je skozi čas nastalo veliko algoritmov, ki nam omogočajo urejanje podatkov v logaritemskem času, se pravi v času $O(n \log n)$. Eden najbolj znanih in najbolj razširjenih je algoritem Hitro uredi (*quick sort*), ki ga uporabljamo za urejanje podatkov, zapisanih v polju.

Algoritem Hitro uredi deluje po strategiji deli in vladaj, to pomeni, da nek problem razdelimo v identične podprobleme, ki so zaradi svoje majhnosti lažje rešljivi. Ko podprobleme rešimo, je potrebno delne rešitve združiti v končno rešitev.

Pri tem se pojavita dve vprašanji, do kakšne velikosti bomo zaporedje delili in na kakšen način bomo delne rešitve združili v končno rešitev. Odgovor na prvo vprašanje je preprost. Problem bomo delili tako dolgo, dokler ne pridemo do primera, ki ga je enostavno rešiti. Če upoštevamo, da gre v našem primeru za urejanje, to pomeni, da z delitvijo končamo, ko pridemo do trivialnega primera sortiranja, to je da dobimo v polju eno samo vrednost, ali pa je polje prazno.

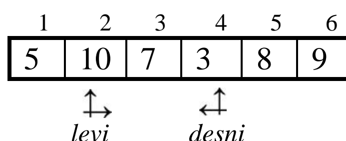
Odgovor na drugo vprašanje je zahtevnejši in je skrit v delitvi problema na manjše podprobleme. V polju si je namreč potrebno izbrati nek primerjalni element glede na katerega pri iskanju točke delitve zaporedje urejamo v levi del, kjer so vrednosti, manjše od primerjalnega elementa in v desni del, kjer so vrednosti večje ali enake primerjalnemu elementu. Poglejmo primer tega iskanja na konkretnem primeru.

Imejmo polje šestih števil 5, 10, 7, 3, 8 in 9, prikazano na sliki 1. Zaporedje v polju je določeno z dvema indeksoma: *dno* in *vrh*. Da začnemo s preurejanjem zaporedja, moramo določiti primerjalni element. Najenostavnejša izbira je ta, da za primerjalni element vzamemo kar element *dna*, ki je v našem primeru 5.

1	2	3	4	5	6
5	10	7	3	8	9
↑ *dno*					↑ *vrh*

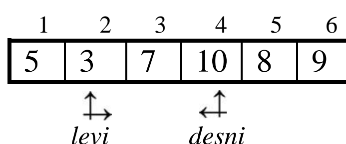
Slika 1: Primer zaporedja, v polju

Preden lahko začnemo z delnim urejanjem, je potrebno definirati še dva indeksa, s katerima se bomo pomikali po polju, ki ju bomo poimenovali *levi indeks* in *desni indeks*. *Levi indeks* začne z iskanjem pri dnu in se pomika proti vrhu, z njim pa iščemo vrednosti, ki so strogo večje od primerjalnega elementa. *Desni indeks* začne z iskanjem pri vrhu, se premika proti dnu in išče števila, strogo manjša od primerjalnega elementa. Ko najdeta taki števili, se iskanje ustavi, kar lahko vidimo na sliki 2.



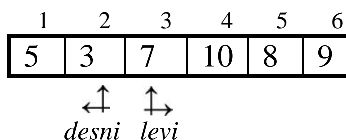
Slika 2: Iskanje večjega in manjšega elementa s pomočjo levega in desnega indeksa

Ker je primerjalni element enak 5, se *levi indeks* ustavi pri indeksu 2, saj je 10 prvo število, večje od 5, *desni indeks* pa se ustavi pri indeksu 4, saj je vrednost 3 prva manjša od vrednosti 5. Sedaj števili 10 in 3 zamenjamo, pri čemer dobimo situacijo, ki jo prikazuje slika 3.



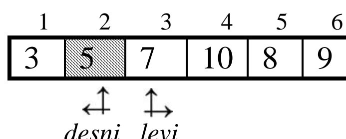
Slika 3: Zamenjava vrednosti, pri indeksih levi in desni

Po zamenjavi najdenih vrednosti z iskanjem nadaljujemo in *levi indeks* se ustavi pri vrednosti 7, *desni* pa se ustavi pri vrednosti 3, ki je spet prva manjša od primerjalnega elementa (glej sliko 4).



Slika 4: Zamenjava vrednosti, pri indeksih levi in desni

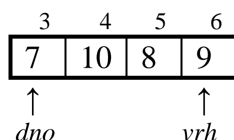
Iz slike 4 je razvidno, da sta se *levi* in *desni* indeks prekrižala, kar pomeni, da smo polje v celoti pregledali, zato v tem primeru zamenjamo vrednost na katero kaže *desni indeks* in element *dna*, zamenjamo torej števili 3 in 5. S to zamenjavo smo dosegli, da so sedaj levo od vrednosti 5 vsa števila, ki so manjša od njega, desno pa vsa števila, ki so večja ali enaka 5, medtem ko je 5 ravno na tistem mestu v polju, kjer mora biti, se pravi je že urejeno (glej sliko 5).



Slika 5: Zamenjava vrednosti, pri križanju levega in desnega indeksa

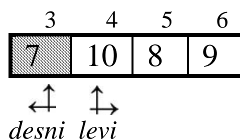
Zaporedje sedaj lahko razpade na dva dela, *desni indeks* postane delilni, saj se razpad zgodi ravno na njegovi poziciji (v našem primeru 2). V levem podzaporedju imamo eno samo število, to je 3, kar pomeni, da je to podzaporedje urejeno, v desnem delu pa

imamo števila 7, 10, 8 in 9. To zaporedje pa je potrebno še urediti. Urejanje desnega podzaporedja poteka na enak način kot prej, le da je indeks *dna* v tem primeru enak 3, indeks *vrha* pa ostane enak kot prej, torej 6, glej sliko 6.



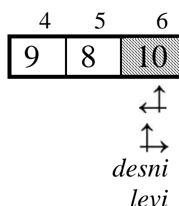
Slika 6: Urejanje desnega podzaporedja

Tudi pri urejanju desnega podzaporedja moramo najprej določiti primerjalni element, ki je prav tako element dna, in je sedaj enak 7. Tako sedaj z *levim indeksom* iščemo števila večja od sedem, z *desnim* pa števila manjša od sedem. *Levi indeks* se ustavi pri vrednosti 10, saj je to že prva vrednost, ki je večja od 7, *desni indeks* pa se ustavi pri 7, saj je 7 prvo število v zaporedju, *desni indeks* pa ne more nižje od *dna*. Situacija, ki jo dobimo, je prikazana na sliki 7.



Slika 7: Križanje levega in desnega indeksa pri urejanju desnega podzaporedja

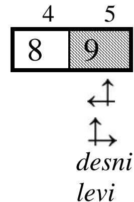
Ker sta se *levi* in *desni indeks* prekrižala, zaporedje spet razpade v dve podzaporedji. Levo podzaporedje je prazno, saj nimamo nobene vrednosti, manjše od sedem in je zato urejeno, v desnem podzaporedju pa imamo vrednosti 10, 8 in 9, ki jih je spet potrebno urediti.
Primerjalni element je v tem primeru 10. *Levi indeks* išče vrednosti večje od 10, *desni* pa manjše. *Desni indeks* se sedaj ne premakne nikamor, saj je 9 že manjše od 10, *levi indeks* pa prav tako pride do iste vrednosti, saj več ne more, ker je dosegel vrh zaporedja. Tako sta se *levi* in *desni indeks* spet prekrižala. To pomeni, da je potrebno zamenjati element na katerega kaže *desni indeks* in element *dna*, to je, 9 zamenjamo z 10, ki s tem postane urejeno, glej sliko 8.



Slika 8: Zamenjava elementov zaporedja ob križanju levega in desnega indeksa pri vnovičnem urejanju desnega podzaporedja

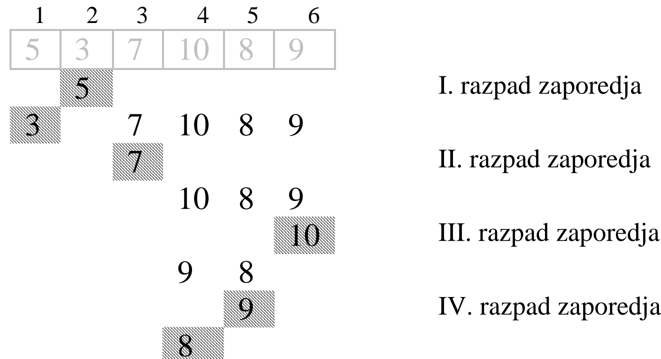
Tudi v tem primeru zaporedje razpade na dva dela. V levem delu so vsa števila, ki so manjša od 10, to sta vrednosti 9 in 8, v desnem podzaporedju pa so števila, ki so večja od 10. Ker takih števil ni bilo, je desno podzaporedje prazno in s tem tudi urejeno.

Sedaj je potrebno urediti samo še levo podzaporedje z vrednostma 9 in 8. Indeks *dno* je v tem primeru enak 4, indeks *vrha* pa 5. Novi primerjalni element je vrednost 9. Z levim indeksom znova iščemo števila večja od 9, z desnim pa števila manjša od 9. Tudi v tem primeru se *desni indeks* ne premakne, saj 8 ustreza kriterijem iskanja. *Levi indeks* pa se enako kot prej premakne do iste vrednosti, saj je tudi v tem primeru prišel do vrha in dalje ne more. Indeksa sta se prekrižala, torej zamenjamo vrednosti 9 in 8, s čemer postavimo 9 na pravo mesto, glej sliko 9.



Slika 9: Zamenjava elementov zaporedja ob križanju levega in desnega indeksa pri urejanju levega podzaporedja

Ob vnovičnem križanju *levega* in *desnega* indeksa pride do razpada zaporedja. In sicer dobimo levo podzaporedje z vrednostjo 8 in desno podzaporedje, ki je prazno, saj v podzaporedju nismo imeli elementa večjega od 9. V obeh primerih sta podzaporedji urejeni, s tem pa je urejeno tudi zaporedje v celoti. Da je temu res tako, se lahko prepričamo na sliki 10.



Slika 10: Urejanje polj ob razpadih na manjša podzaporedja

Kot lahko vidimo, smo rešili tudi vprašanje spajanja rešitev posameznih podproblemov v rešitev celotnega problema, saj celotna rešitev nastaja hkrati z reševanjem podproblemov, zato za končno spajanje delnih rešitev ni potrebno skrbeti. Prepričali pa smo se lahko tudi o tem, da vsa podzaporedja obravnavamo na enak način. To nam daje tudi napotek za implementacijo.
Program za urejanje bomo razdelili v dve proceduri. Prva bo skrbela za rekurzivne klice same sebe, in jo bomo imenovali *HITRO_UREDI*, druga pa bo poskrbela za delno urejanje in iskanje delilnega indeksa. To proceduro bomo imenovali *DELI*. Ker bo naše zaporedje na vsakem nivoju rekurzije razpadlo v dve podzaporedji, lahko tvorimo binarno drevo, ki ga v primeru, da ga opremimo s klici parametrov procedure *HITRO-_UREDI* imenujemo **drevo rekurzivnih klicev**.

2. Pomoč pri implementaciji

Najprej si bomo ogledali postopek za implementacijo procedur *HITRO_UREDI* in *DELI* ter način gradnje drevesa rekurzivnih klicev, nato pa bomo opisali način preverjanja pravilnosti delovanja obeh funkcij in način merjenja časa izvajanja algoritma.

Funkcije za urejanje zaporedja

V prejšnji točki smo podrobno predstavili delovanje algoritma za hitro urejanje, tako da sedaj zapišimo povedano v obliki psevdokoda. Najprej poglejmo funkcijo *DELI*. Njen psevdokod je zapisan v izpisu 1.

```
function DELI(a, dno, vrh)
 begin
      l // levi indeks
      d // desni indeks
      while indeksa se nista prekrižala do
        begin
          poišči prvi tak l, da velja a[l] > pe in l < vrh

          poišči prvi tak d, da velja a[d] < pe in d > dno

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

Izpis 1: Psevdokod funkcije *DELI*

Kot lahko vidimo v izpisu 1, ima funkcija *DELI* tri vhodne parametre: kazalec na začetek polja s podatki, *a*, in indeksa *dno* ter *vrh*, s katerima je določeno zaporedje v polju. V kolikor je polje definirano globalno, imamo samo dva parametra, *dno* in *vrh*. Psevdokod procedure *DELI* delno ureja dano zaporedje na način opisan v poglavju 1. Ta način pa ni najboljši, v kolikor so podatki v zaporedju že urejeni. V tem primeru se drevo rekurzivnih klicev namreč izrodi, časovna zahtevnost urejanja pa postane enaka $O(n^2)$. Da to preprečimo, je treba na začetek funkcije *DELI* dodati naslednjo sekvenco ukazov:

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

Izpis 2: Razširitev funkcije *DELI* z računanjem mediame

Spremenljivko *m* v izpisu 2 imenujemo **mediana**, ker je to indeks srednjega elementa v zaporedju. V kolikor *DELI* opremimo s kodo iz izpisa 2, dobimo hitro urejanje z

mediano. To pomeni, da je sedaj naš primerjalni element sredinski element zaporedja. Za urejanje z mediano je značilno to, da je izjemno hitro v kolikor je zaporedje urejeno, pa naj bo to v naraščajočem ali padajočem vrstnem redu.

Ostane nam še psevdokod funkcije *HITRO_UREDI*, ko bo poskrbel za pravilne rekurzivne klice. Psevdokod najdemo v izpisu 3.

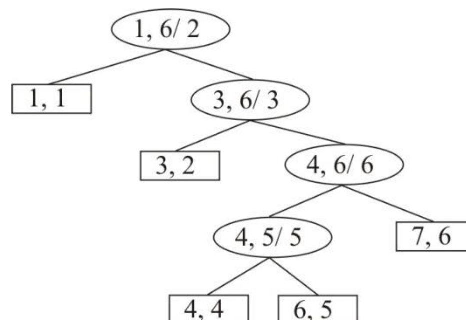
```
function HITRO_UREDI(a, dno, vrh)
 begin
      if dno < vrh then
        begin
          j := DELI(a, dno, vrh);
          HITRO_UREDI(a, dno, j-1);
          HITRO_UREDI(a, j+1, vrh);
        end;
end
```

Izpis 3: Psevdokod funkcije *HITRO_UREDI*

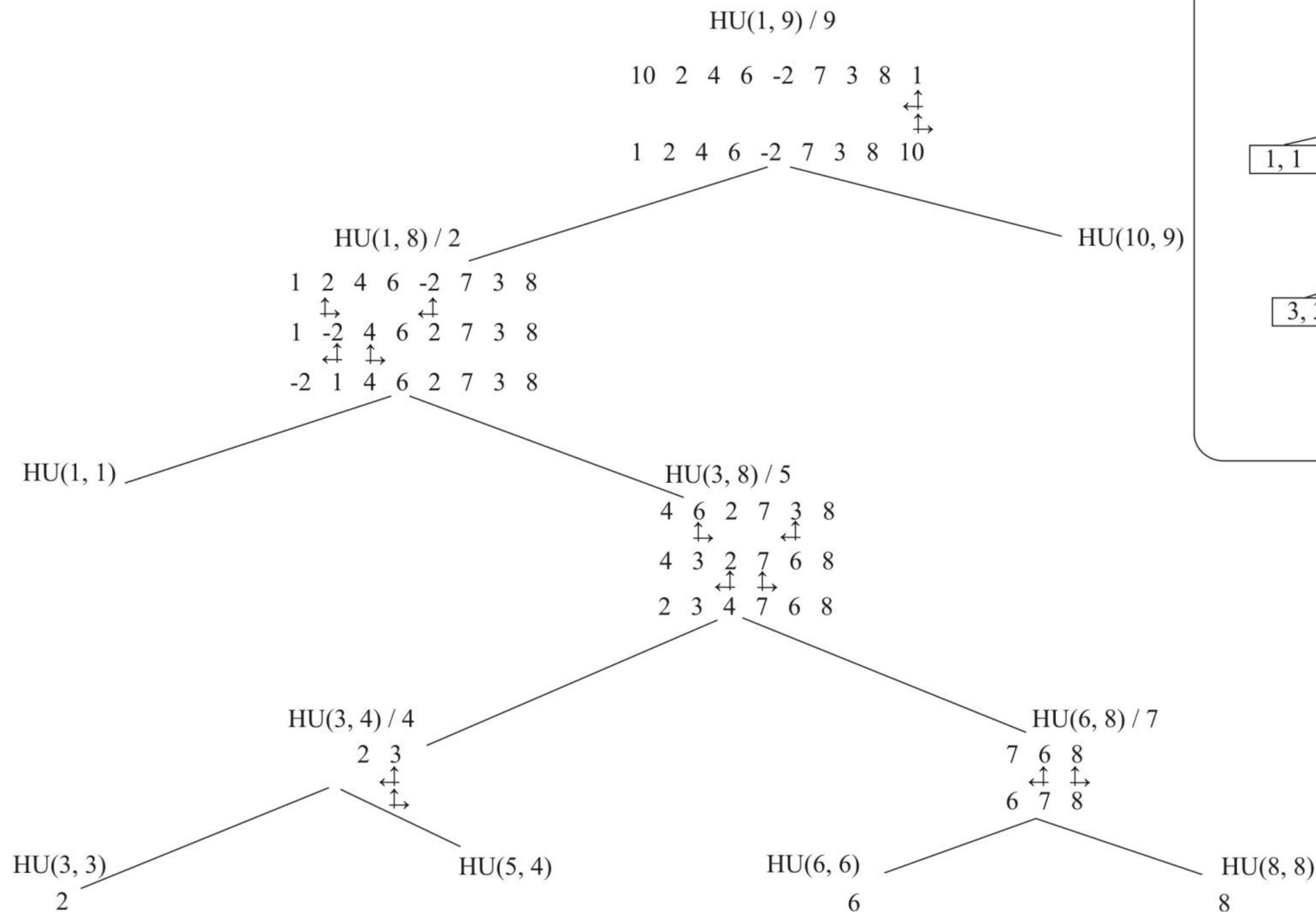
Tudi tukaj velja, da ima funkcija *HITRO_UREDI* tri vhodne parametre, v primeru, če polje ni definirano globalno, sicer pa ima samo dva parametra, to sta indeksa *dno* in *vrh*.

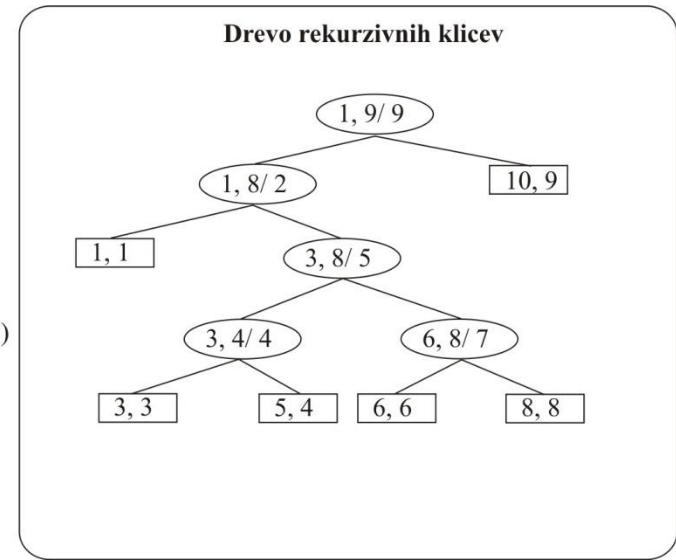
Drevo rekurzivnih klicev

Pojem drevesa rekurzivnih klicev smo definirali že v poglavju 1, nismo pa povedali, kako ga dobimo. Za generiranje drevesa rekurzivnih klicev je ključen algoritem v izpisu 3, saj v vozlišča tega drevesa vpisujemo indeksa *dno* in *vrh*, s katerima je klicana funkcija *HITRO_UREDI*, zraven pa še posebej pripišemo vrednost *delilnega indeksa*, ki ga vrne funkcija *DELI*. Vozlišče drevesa rekurzivnih klicev ima tako obliko (*dno*, *vrh/ delilni indeks*). Primer drevesa rekurzivnih klicev za primer s po-glavja 1 lahko vidimo na sliki 11.



Slika 11: Drevo rekurzivnih klicev za primer s poglavja 1





Slika 12: Primer urejanja zaporedja in gradnja drevesa rekurzivnih klicev

Na sliki 12 je prikazan še en primer gradnje drevesa rekurzivnih klicev. Listi drevesa so prazni klici funkcije *HITRO_UREDI*, to pomeni, da indeks *dno* ni manjši od indeksa *vrh*. Liste označujemo s kvadrati, medtem ko polne klice procedure označujemo z elipso.

Preverjanje pravilnosti delovanja

Ker je algoritem za hitro urejanje namenjen urejanju velikemu številu podatkov, je težko preveriti pravilnost delovanja, še posebej če upoštevamo, da je lahko v zaporedju več sto tisoč števil. Zaradi tega bomo preverjanje opravili s pomočjo funkcije, ki upošteva, da v urejenem zaporedju naslednik ne sme biti manjši od svojega predhodnika. Psevdokod funkcije za preverjanje je prikazan v izpisu 4.

```
function PREVERI(a, dno, vrh)
 begin
      for i:=dno to vrh-1 do
        if a[i+1] < a[i] then
          Izpiši napačno urejeno zaporedje;
 end
```

Izpis 4: Psevdokod funkcije *PREVERI*

Tudi v tem primeru velja, da v kolikor je polje definirano globalno, kazalca na začetek polja ni treba vnašati kot vhodni parameter v funkcijo.

Merjenje časa izvajanja programa

Hitrost delovanja algoritma je zelo pomembna pri določanju njegove kvalitete, zato je pomembno, da znamo le – to tudi izmeriti. V našem primeru uporabimo knjižnico `<time.h>`. Meritev časa izvajanja urejanja 100.000 števil je prikazana v izpisu 5.

```
#include <time.h>
                           .
                           .
                           .
 clock_t start, finish;
 double duration;
                           .
                           .
                           .
 start = clock();
 qSort(0,99999);
 finish = clock();
 duration = (double)(finish - start)/CLOCKS_PER_SEC;
```

Izpis 5: Merjenje časa urejanja zaporedja

Čas urejanja v spremenljivki `duration` v izpisu 5 merimo v sekundah.

3. Zahteve naloge

Implementirati je potrebno algoritem hitro uredi z mediano, ki bo uredil zaporedje števil dolgo do 1.000.000 podatkov. Števila v zaporedju so lahko naključna, lahko pa so urejena v naraščajočem ali pa padajočem vrstnem redu. Po urejanju je potrebno preveriti pravilnost delovanja in izpisati, ali je zaporedje pravilno ali pa nepravilno urejeno. Izmeriti pa je potrebno tudi čas delovanja, ki ga prav tako izpišete, ko je urejanje končano.
Ob zagonu programa se mora zagnati meni, ki je prikazan na sliki 13.

```
Hitro uredi - izbira:

1 Generiraj naključno zaporedje
2 Generiraj urejeno naraščajoče zaporedje
3 Generiraj urejeno padajoče zaporedje
4 Izpis zaporedja
5 Uredi
6 Konec

Vaša izbira:
```

Slika 13: Začetni meni, ob zagonu aplikacije

Ob izbiri menijske postavke za generiranje katerega od zaporedij, je potrebno od uporabnika zahtevati vpis dolžine zaporedja, nato pa v polje vnesti ustrezna števila. Program se konča, ko uporabnik izbere menijsko postavko *Konec*.