Zadatak 3. (8 bodova)

Odredite gornju granicu (veliki O) složenosti i asimptotsku složenost zadanih odsječaka uz pretpostavku konstantne asimptotske složenosti funkcije f(x) i da je n >> 1. Obrazložite odgovor.

a)
```
a=0;
for(i=0;i<n;i++)
        for(j=n;j>i;j--)
                a+=f(j);
```

O:	asimptotska:

b)
```
a=0;
for(i=n; i>1; i=(int)sqrt(i))
        for(j=1;j<i;j*=2)
                a+=f(j);
```

O:	asimptotska:

Zadatak 3. (8)

a) broj iteracija je n + (n – 1) + (n – 2) + ... + 1, dakle:
$O(n^2)$;
$\Theta((n+1)*n/2)$

b) unutarnja petlja je logaritamske složenosti prema varijabli i, a i poprima po vrijednosti n, $\sqrt{n}$, $\sqrt{\sqrt{n}}$,... dakle:
$\Theta(\log_2 n^{1/2} + \log_2 n^{1/4} + \log_2 n^{1/8} + ...) = \Theta((1 + ½ + ¼ + ...) * \log_2 n)) \sim= \Theta(2 \log_2 n)$
tj. $O(\log_2 n)$;

Zadatak 3. (12 bodova)

a) Koja je apriorna složenost zbrajanja dvaju n-znamenkastih brojeva na papiru?

b) Poredajte uzlazno sljedeće složenosti (nakon što su poredani, među izrazima mora biti znak >,< ili =):
$O(2^n)$, $O(n^{1.5})$, $O(n)$, $O(n \log_2 n)$, $O(n \ln n)$, $O(\sqrt{n})$, $O(n!)$, $O(n^n)$, $O(n^{1000})$, $O(3^n)$

c) Koja je apriorna složenost posljednjih triju linija sljedećeg programskog odsječka:

```
double n;
/* ... odsječak u kojem se n postavlja
na neku vrijednost ...*/
while (n>1) {
        n*=0.999;
}
```

Zadatak 3. (12)

a) $O(n)$

b) $O(\sqrt{n}) < O(n) < O(n \log_2 n) = O(n \ln n) < O(n^{1.5}) < O(n^{1000}) < O(2^n) < O(3^n) < O(n!) < O(n^n)$

c) $O(\log n)$

Zadatak 4. (10 bodova)

Odredite asimptotsku složenost sljedećih algoritama:

a)
```
s=0;
for(i=1;i<=n;i*=2)
        for(j=1;j<=n;j++)
                s+=radi(n,j)+radi(n,i);
```

b)
```
s=0;
for(i=1;i<=n;i*=2)
        for(j=1;j<=n;j++)
                s+=radi(j,n)+radi(j,i);
```

uz pretpostavku da je asimptotska složenost funkcije radi(x,y) jednaka Θ(radi(x,y))=x. Obrazložite odgovor.

Zad4.

a) $2n^2\log_2 n$ b) $n(n+1)\log_2 n$

Zadatak 3. (5 bodova)

a) Napišite bilo kakvu funkciju vremenske složenosti $O(2^n)$. **n** neka bude neki cjelobrojni ulazni argument.
b) Kolika je vremenska složenost izračunavanja faktorijela nekog broja **n** u O notaciji?
c) Kolika je vremenska složenost pronalaska svih permutacija nekog niza u O notaciji (u ovisnosti o **n**, duljini niza)?

3. zadatak

a) Npr.

```
#include <math.h>
void func(int n){
        int pon=pow(2,n)
        for (i=0;i<pon;i++);
}
```

Česta valjana alternativna rješenja

```
int func(int n){
        return func(n-1)+ func(n-1);
}

int func(int n){    //fibonnaci
        return func(n-1)+ func(n-2);
}
```

b) O(n)
c) O(n!)

6. zadatak (8 bodova)

a) (4 boda) Odredite apriornu složenost programskog odsječka i detaljno obrazložite odgovor. O kojem algoritmu je riječ?

```
void StoJaRadim (int A [], int N) {
  int i, j;
  int pom;
  for (i = 1; i < N; i++) {
    pom = A[i];
    for (j = i; j >= 1 && A[j-1] > pom; j--)
      A[j] = A[j-1];
    A[j] = pom;
  }
}
```

b) (4 boda) Odredite apriornu složenost programskih odsječaka i detaljno obrazložite odgovore:

```
1) int nekaFunkcija(int n) {
     int br = 0;
     int nasao = 0;
     do {
       br++;
       if (br == korak)
         nasao = 1;
       n /= 2;
     } while (n > 0);
     return br;
   }
```

```
2)   nasao = 0;
   for (double i=0; i<n; i++)
     for (double j=0; j<n-i; j++)
       if (i==j && a[i][j]==trazeni)
         nasao = 1;
```

Zadatak 6. (8 bodova)

a) (4 boda)
$O(n^2)$
Ovo je sortiranje umetanjem („Insertion sort"). Postoje dva dijela ulaznog statičkog polja A: sortirani i nesortirani. U svakom koraku algoritma sortirani dio se proširuje tako da se u njega na ispravno mjesto ubaci prvi element iz nesortiranog dijela polja. Vanjska petlja služi za određivanje granice sortiranog dijela, dok unutarnja ubacuje element u sortirani niz i pomiče ostale elemente. Složenost i u najgorem slučaju (kad je niz sortiran naopako) je $O(n^2)$.

b) (2 boda)
$O(\log_2 n)$
Koristi se do-while petlja i provjeravanje uvjeta (if naredba) nema utjecaja na duljinu izvođenja algoritma. Algoritam uzastopce cjelobrojno dijeli ulazni broj s 2 dok god je rezultat veći od 0. Dakle, petlja će se izvršiti onoliko puta koliko je n potencija broja 2, odnosno točnije, $\log_2 n + 1$ puta. Naredbe prije i poslije do-while petlje imaju konstantu složenost.

c) (2 boda)
$O(n^2)$
Kroz petlje se prolazi uvijek, bez obzira je li uvjet unutar petlje zadovoljen ili ne, jer nema break naredbe. Broj izvršavanja 1 (n=1), 55 (n=10), 5050 (n=100), 500500 (n=1000), itd. Točniji opis složenosti je $O(½\ n^2)$.

Zadatak 5. (4 boda)

Zadano je polje brojeva s elementima: **91, 7, 8, 20, 5, 18, 4, 9**. Ilustrirajte uzlazno sortiranje zadanog niza brojeva (ispišite polje nakon svake promjene i potcrtajte sve brojeve relevantne za sljedeći korak) algoritmom ***shellsort*** **uz korake k = 4, 2, 1**

5. (4 boda)

Korak = 4							
91	7	8	20	5	18	4	9
5	7	8	20	91	18	4	9
5	7	8	20	91	18	4	9
5	7	4	20	91	18	8	9
5	7	4	9	91	18	8	20

Korak = 2							
5	7	4	9	91	18	8	20
4	7	5	9	91	18	8	20
4	7	5	9	91	18	8	20
4	7	5	9	91	18	8	20
4	7	5	9	91	18	8	20
4	7	5	9	8	18	91	20
4	7	5	9	8	18	91	20

Korak = 1							
4	7	5	9	8	18	91	20
4	7	5	9	8	18	91	20
4	5	7	9	8	18	91	20
4	5	7	9	8	18	91	20
4	5	7	8	9	18	91	20
4	5	7	8	9	18	91	20
4	5	7	8	9	18	91	20
4	5	7	8	9	18	20	91

Zadatak 4. (10 bodova)

U polje cjelobrojnih brojeva pohranjen je sljedeći niz brojeva:

4, 6, 7, 2, 9, 8, 1, 5, 3

a) **(5 bodova)** Ilustrirajte uzlazno sortiranje algoritmom mergesort. Potrebno je prikazati sadržaj polja nakon svake promjene.Pri particioniranju neparnog broja elemenata neka bude desna particija veća. Redosljed izmjena je bitan: nije dopušteno paralelno izvršavanje operacija, a lijeve particije se obrađuju prije desnih (mergesort je implementiran rekurzivnom funkcijom koja radi u jednoj dretvi odnosno *thread-u*).

b) **(5 bodova)** Ilustrirajte uzlazno sortiranje algoritmom quicksort. Stožer za quicksort birajte metodom aproksimacije medijana temeljem prvog, srednjeg i zadnjeg člana, a vrijedi da je `cutoff = 3`, nakon čega se `sortira bez navođenja koraka`.

Zadatak 4. (10 bodova)

a) (5 bodova)

4	6	**7**	**2**	9	8	1	5	3	
4	**6**	**2**	**7**	9	8	1	5	3	prva izmjena: desna potparticija lijeve particije
2	4	6	7	**9**	**8**	1	5	3	nakon merge-a lijevih particija
2	4	6	7	8	9	1	**5**	**3**	kod desne particije prvo se složi lijeva (manja) potparticija
2	4	6	7	**8**	**9**	**1**	**3**	**5**	zatim desna potparticija (od desne particije)
2	**4**	**6**	**7**	**1**	**3**	**5**	**8**	**9**	pa i pri merge-anju desne ima izmjena
1	2	3	4	5	6	7	8	9	i konačni merge

b)(5 bodova)

4	6	7	2	**9**	8	1	5	**3**	početno
3	6	7	2	**4**	8	1	**5**	9	stožer odabran
3	**6**	7	2	5	8	**1**	4	9	stožer sklonjen
3	1	**7**	**2**	5	8	6	4	9	i=1, j=6 zamijenjeni
3	1	2	**7**	5	8	6	**4**	9	i=2, j=3 zamijenjeni
3	**1**	**2**	4	5	8	6	7	9	i,j se mimoišli, stožer vraćen
1	2	3	4	5	8	**6**	**7**	9	lijevo riješeno insertionom
1	2	3	4	5	**8**	7	**6**	9	stožer desne particije sklonjen
1	2	3	4	5	**6**	7	**8**	9	zamjena u desnom dijelu, zadnja

Zadatak 5. (10 bodova)

U polje cijelih brojeva pohranjen je sljedeći niz brojeva:

5, 9, 2, 1, 8, 6, 7, 3, 4

a) **(5 bodova)** Ilustrirati uzlazno sortiranje algoritmom Bubblesort.

b) **(5 bodova)** Ilustrirati uzlazno sortiranje algoritmom Quicksort. Stožer za quicksort odabrati metodom aproksimacije medijana temeljem prvog, središnjeg i zadnjeg člana. Polja s manje ili točno `cutoff = 3` elementa sortirati Insertion sortom.

Zadatak 5. (10)

a)

5 **9** **2** 1 8 6 7 3 4
5 2 **9** **1** 8 6 7 3 4
5 2 1 **9** **8** 6 7 3 4
5 2 1 8 **9** **6** 7 3 4
5 2 1 8 6 **9** **7** 3 4
5 2 1 8 6 7 **9** **3** 4
5 2 1 8 6 7 3 **9** **4**
5 **2** 1 8 6 7 3 4 9
2 **5** **1** 8 6 7 3 4 9
2 1 5 **8** **6** 7 3 4 9
2 1 5 6 **8** **7** 3 4 9
2 1 5 6 7 **8** **3** 4 9
2 1 5 6 7 3 **8** **4** 9
2 **1** 5 6 7 3 4 8 9
1 2 5 6 **7** **3** 4 8 9
1 2 5 6 3 **7** **4** 8 9
1 2 5 **6** **3** 4 7 8 9
1 2 5 3 **6** **4** 7 8 9
1 2 **5** **3** 4 6 7 8 9
1 2 3 **5** **4** 6 7 8 9
1 2 3 4 5 6 7 8 9

b)

5 9 2 1 **8** 6 7 3 **4**
4 9 2 1 **5** 6 7 **3** 8
4 **9** 2 1 **3** 6 7 5 8
4 3 2 1 **9** 6 7 **5** 8
4 **3** 2 **1** 5 6 7 9 8
1 **3** **2** 4 5 6 7 9 8
1 2 3 4 5 **6** **7** 9 **8**
1 2 3 4 5 6 **7** **9** 8
1 2 3 4 5 6 **9** **7** 8
1 2 3 4 5 6 7 **9** **8**
1 2 3 4 5 6 7 8 9

Zadatak 5. (10 bodova)

U polje cijelih brojeva pohranjen je sljedeći niz brojeva:

16, 20, 6, 22, 5, 8, 17, 21, 23, 3, 2, 19.

a) Ilustrirajte (napišite sadržaj polja nakon svake promjene) stvaranje gomile s relacijom ***veći od*** (max heap) od zadanog polja brojeva algoritmom čija je složenost za najgori slučaj *O(n)*.

b) Počevši od gomile kreirane u podzadatku a), prikažite sortiranje zadanog niza *heapsortom*, prikazujući svaki korak sortiranja (napišite sadržaj polja nakon svake promjene).

Zadatak 5. (10)

16	20	6	22	5	**8**	17	21	23	3	2	**19**
16	20	6	**22**	5	19	17	21	**23**	3	2	8
16	20	**6**	23	5	**19**	17	21	22	3	2	8
16	20	19	23	5	**6**	17	21	22	3	2	**8**
16	**20**	19	**23**	5	8	17	21	22	3	2	6
16	23	19	**20**	5	8	17	21	**22**	3	2	6
16	**23**	19	22	5	8	17	21	20	3	2	6
23	**16**	19	**22**	5	8	17	21	20	3	2	6
23	22	19	**16**	5	8	17	**21**	20	3	2	6
23	22	19	21	5	8	17	16	20	3	2	6

23	22	19	21	5	8	17	16	20	3	2	6
6	22	19	21	5	8	17	16	20	3	2	**23**
22	6	19	21	5	8	17	16	20	3	2	**23**
22	21	19	6	5	8	17	16	20	3	2	**23**
22	21	19	20	5	8	17	16	6	3	2	**23**
2	21	19	20	5	8	17	16	6	3	**22**	**23**
21	2	19	20	5	8	17	16	6	3	**22**	**23**
21	20	19	2	5	8	17	16	6	3	**22**	**23**
21	20	19	16	5	8	17	2	6	3	**22**	**23**
3	20	19	16	5	8	17	2	6	**21**	**22**	**23**
20	3	19	16	5	8	17	2	6	**21**	**22**	**23**
20	16	19	3	5	8	17	2	6	**21**	**22**	**23**
20	16	19	6	5	8	17	2	3	**21**	**22**	**23**
3	16	19	6	5	8	17	2	**20**	**21**	**22**	**23**
19	16	3	6	5	8	17	2	**20**	**21**	**22**	**23**
19	16	17	6	5	8	3	2	**20**	**21**	**22**	**23**
2	16	17	6	5	8	3	**19**	**20**	**21**	**22**	**23**
17	16	2	6	5	8	3	**19**	**20**	**21**	**22**	**23**
17	16	8	6	5	2	3	**19**	**20**	**21**	**22**	**23**
3	16	8	6	5	2	**17**	**19**	**20**	**21**	**22**	**23**
16	3	8	6	5	2	**17**	**19**	**20**	**21**	**22**	**23**
16	6	8	3	5	2	**17**	**19**	**20**	**21**	**22**	**23**
8	6	2	3	5	**16**	**17**	**19**	**20**	**21**	**22**	**23**
5	6	2	3	**8**	**16**	**17**	**19**	**20**	**21**	**22**	**23**
6	5	2	3	**8**	**16**	**17**	**19**	**20**	**21**	**22**	**23**
3	5	2	**6**	**8**	**16**	**17**	**19**	**20**	**21**	**22**	**23**
5	3	2	**6**	**8**	**16**	**17**	**19**	**20**	**21**	**22**	**23**
2	3	**5**	**6**	**8**	**16**	**17**	**19**	**20**	**21**	**22**	**23**
3	2	**5**	**6**	**8**	**16**	**17**	**19**	**20**	**21**	**22**	**23**
2	**3**	**5**	**6**	**8**	**16**	**17**	**19**	**20**	**21**	**22**	**23**

Zadatak 5. *(5 bodova)*

Neka je definiran tip podatka:

```
typedef struct{
      int prvi;
      int drugi;
} podatak;
```

Zadano je polje tipa **podatak** s elementima:

3	7	2	9	3	2	9	4	8	5
4	3	4	5	1	1	6	7	4	0

Ilustrirajte uzlazno sortiranje zadanog polja po varijabli (atributu) **prvi** (ispišite polje nakon svake promjene i označite sve elemente važne za sljedeći korak):

a) *(2 boda)* algoritmom ***mergesort*** i

b) *(3 boda)* algoritmom ***quicksort***. Stožer za quicksort birajte metodom aproksimacije medijana temeljem prvog, srednjeg i zadnjeg člana.

*Napomena: Varijabla **drugi** se u ovom zadatku nikad ne koristi kao kriterij sortiranja.*

5. **Mergesort** – uzlazno. Više je mogućih, točnih, rješenja - ovisno o izborima. Ovdje je dano rješenje za konzistentno veću lijevu podparticija u slučaju dijeljenja particije neparne veličine, te u slučaju jednakosti ključeva uzimanje elementa iz lijeve particije(ovaj zadnji izbor osigurava stabilnost algoritma). Zadatak za mergesort je moguće riješiti i sortiranjem samo niza vrijednosti u varijabli ***prvi*** i nadopunjavanje dobivenog niza na kraju dodajući adekvatno poredane vrijednosti variabli ***drugi*** koristeći svojstvo stabilnosti mergesorta(za quicksort u sljedećem zadatku to ne možete).

3	7	2	9	3	2	9	4	8	5
4	3	4	5	1	1	6	7	4	0

2	3	7	9	3	2	9	4	8	5
4	4	3	5	1	1	6	7	4	0

2	3	7	3	9	2	9	4	8	5
4	4	3	1	5	1	6	7	4	0

2	3	3	7	9	2	9	4	8	5
4	4	1	3	5	1	6	7	4	0

2	3	3	7	9	2	4	9	8	5
4	4	1	3	5	1	7	6	4	0

2	3	3	7	9	2	4	9	5	8
4	4	1	3	5	1	7	6	0	4

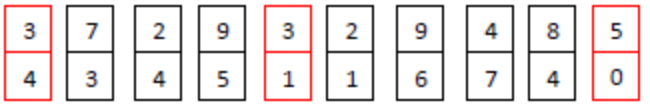
2	3	3	7	9	2	4	5	8	9
4	4	1	3	5	1	7	0	4	6

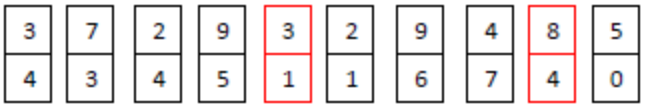
2	2	3	3	4	5	7	8	9	9
4	1	4	1	7	0	3	4	5	6

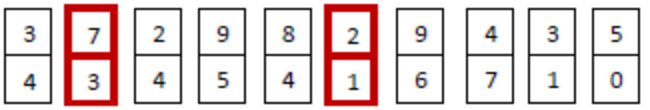
Najčešće greške (redoslijed po učestalosti):

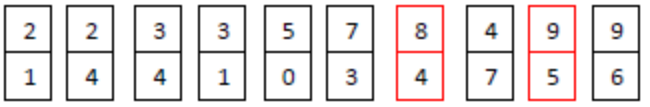
- Sortiranje samo niza vrijednosti u varijabli ***prvi*** bez ikad navođenja pripadnih vrijednosti ***drugi;*** u zadatku je dano polje tipa ***podatak*** koji je strukturni tip i varijable ***prvi*** i ***drugi*** su nerazdvojive, sortiranje smije mijenjati samo redoslijed, ali ne i originalne podatke članova polja. Uz to, vrijednosti varijable ***drugi*** su važne i ne mogu se ignorirati jer nam omogućuju da razlikujemo podatke s identičnim vrijednostima varijable ***prvi.***
- Nekonzistetnost u izvođenju algoritma (pogotovo u pogledu korištenih izbora)
- Sortiranje samo niza vrijednosti varijable prvi te na kraju, u zadnjem dijelu rješenja adekvatno poredane vrijednosti varijable drugi, ali nije se navelo da se pritom koristi svojstvo stabilnosti mergesorta

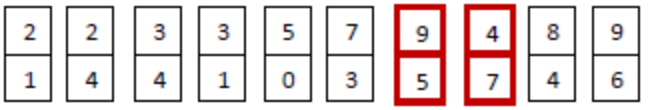
Quicksort – uzlazno. Više je mogućih, točnih, rješenja - ovisno o izborima. Ovdje je dano rješenje za konzistentno biranje stožera lijevo u slučaju nepostojanja jedinstvenog srednjeg indeksa i prebacivanje jednakih elemenata u lijevu particiju. Također je napravljen izbor kod insertion sorta, korištenog za sortiranje particija s manje od 3 elementa, da redoslijed elemenata s istim ključevima sortiranja(prvi) ostaje isti.

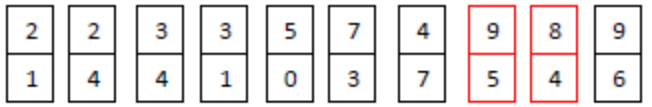


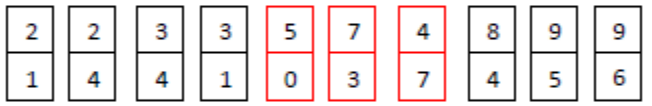


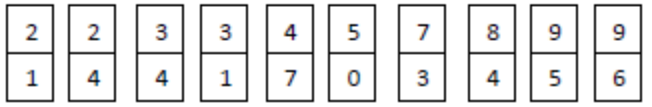












Najčešće greške (redoslijed po učestalosti):

- Sortiranje samo niza vrijednosti u varijabli ***prvi*** bez ikad navođenja pripadnih vrijednosti ***drugi;*** u zadatku je dano polje tipa ***podatak*** koji je strukturni tip i varijable ***prvi*** i ***drugi*** su nerazdvojive, sortiranje smije mijenjati samo redoslijed, ali ne i originalne podatke članova polja. Uz to, vrijednosti varijable ***drugi*** su važne i ne mogu se ignorirati jer nam omogućuju da razlikujemo podatke s identičnim vrijednostima varijable ***prvi.***
- Nekonzistetnost u izvođenju algoritma (pogotovo u pogledu korištenih izbora)
- Preskakanje koraka
- Krivo izvedeno rukovanje stožerom
  - Uopće nema skrivanja stožera
  - Kod skrivanja i vraćanja stožera svi se elementi između pomiču lijevo ili desno, umjesto samo zamjene stožera s nekim drugim elementom
- Krivo preslagivanje elemenata u particije

Zadatak 3. (14 bodova)

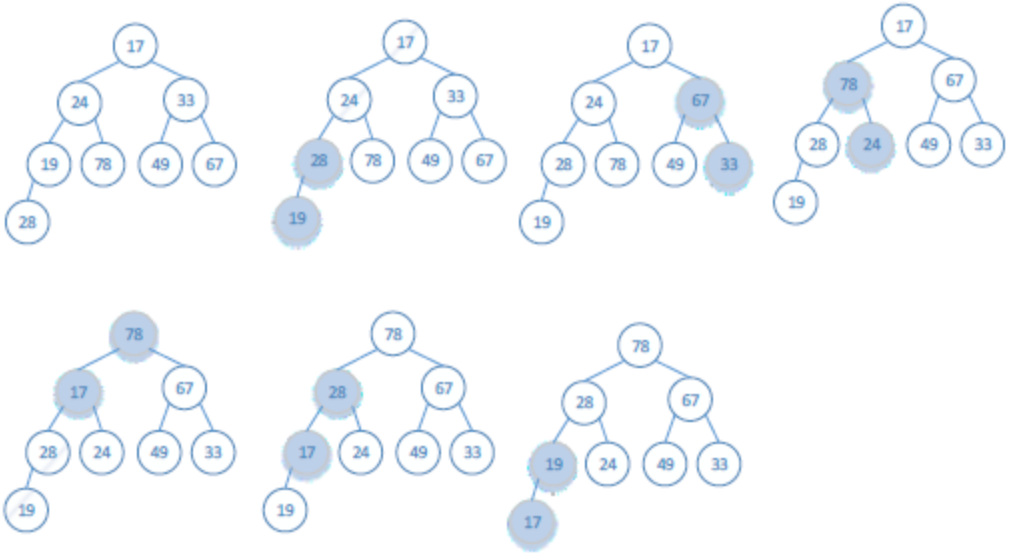
Zadan je niz brojeva: **17, 24, 33, 19, 78, 49, 67, 28**.

a) **(7 bodova)** Ilustrirajte (nacrtajte stablo nakon svake promjene) stvaranje gomile s relacijom ***veći od*** (max heap) od zadanog polja brojeva algoritmom čija je složenost za najgori slučaj *O(n)*.

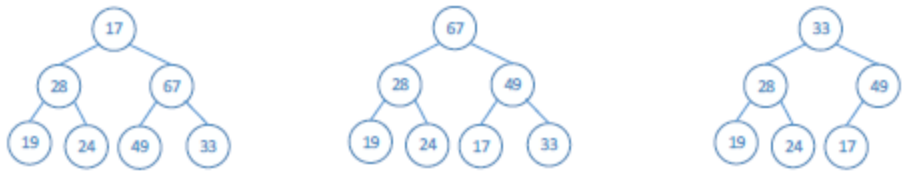
b) **(7 bodova)** Za gomilu iz a) zadatka prikažite postupak uzlaznog *heapsorta*. Prikažite svaki korak sortiranja (nacrtajte stablo nakon svake zamjene dva elementa).

Zadatak 3. (14 bodova)

a)



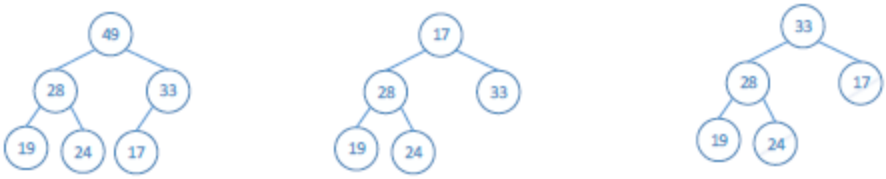
b)



17, 28, 67, 19, 24, 49, 33, 78

67, 28, 49, 19, 24, 17, 33, 78

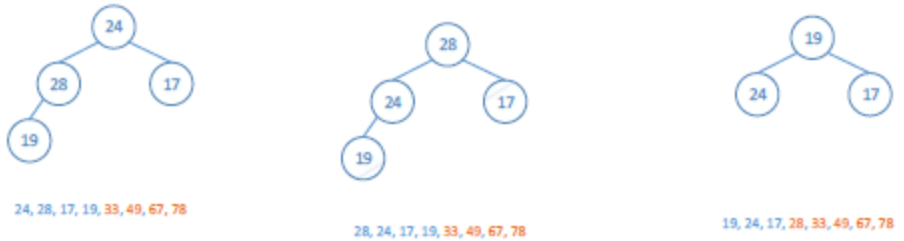
33, 28, 49, 19, 24, 17, 67, 78



49, 28, 33, 19, 24, 17, 67, 78

17, 28, 33, 19, 24, 49, 67, 78

33, 28, 17, 19, 24, 49, 67, 78



24, 28, 17, 19, 33, 49, 67, 78

28, 24, 17, 19, 33, 49, 67, 78

19, 24, 17, 28, 33, 49, 67, 78

24, 19, 17, 28, 33, 49, 67, 78

17, 19, 24, 28, 33, 49, 67, 78

19, 17, 24, 28, 33, 49, 67, 78

17, 19, 24, 28, 33, 49, 67, 78

Zadatak 5. *(5 bodova)*

Zadano je polje brojeva s elementima: **9, 6, 7, 2, 1, 8, 4, 5, 0, 3.** Ilustrirajte uzlazno sortiranje zadanog niza brojeva (ispišite polje nakon svake promjene i označite sve brojeve relevantne za sljedeći korak) algoritmom **mergesort** i algoritmom **shellsort** za niz koraka {4,3,1}.

5. zadatak

1. MergeSort

   (u slučaju uvijek veće ili jednake lijeve polovice)

   9, 6, 7, 2, 1, 8, 4, 5, 0, 3
   6, 9, 7, 2, 1, 8, 4, 5, 0, 3
   6, 7, 9, 2, 1, 8, 4, 5, 0, 3
   6, 7, 9, **1, 2**, 8, 4, 5, 0, 3
   1, 2, 6, 7, 9, 8, 4, 5, 0, 3
   1, 2, 6, 7, 9, **4, 8**, 5, 0, 3
   1, 2, 6, 7, 9, **4, 5, 8**, 0, 3
   1, 2, 6, 7, 9, **0, 3, 4, 5, 8**
   0, 1, 2, 3, 4, 5, 6, 7, 8, 9

   (u slučaju uvijek veće ili jednake desne polovice)

   9, 6, 7, 2, 1, 8, 4, 5, 0, 3
   6, 9, 7, 2, 1, 8, 4, 5, 0, 3
   6, 9, 7, **1, 2**, 8, 4, 5, 0, 3
   6, 9, **1, 2, 7**, 8, 4, 5, 0, 3
   1, 2, 6, 7, 9, 8, 4, 5, 0, 3
   1, 2, 6, 7, 9, **4, 8**, 5, 0, 3
   1, 2, 6, 7, 9, 4, 8, **0, 3, 5**
   1, 2, 6, 7, 9, **0, 3, 4, 5, 8**
   0, 1, 2, 3, 4, 5, 6, 7, 8, 9

2. Shell sort {4,3,1}

   9, 6, 7, 2, 1, 8, 4, 5, 0, 3
   1, 6, 7, 2, **9**, 8, 4, 5, 0, 3 t=4
   1, 6, **4**, 2, 9, 8, **7**, 5, 0, 3
   0, 6, 4, 2, **1**, 8, 7, 5, **9**, 3
   0, **3**, 4, 2, 1, **6**, 7, 5, 9, **8**
   0, **1**, 4, 2, **3**, 6, 7, 5, 9, 8 t=3
   0, 1, **2**, **4**, 3, 6, 7, 5, 9, 8 t=1
   0, 1, 2, **3**, **4**, 6, 7, 5, 9, 8
   0, 1, 2, 3, 4, **5**, **6**, **7**, 9, 8
   0, 1, 2, 3, 4, 5, 6, 7, **8**, **9**

5. zadatak (18 bodova)

Napisati funkciju `void merge (char *prva, char *druga, char *treca)` koja će od podataka iz sortiranih datoteka `prva` i `druga` stvoriti sortiranu datoteku `treca`. Svaki redak datoteka sadrži jedan znakovni niz maksimalne duljine 100 znakova. Ne treba kontrolirati je li otvaranje ili stvaranje datoteka uspjelo.

5. (14 bodova)

```
void merge(char *prva, char *druga, char *treca){
	FILE *prvaD, *drugaD, *trecaD;
	int imaPrva = 1, imaDruga = 1;
	char redakPrva[ROW_MAX_LEN + 1] = {'\0'};
	char redakDruga[ROW_MAX_LEN + 1] = {'\0'};

	prvaD = fopen(prva, "r");
	drugaD = fopen(druga, "r");
	trecaD = fopen(treca, "w");

	//Procitaj redak iz prve
	if(fscanf(prvaD, "%s\n", redakPrva) == EOF){
		//Prva datoteka je prazna
		imaPrva = 0;
	}
	if(fscanf(drugaD, "%s\n", redakDruga) == EOF){
		//Druga datoteka je prazna
		imaDruga = 0;
	}

	do{
		if(imaPrva && imaDruga){
			if(strcmp(redakPrva, redakDruga) < 0){
				fprintf(trecaD, "%s\n", redakPrva);
				if(fscanf(prvaD, "%s\n", redakPrva) == EOF){
					imaPrva = 0;
				}
			}else if(strcmp(redakPrva, redakDruga) > 0){
				fprintf(trecaD, "%s\n", redakDruga);
				if(fscanf(drugaD, "%s\n", redakDruga) == EOF){
					imaDruga = 0;
				}
			}else{
				fprintf(trecaD, "%s\n", redakPrva);
				fprintf(trecaD, "%s\n", redakDruga);
				if(fscanf(prvaD, "%s\n", redakPrva) == EOF){
					imaPrva = 0;
				}
				if(fscanf(drugaD, "%s\n", redakDruga) == EOF){
					imaDruga = 0;
				}
			}
		}else if(imaPrva){
			fprintf(trecaD, "%s\n", redakPrva);
			if(fscanf(prvaD, "%s\n", redakPrva) == EOF){
				imaPrva = 0;
			}
		}else if(imaDruga){
			fprintf(trecaD, "%s\n", redakDruga);
			if(fscanf(drugaD, "%s\n", redakDruga) == EOF){
				imaDruga = 0;
			}
		}else{
			break;
		}
	} while(1);

	fclose(prvaD);
	fclose(drugaD);
	fclose(trecaD);
}
```

Zadatak 2. (5 bodova)

Kvazi-binomni koeficijenti K(n,m) su definirani sa

$$K(n, m) = K(n-1, m-1) + m \cdot K(n-1, m+1) ;$$
$$K(n, n) = K(n, 0) = 1 .$$

Ako je $n < 0$ ili $m > n$, onda je $K(n, m) = 0$.

Napišite rekurzivnu funkciju `kvazi_binomni` koja će za bilo koji `n` i `m` izračunati vrijednost kvazi-binomnog koeficijenta `K(n,m)`.

2. (5 bodova)

```
int kvazi_binomni(int n, int m) {
      if( ( n < 0 ) || ( m > n ) ) return 0;
      if( ( m == 0 ) || ( n == m ) ) return 1;
      return kvazi_binomni( n - 1, m - 1 )
            + m * kvazi_binomni( n - 1, m + 1 );
}
```

Zadatak 3. (5 bodova)

Svaki prirodni broj $N \geq 2$ može se napisati u obliku rastava na proste faktore:

$$N=p_1^{n1} \cdot p_2^{n2} \cdots p_k^{nk},$$

gdje su p_i, $i=1,\ldots,k$, različiti prosti brojevi (prosti faktori), a eksponenti $n_1,\ldots,n_k$ su prirodni brojevi.

Napišite rekurzivnu funkciju `najveci_eksponent` koja će za zadani broj `N` odrediti najveći eksponent s kojim se neki prosti broj pojavljuje u rastavu broja `N` na proste faktore (najveći od brojeva n_1). Prototip funkcije je:

```
int najveci_eksponent (int n, int m);
```

pri čemu je `m` trenutačni djelitelj (p_1). Početni poziv funkcije za zadani `n` je oblika `najveci_eksponent(n,2)`.

Primjeri:

Za `N` = 6, funkcija treba vratiti 1 jer $6 = 2^1 \cdot 3^1$.
Za `N` = 8, funkcija treba vratiti 3 jer $8 = 2^3$.
Za `N` = 36, funkcija treba vratiti 2 jer $36 = 2^2 \cdot 3^2$.

3. (5 bodova)

```
int najveci_eksponent(int n, int m){
      int dalje, eksponent=0;
      if (n<m) return 0; /* Osnovni slucaj */
      while(n % m == 0){   /* Koliko puta m dijeli n */
            eksponent++;
            n /= m;
      }
      dalje= najveci_eksponent (n,m+1); /* Rekurzivno napredovanje */

      if (dalje>eksponent) return dalje; /* Veci se vraca pozivajucoj proceduri */
      return eksponent;
}
```

Zadatak 2. (5 bodova)

Napišite rekurzivnu funkciju prototipa:

```
int prebroji_znamenke(int broj, int znamenka);
```

koja će odrediti učestalost (broj pojavljivanja) zadane znamenke u zadanom broju.

Napišite glavni program koji od korisnika (putem tipkovnice) prima broj i traženu znamenku, poziva funkciju `prebroji_znamenke` i ispisuje dobiveni rezultat.

Napomena: Nerekurzivno rješenje neće se priznavati.

2.

```
int prebroji_znamenke(int broj, int znamenka)
{
   //Negativni broj pretvori u pozitivan
   if (broj < 0) broj *= -1;

   //Sve znamenke do zadnje
   if (broj > 10)
   {
      if (broj % 10 == znamenka) return 1 + prebroji_znamenke(broj/10, znamenka);
      else return prebroji_znamenke(broj/10, znamenka);
   }

   //Zadnja znamenka - uvjet završetka - funkcionira i za ulaz (0,0)
   else
   {
      if (broj == znamenka) return 1;
      else return 0;
   }
}


int main(int argc, char *argv[])
{
  int broj, znamenka;

  printf("\nUnesite broj: ");
  scanf("%d", &broj);

  printf("\nUnesite znamenku: ");
  scanf("%d", &znamenka);

  printf("\nZnamenka %d pojavljuje se %d puta u broju %d.", znamenka,
           prebroji_znamenke(broj, znamenka), broj);

  return 0;
      }
```

Najčešće greške (redoslijed po učestalosti):

- Ne uzimanje u obzir negativnih brojeva
- Loš uvjet terminiranja koji ne provjerava za slučaj ulaza (0,0)
- Krivi rekurzivni poziv
- Neprihvaćanje i neobrađivanje vraćene vrijednosti iz rekurzivnog poziva

Zadatak 3. *(5 bodova)*

Neka je zadano polje **a** koje se sastoji od **n** cijelih brojeva. Napišite rekurzivnu funkciju `maxRekurzivno` prototipa:

```
int maxRekurzivno(int a[],int n, int *brojPojavljivanja)
```

koja će pronaći najveći element u polju **a** te njegovu učestalost (koliko puta se taj element pojavljuje u **a**).

Napomena: Nerekurzivno rješenje neće se priznavati.

3.

```
int maxRekurzivno(int a[],int n, int *brojPojavljivanja)
{
	int kandidat;
	if (n==1){
		*brojPojavljivanja=1;
		return a[0];
	}
	kandidat=maxRekurzivno(a,n-1, brojPojavljivanja);
	if (kandidat>a[n-1]){
		return kandidat;
	}
	else if(kandidat==a[n-1]){
		(*brojPojavljivanja)++;
		return kandidat;
	}
	else{
		*brojPojavljivanja=1;
		return a[n-1];
	}
}
```

Najčešće greške (redoslijed po učestalosti):

- Loš uvjet terminiranja koji vraća 0 što u slučaju polja koji sadrži samo negativne brojeve daje krivo rješenje
- Krivi rad s pokazivačima
- Neprihvaćanje i neobrađivanje povratne vrijednosti rekurzivnog poziva
- Nepostavljanje brojača `*brojPojavljivanja` na 1 u slučaju da smo naišli na veći broj od dosadašnjeg maksimuma
- Rekurzivni poziv izveden na način koji nije sintaksno točan u C-u (?!)
  - Nedostaje ime funkcije, navedene samo zagrade s parametrima
- Mijenjanje sadržaja ulaznog polja
- Pisanje nedohvatljivog koda (npr. iza return naredbe)

Zadatak 1. (11 bodova)

Jedan od načina računanja binomnih koeficijenta $\binom{n}{k}$, n≥0, k≥0, n≥k, je korištenjem rekurzivne formule $\binom{n}{k}=\binom{n-1}{k-1}+\binom{n-1}{k}$, s time da je $\binom{n}{0}=\binom{n}{n}=1$. Napišite rekurzivnu funkciju `binKoef` koja izračunava binomni koeficijent $\binom{n}{k}$, a kao argumente prima nenegativne cijele brojeve n i k.

Zad 1.

```
int binKoef(int n, int k)
{
	if (k==0) return 1;
	if (k==n) return 1;
	return binKoef(n-1,k-1)+binKoef(n-1,k);
}
```

Zadatak 2. (14 bodova)

Neka je zadano polje **a** koje se sastoji od **n** pozitivnih cijelih brojeva sortiranih silazno. Napišite rekurzivnu funkciju **`postojiZbroj`** koja će za zadani cijeli broj **m** vratiti 1 ako je **m** moguće napisati kao zbroj elemenata polja **a**, odnosno 0 ako to nije moguće. Podrazumijeva se da se elementi polja **a** mogu upotrijebiti samo po jednom.
Prototip funkcije **`postojiZbroj`** treba biti

```
int postojiZbroj(int a[],int n, int m);
```

Zad 2.

```
int postojiZbroj(int a[],int n, int m){
	int i, noviM;

	if(m<0) return 0;
	if (m==0) return 1;

	for(i=n-1; i>=0; i--){
		noviM = m - a[i];
		if (postojiZbroj(a,i,noviM)) return 1;
	}
	return 0;
}
```

Zadatak 2. (5 bodova)

a) Napišite rekurzivnu funkciju koja će tehnikom raspolavljanja izračunati i vratiti sumu članova polja.
b) Napišite glavni program u kojem ćete učitati elemente polja s tipkovnice. Broj elemenata polja nije unaprijed poznat, a učitavanje se prekida kada se učita broj 0. Za pohranu elemenata potrebno je koristiti dinamički alocirano polje, a elementi se smiju učitati samo jednom.

Napomena: Nerekurzivno rješenje neće se priznavati.

2. zadatak

```
int suma(int a[], int lijevo, int desno) {
        int sred = 0;
        if(lijevo > desno) {
                return 0;
        } else if(lijevo == desno){
                return a[lijevo];
        } else {
                sred = ((desno - lijevo) / 2) + lijevo;
                return a[sred] + suma(a,lijevo,sred-1) + suma(a,sred+1,desno);
        }
}
```

ili

```
int suma (int a[], int duljina){
    if (duljina == 0) {
        return 0;
    }
    if (duljina == 1) {
        return a[0];
    }
    return suma (a, duljina/2) + suma (a+duljina/2, duljina-duljina/2);
}

int main(){
        int *polje = NULL, element, cnt = 0, s = 0;

        do {
                printf("Unesite %d. element: ", cnt + 1);
                scanf("%d", &element);
                if(element != 0){
                        polje = (int*)realloc(polje, sizeof(int) * (cnt + 1));
                        polje[cnt] = element;
                        cnt++;
                }
        } while(element != 0);

        s = suma(polje, 0, cnt - 1);
        printf("Suma clanova polja je %d", s);
        return 0;
}
```

2. zadatak (18 bodova)

a) Napisati funkciju `strcatr`, rekurzivnu inačicu funkcije `strcat`. U svakom rekurzivnom pozivu potrebno je konkatenirati po jedan znak. Možete pretpostaviti da je odredišni niz znakova (destination) dovoljne veličine da se u njega pohrani i znakovni niz koji se konkatenira (source).
Napomena: Nerekurzivno rješenje neće se priznavati.

b) Napisati rekurzivnu funkciju `spoji` koja će s tipkovnice učitavati nizove znakova (maksimalne duljine 20 znakova) sve dok se ne učita niz `"kraj"`. Funkcija treba vratiti dva niza znakova:
1. Niz znakova koji se sastoji od učitanih **brojeva** međusobno odvojenih razmakom
2. Niz znakova koji se sastoji od učitanih **riječi** međusobno odvojenih razmakom

Za konkatenaciju nizova znakova potrebno je koristiti funkciju iz **a)** dijela zadatka. Možete pretpostaviti da se riječi sastoje samo od malih slova engleske abecede.
Nije dozvoljeno unaprijed rezervirati memoriju za nizove koje funkcija mora vratiti.

Primjer : za ulazne nizove znakova `"prvi" "111" "222" "drugi" "333" "treci"` funkcija treba vratiti nizove znakova `"111 222 333 "i"prvi drugi treci "`

Napomena: Nerekurzivno rješenje neće se priznavati

c) Napisati glavni program u kojem ćete pozvati funkciju iz **b)** dijela zadatka (funkciju `spoji`)

Zad 2. (18 bodova)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

a) (5 bodova)

```c
char *strcatr (char *destination, char *source){
	int duljina = 0;
	if(strlen(source) == 0){
		return destination;
	}else{
		duljina = strlen(destination);
		destination[duljina] = source[0];
		destination[duljina + 1] = '\0';
		return strcatr(destination, source + 1);
	}
}
```

b) (10 bodova)

```c
#define MAX 20

void spoji(char **brojevi, char **rijeci){
	char buff[MAX+1];
	int duljina;
	//Ucitaj niz sa tipkovnice
	scanf("%s", buff);
	if(strcmp(buff, "kraj") == 0){
		return;
	}
	if(buff[0] >= '0' && buff[0] <= '9'){
		//Uneseni niz je broj
		//Realociramo memoriju
		//trenunta duljina niza + duljina unesenog niza + 1 za razmak + 1 za '\0'
		*brojevi = (char*)realloc(*brojevi, strlen(*brojevi) + strlen(buff) + 2);
		strcatr(*brojevi, buff);
		strcatr(*brojevi, " ");
	}else{
		//Uneseni niz je rijec
		//Realociramo memoriju
		//trenunta duljina niza + duljina unesenog niza + 1 za razmak + 1 za '\0'
		*rijeci = (char*)realloc(*rijeci, strlen(*rijeci) + strlen(buff) + 2);
		strcatr(*rijeci, buff);
		strcatr(*rijeci, " ");
	}
	spoji(brojevi, rijeci);
}
```

c) (3 boda)

```c
int main(){
	char *brojevi = NULL;
	char *rijeci = NULL;

	brojevi = (char*)malloc(sizeof(char));
	rijeci = (char*)malloc(sizeof(char));
	brojevi[0] = '\0';
	rijeci[0] = '\0';

	spoji(&brojevi, &rijeci);
	return 0;
}
```

Zadatak 1. (7 bodova)

Svaki zapis datoteke `podaci.dat` organizirane po načelu raspršenog adresiranja sadrži podatke o jednom studentu i njegovoj ocjeni na kolegiju Algoritmi i strukture podataka: *šifru studenta (znakovni niz duljine 10 znakova), ime studenta (duljine do 20 znakova), prezime studenta (duljine do 30 znakova)* te *ocjenu (`int`)*.

Šifra nula (`"0"`) označava prazan zapis. Veličina bloka na disku je 4096 B. Očekuje se najviše 10000 zapisa, a kapacitet tablice treba biti 15% veći. Prilikom upisa primjenjuje se metoda cikličkog preljeva. Ključ zapisa je **`sifra`**, a pretvorba ključa u adresu se obavlja već pripremljenom funkcijom prototipa:

```
int adresa (char *sifra);
```

Napišite program koji će odrediti prosječnu ocjenu svih studenata čiji su zapisi završili u preljevu. Pretpostavite da u datoteci nije bilo brisanja (i iskoristite tu pretpostavku u radu s datotekom). Na zaslon je potrebno ispisati podatke za sve studente za koje zapisi nisu završili u preljevu, a kojima je ocjena veća ili jednaka prosječnoj ocjeni studenata čiji su zapisi završili u preljevu.

Primjer: Neka je prosječna ocjena `3,64`. Student `Ivan Novak` ima ocjenu `4` i šifru `0036123456`, a njegov zapis ne nalazi se u preljevu. Funkcija na zaslon mora ispitati njegove podatke u sljedećem formatu (prva linija označava znakovna mjesta pri ispisu):

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
0036123456 Novak                           Ivan                4
```

1. (7 bodova)

```c
#include <stdio.h>
#include <string.h>

#define N 10000
#define BLOK 4096
#define C BLOK/sizeof (zapis)
#define M (int)(N * 1.15/C)

typedef struct {
  char sifra[10 + 1];
  char ime[20 + 1];
  char prezime[30 + 1];
  int ocjena;
} zapis;

int adresa(int sifra);

int main() {

  FILE *f = NULL;
  zapis pretinac[C];
  int i, j, sumaOcjena = 0, brStudenata = 0;
  float prosjecnaOcjena = 0;

  f = fopen("podaci.dat", "rb");

  /*Dohvat ocjena iz zapisa koji su zavrsili u preljevu*/
  for(i = 0; i < M; i++) {
    fseek(f, i * BLOK, SEEK_SET);
    fread(pretinac, sizeof(pretinac), 1, f);
    for(j = 0; j < C; j++) {
      if(strcmp(pretinac[j].sifra,"0") == 0) {
        /*Prazan zapis - prelazimo na sljedeci pretinac*/
        break;
      }
      if(adresa(pretinac[j].sifra) != i) {
        /*Preljev*/
        brStudenata++;
        sumaOcjena += pretinac[j].ocjena;
      }
    }
  }

  /*Racunanje prosjecne ocjene*/
  prosjecnaOcjena = (float)sumaOcjena / brStudenata;

  /*Na zaslon ispisujemo podatke za studente cija je ocjena >= prosjecnoj ocjeni*/
  for(i = 0; i < M; i++) {
    fseek(f, i * BLOK, SEEK_SET);
    fread(pretinac, sizeof(pretinac), 1, f);
    for(j = 0; j < C; j++) {
      if(strcmp(pretinac[j].sifra,"0") == 0) {
        /*Prazan zapis - prelazimo na sljedeci pretinac*/
        break;
      }
      if(adresa(pretinac[j].sifra) == i && pretinac[j].ocjena >= prosjecnaOcjena) {
        /*Zapis koji nije preljev*/
        printf("%-11s%-30s%-20s%-9d\n",
          pretinac[j].sifra,
          pretinac[j].prezime,
          pretinac[j].ime,
          pretinac[j].ocjena
        );
      }
    }
  }

  fclose(f);
  return 0;
}
```

Zadatak 5. (15 bodova)

Jedan zapis datoteke organizirane po načelu raspršenog adresiranja sadrži podatke o tjednim promjenama cijena proizvoda: šifru (int), naziv (50+1 znak) i promjenu cijene (float) proizvoda u odnosu na protekli tjedan, te godinu (int) i tjedan u godini (int) na koji se promjena odnosi. Prazni se zapis prepoznaje po šifri jednakoj nula (0). Veličina bloka na disku je 2048 B. Očekuje se najviše 50000 zapisa, a tablica je dimenzionirana za 35% veći kapacitet. Prilikom upisa primjenjuje se metoda cikličkog prelјeva. Poznato je da nakon što se zapis jednom upiše, nema više brisanja zapisa. Ključ zapisa je kombinacija šifre, godine i tjedna, a pretvorba ključa u adresu se obavlja već pripremljenom funkcijom

```
int adresa(int sifra, int godina, int tjedan);
```

Napišite sve potrebne deklaracije (strukturu i parametre za hash).

Napisati funkciju koja će za svaki pretinac odrediti najveću promjenu cijena proizvoda u tom pretincu, ali samo među onim proizvodima koji su u taj pretinac izvorno upućeni (nisu u preljevu) i potom tu ocjenu ispisati u formatu

```
Pretinac broj_pretinca: najveća promjena cijena
```

a ako takva cijena ne postoji, nije potrebno ispisati ništa. Funkcija treba imati prototip:

```
void fun(FILE *f);
```

Zadatak 5. (15 bodova)

```
typedef struct{
       int sifra;
       char naziv[50+1];
       int godina;
       int tjedan;
       float promjena;
} zapis;


void najveca(FILE *f){
  zapis pretinac[C];
  int i, j, adr;
  double najPretinac;
  int imaNesto;

  if (f==NULL) return;

  for (i = 0; i < M; i++) {
    fseek (f, i * BLOK, SEEK_SET);
    fread (pretinac, sizeof (pretinac), 1, f);
    imaNesto=0;
    for (j = 0; j < C; j++) {
      if (pretinac[j].sifra != 0) {
        /* Ako zapis nije prazan */
        if (adresa(pretinac[j].sifra
            , pretinac[j].godina, pretinac[j].tjedan) == i) { // ako je zapis u pravom pretincu ..
          if (imaNesto!=0) { // postoji kandidat za najveću vrijednost
            if(pretinac[j].promjena >najPretinac)
              najPretinac=pretinac[j].promjena;
          }
          else{
            najPretinac=pretinac[j].promjena;
            imaNesto=1;
          }
        }
      }
      else break; // ostatak pretinca je prazan pa ga nije potrebno provjeravati
    }
    if (imaNesto){
      printf("(Pretinac %d) Najveca promjena: %7.2f\n",i,najPretinac);
    }
  }
}
```

Zadatak 1. (20 bodova)

Zadana je datoteka s podatcima o prodajnim artiklima organizirana po načelu raspšenog adresiranja. Svaki zapis se sastoji od šifre artikla (`long`), naziva artikla (do 50 znakova), kategorije (`short`), raspoložive količine (`int`), cijene (`float`) i težine (`float`). Ključ zapisa je šifra, a šifra 0 znači prazan zapis. Pretvorbu ključa u adresu obavlja već pripremljena funkcija prototipa:

```
int adresa( long sifra );
```

Očekuje se do 1.000.000 zapisa, s time da je kapacitet datoteke 10% veći. Veličina bloka na disku je 4096 B. Za preljeve je rezervirano preljevno područje veličine četvrtine primarnog područja. Preljevno područje nalazi se na početku datoteke.

Potrebno je napisati funkciju prototipa

```
stavka *analiza( FILE *fi );
```

koja će napraviti analizu podataka iz datoteke i vratiti tražene informacije. Rezultat analize je polje struktura tipa stavka, pri čemu je stavka zadana odsječkom:

```
typedef struct st_stavka {
       short kategorija;
       float uk_vrijednost;
       float uk_tezina;
} stavka;
```

Funkcija vraća pokazivač na dinamički alocirano polje struktura s onoliko elemenata koliko je kategorija proizvoda u datoteci. Svaki element polja sadrži zbirne podatke za sve proizvode pojedine kategorije. Kategorije u datoteci ne moraju biti uzastopni brojevi, a polje ne mora biti sortirano po kategorijama.

Napomena: polje treba dinamički generirati prema broju kategorija (nije unaprijed poznato koliko je kategorija proizvoda u datoteci).

Zadatak 1. (20)

```
#define BLOK 4096
/* kapacitet pretinca: */
#define C ( BLOK / sizeof( artikl ) )
/* broj pretinaca u primarnom podrucju: */
#define M ( ( int ) ( 1.1 * N / C ) )
/* broj pretinaca u overflow podrucju: */
#define O ( ( int ) ( 0.25 * M ) )

typedef struct st_artikl {
       long sifra;
       char naziv[ 50 + 1 ];
       short kategorija;
       int kolicina;
       float cijena; float tezina;
} artikl;

stavka *analiza( FILE *fi ) {
  int i, j, k, b_stavki = 0, found = 0;
  stavka *stat = NULL, * temp;
  artikl pretinac[ C ];
  fseek( fi, 0L, SEEK_SET );
  for( i = 0; i < M + O; i ++ ) {
    fread( pretinac, sizeof( pretinac ), 1, fi );
    for( j = 0; j < C; j ++ ) {
      if( pretinac[j].sifra == 0 )
        continue;
      for( k = 0; k < b_stavki; k++ ) {
         if( stat[ k ].kategorija == pretinac[j].kategorija ) {
          found = 1;
          break;
        }
      }
      if( !found ) {
        stat = ( stavka * ) realloc( stat , ( b_stavki + 1 ) * sizeof( stavka ) );
        stat[ b_stavki ].kategorija = pretinac[ j ].kategorija;
        stat[ b_stavki ].vrijednost = pretinac[ j ].kolicina * pretinac[ j ].cijena;
        stat[ b_stavki ].uk_tezina = pretinac[ j ].kolicina * pretinac[ j ].tezina;
        b_stavki ++;
      }
      else {
        stat[ k ].vrijednost += pretinac[  j ].kolicina * pretinac[ j ].cijena;
        stat[ k ].uk_tezina += pretinac[ j ].kolicina * pretinac[ j ].tezina;
      }
    }
  }
  return stat;
}
```

Zadatak 1. (20 bodova)

Svaki zapis datoteke organizirane po načelu raspršenog adresiranja sadrži podatke o jednom proizvodu i definiran je strukturom:

```
typedef struct{
   int sifra;
   char naziv[50+1];
   double cijena;
} zapis;
```

Šifra nula (0) označava prazan zapis. Domena šifri je određena tipom podataka, a cijene su garantirano manje od 10^8. Veličina bloka na disku je definirana simboličkom konstantom BLOK. Očekuje se najviše 2500000 zapisa, a kapacitet tablice je 20% veći. Prilikom upisa primjenjuje se metoda cikličkog preljeva. Ključ zapisa je *sifra*, a pretvorba ključa u adresu se obavlja već pripremljenom funkcijom prototipa:

```
int adresa(int sifra);
```

Napišite funkciju `ispis` koja će ispisati šifre, nazive i cijene onih proizvoda koji su završili u preljevu i čija je cijena veća od prosječne cijene svih valjanih zapisa u tom pretincu. Format ispisa je unaprijed zadan primjerom (prvi red označava znamenku desetica, a drugi znamenku jedinica rednog broja kolone znakovnog sučelja):

```
         1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890123
     10000 Crvena kemijska za ispravljanje ispita                   99.90
```

Prototip funkcije zadan je s:

```
void ispis(FILE *fi);
```

Zadatak 1. (20)

```
#define N 250000
#define C (BLOK/sizeof (zapis))
#define M ((int)(N*1.2/C))

void ispis(FILE *fi) {
  zapis pretinac[C];
  int i, j;
  int br_zapisa = 0;
  double suma = 0;
  double prosjecna_cijena = 0;

  for (i = 0; i < M; i++) {
    fseek (fi, i*BLOK, SEEK_SET);
    fread (pretinac, sizeof (pretinac), 1, fi);
    suma = 0; br_zapisa = 0;
    for (j = 0; j < C; j++) {
      if (pretinac[j].sifra) { //preskačemo prazne zapise
        br_zapisa++;
        suma += pretinac[j].cijena;
      }
    }
    prosjecna_cijena = suma / br_zapisa;
    for (j = 0; j < C; j++) {
      /*preskacemo obrisane zapise, zapise koji nisu u preljevu i zapise cijene manje od prosjeka*/
      if(pretinac[j].sifra != 0 && pretinac[j].cijena > suma && adresa(pretinac[j].sifra) != i) {
        printf( "%10d %-50s %11.2f\n", pretinac[j].sifra, pretinac[j].naziv, pretinac[j].cijena);
      }
    }
  }
}
/* primjetiti: bolje je prvo raditi usporedbu cijene i sume (poznato i kratko vrijeme izvodjenja)
nego  pozivati funkciju koja moze i ne mora biti banalne izvedbe i brzog vremena izvodjenja */
```

Zadatak 1. *(5 bodova)*

Jedan zapis datoteke organizirane po načelu raspršenog adresiranja sadrži matični broj studenta (int), ime i prezime (50+1 znak), godinu studija (int) te trenutni prosjek ocjena (float). Prazni se zapis prepoznaje po matičnom broju jednakom nula (0). Veličina bloka na disku je 2048 B. Očekuje se najviše 100000 zapisa, a tablica je dimenzionirana za 35% veći kapacitet. Prilikom upisa primjenjuje se metoda cikličkog preljeva. Ključ zapisa je matični broj, a pretvorba ključa u adresu se obavlja već pripremljenom funkcijom

```
int adresa(int matbr);
```

Napisati funkciju koja će odrediti u koji je pretinac izvorno (prilikom upisa u datoteku) bilo upućeno najviše zapisa i potom vratiti broj zapisa upućenih u taj pretinac, a koji su završili u preljevu. Ako ima više takvih pretinaca vratiti rezultat za bilo koji od njih.
Funkcija treba imati prototip:

```
int fun(FILE *f);
```

Najčešće greške (redoslijed po učestalosti):

- Nepreskakanje u sljedeći pretinac nakon nailaska na prvog praznog u tekućem
- Tretiranje praznog zapisa kao punog
- Krivo shvaćanje zadatka; brojanje zapisa i preljeva u svakom pretincu zasebno i vraćanje pretinca s najviše pripadajućih zapisa u njemu (ili varijacije na temu)

Rješenja

1.

```
#define N 100000
#define BLOK 2048
#define C BLOK/sizeof (zapis)
#define M (int)(N*1.35/C)

typedef struct z {
      int matbr;
      char ipr[50+1];
      int godina;
      float prosjek;
} zapis;

int fun(FILE *f){
      zapis pretinac[C];
      int i, j, zeljeni_pretinac;
      int max_pretinac = 0, max_vrijednost = 0;
      int broj_zapisa_za_pretinac[M] = {0};
      int broj_preljeva_za_pretinac[M] = {0};
      for (i = 0; i < M; i++) {
            fseek (f, i*BLOK, SEEK_SET);
            fread (pretinac, sizeof (pretinac), 1, f);
            for (j = 0; j < C; j++) {
                  if (pretinac[j].sifra != 0) { //provjera je li zapis „pun"
                        zeljeni_pretinac = adresa(pretinac[j].sifra);
                        broj_zapisa_za_pretinac[zeljeni_pretinac]++;
                        if (zeljeni_pretinac != i) {
                              broj_preljeva_za_pretinac[zeljeni_pretinac]++;
                        }
                  }
                  else
                        break; //nakon prvog praznog u pretincu, svi su
                  //ostali prazni
            }
      }
      for (i = 0; i < M; i++) {
            if (broj_zapisa_za_pretinac[i] > max_vrijednost) {
                  max_pretinac = i;
                  max_vrijednost = broj_zapisa_za_pretinac[i];
            }
      }
      return broj_preljeva_za_pretinac[max_pretinac];
}
```

Zadatak 5. (15 bodova)

Svaki zapis datoteke organizirane po načelu raspršenog adresiranja sadrži podatke o jednom proizvodu i definiran je strukturom:

```
typedef struct{
   int sifra;
   char naziv[50+1];
   double cijena;
} zapis;
```

Šifra nula (0) označava prazan zapis. Veličina bloka na disku je 2048 B. Očekuje se najviše 100000 zapisa, a kapacitet tablice je 25% veći. Prilikom upisa primjenjuje se metoda cikličkog preljeva. Ključ zapisa je *sifra*, a pretvorba ključa u adresu se obavlja već pripremljenom funkcijom

```
int adresa(int matbr);
```

Napišite funkciju koja će dinamički alocirati jednodimenzionalno polje koje sadrži šifre onih zapisa koji su završili u preljevu i čija je cijena manja ili jednaka prosječnoj cijeni prozvoda. Funkcija treba vratiti pokazivač na alocirano polje i broj elemenata polja te treba imati prototip:

```
int* fun(FILE *f, int *brZapisa);
```

Zad5.

```
#define N 100000
#define BLOK 2048
#define C BLOK/sizeof (zapis)
#define M (int)(N*1.25/C)

typedef struct {
  int sifra;
  char naziv[50+1];
  double cijena;
} zapis;

int* fun(FILE *f, int *brZapisa){
  zapis pretinac[C];
  int i, j, predvidjeni_pretinac;
  int br_zapisa = 0;
  double suma = 0;
  double prosjek = 0;
  int *polje = NULL;

  //Brojanje zapisa
  for (i = 0; i < M; i++) {
    fseek (f, i*BLOK, SEEK_SET);
    fread (pretinac, sizeof (pretinac), 1, f);
    for (j = 0; j < C; j++) {
      if (pretinac[j].sifra != 0) { //preskačemo prazne zapise
        br_zapisa++;
        suma += pretinac[j].cijena;
      }
      else
        break; //nakon prvog praznog u pretincu, svi su ostali prazni
    }
  }

  //Racunanje prosjeka
  prosjek = (double) suma / br_zapisa;

  //Brojimo samo zapise koji su u preljevu i kojima je cijena <= prosjek
  br_zapisa = 0;

  //Trazenje elemenata i punjenje polja
  for (i = 0; i < M; i++) {
    fseek (f, i*BLOK, SEEK_SET);
    fread (pretinac, sizeof (pretinac), 1, f);
    for (j = 0; j < C; j++) {
      if (pretinac[j].sifra != 0) { //preskačemo prazne zapise
        predvidjeni_pretinac = adresa(pretinac[j].sifra);
        if (predvidjeni_pretinac != i && pretinac[j].cijena <= prosjek) {
          polje = (int*) realloc(polje, (++br_zapisa) * sizeof(int));
          polje[br_zapisa-1] = pretinac[j].sifra;
        }
      }
      else
        break; //nakon prvog praznog u pretincu, svi su ostali prazni
    }
  }

  *brZapisa = br_zapisa;

  return polje;
}
```

Zadatak 5. (17 bodova)

Svaki zapis datoteke organizirane po načelu raspršenog adresiranja sadrži podatke o jednom proizvodu: *sifru proizvoda(int), naziv proizvoda (duljine do 50 znakova) te cijenu proizvoda(double)*.

Šifra nula (0) označava prazan zapis. Veličina bloka na disku je 4096 B. Očekuje se najviše 150000 zapisa, a kapacitet tablice je 15% veći. Prilikom upisa primjenjuje se metoda cikličkog preljeva. Ključ zapisa je *sifra*, a pretvorba ključa u adresu se obavlja već pripremljenom funkcijom

```
int adresa(int sifra);
```

Napišite funkciju koja će dinamički alocirati jednodimenzionalno polje i u njega upisati šifre i nazive proizvoda iz onih zapisa koji su završili u preljevu, a da je cijena proizvoda manja ili jednaka prosječnoj cijeni prozvoda čiji su zapisi završili u preljevu. Funkcija treba vratiti pokazivač na alocirano polje i broj elemenata. Prototip funkcije treba biti:

```
povratni_tip* fun(FILE *f, int *brZapisa);
```

Definirajte sve potrebne strukture i tipove podataka.

Zad5.

```
#define N 150000
#define BLOK 4096
#define C BLOK/sizeof (zapis)
#define M (int)(N*1.15/C)

typedef struct {
  int sifra;
  char naziv[50+1];
  double cijena;
} zapis;

typedef struct {
  int sifra;
  char naziv[50+1];
} povratni_tip;

int* fun(FILE *f, int *brZapisa){
  zapis pretinac[C];
  int i, j, predvidjeni_pretinac;
  int br_zapisa = 0;
  double suma = 0;
  double prosjek = 0;
  int *polje = NULL;

  //Brojanje zapisa
  for (i = 0; i < M; i++) {
    fseek (f, i*BLOK, SEEK_SET);
    fread (pretinac, sizeof (pretinac), 1, f);
    for (j = 0; j < C; j++) {
      if (pretinac[j].sifra != 0) { //preskačemo prazne zapise
        predvidjeni_pretinac = adresa(pretinac[j].sifra);
        if (predvidjeni_pretinac != i) {
          br_zapisa++;
          suma += pretinac[j].cijena;
      }
      else
        break; //nakon prvog praznog u pretincu, svi su ostali prazni
    }
  }

  //Racunanje prosjeka
  prosjek = (double) suma / br_zapisa;

  //Brojimo samo zapise koji su u preljevu i kojima je cijena <= prosjek
  br_zapisa = 0;

  //Trazenje elemenata i punjenje polja
  for (i = 0; i < M; i++) {
    fseek (f, i*BLOK, SEEK_SET);
    fread (pretinac, sizeof (pretinac), 1, f);
    for (j = 0; j < C; j++) {
      if (pretinac[j].sifra != 0) { //preskačemo prazne zapise
        predvidjeni_pretinac = adresa(pretinac[j].sifra);
        if (predvidjeni_pretinac != i && pretinac[j].cijena <= prosjek) {
          polje = (povratni_tip*) realloc(polje, (++br_zapisa) * sizeof(povratni_tip));
          polje[br_zapisa-1] .sifra= pretinac[j].sifra;
          strcpy(polje[br_zapisa-1] .naziv,  pretinac[j].naziv);
        }
      }
      else
        break; //nakon prvog praznog u pretincu, svi su ostali prazni
    }
```

```
}

 *brZapisa = br_zapisa;

 return polje;
}
```

Zadatak 1. *(5 bodova)*

Zadana je tablica raspršenog adresiranja s 8472 zapisa pohranjena u datoteci *"podaci.dat"* koja sadrži zapise definirane strukturom

```
typedef struct {
      int sifra;              //šifra studenta
      char prezimeIme[70+1]; //Prezime i ime studenta
      int logickiObrisan;     //Je li zapis logički obrisan
}  zapis;
```

Zapis je prazan ako je `sifra = 0`. Zapis je logički obrisan ako je `logickiObrisan = 1`. Preljevi su realizirani ciklički, upisom u prvi sljedeći slobodni pretinac. Pretinci su usklađeni s veličinom bloka od 512 okteta. Hash-tablica predimenzionirana je za 30%. Neka je ključ šifra studenta. Transformacija ključa u adresu obavlja se zadanom funkcijom `int adresa (int sifra)`. Napišite pretprocesorske direktive `#define` kojima se određuju parametri raspršenog adresiranja.

a) Napišite funkciju za logičko brisanje zapisa. Funkcija vraća 1 ako je pronašla i obrisala zapis a 0 inače. Funkcija ima prototip:
   `int brisi(int sifra, FILE *f)`
b) Napišite funkciju za dodavanje zapisa koja će zapis upisati na mjesto prvog praznog zapisa ili logički obrisanog zapisa u pretincu. Funkcija vraća 1 ako je uspješno dodala zapis a 0 inače. Funkcija ima prototip:
   `int upisi(int sifra, zapis novi, FILE *f)`

1. zadatak

```
#define BLOK 512L
#define N 8472
#define C ((int) (BLOK / sizeof (zapis)))
#define M ((int) (N / C *1.3))
typedef struct {
        int sifra;
        char prezimeIme[70+1];
        int logickiObrisan;
}  zapis;

int brisi(int sifra, FILE *f){
        zapis pretinac [C];
        int i = 0, poc = 0;
        int adr = adresa(sifra);
        poc = adr;
        do{
                //Procitaj pretinac odredjen sa adresom
                fseek(f, adr * BLOK, SEEK_SET);
                fread (pretinac, sizeof (pretinac), 1, f);
                //Provjeri zapise u pretincu
                for(i=0; i<C; i++){
                        if(pretinac[i].sifra == sifra){
                                pretinac[i].logickiObrisan = 1;
                                //Upis
                                fseek(f, -1 * BLOK, SEEK_CUR);
                                fwrite (pretinac, sizeof (pretinac), 1, f);
                                return 1;
                        }
                }
                //Pretinac je pun, prijeđi ciklički na sljedećega
                adr = (adr + 1) % M;
        } while(adr != poc);
        return 0;
}

int upisi(int sifra, zapis novi, FILE *f){
        zapis pretinac [C];
        int i = 0, poc = 0;
        int adr = adresa(sifra);
        poc = adr;
        do{
                //Procitaj pretinac odredjen sa adresom
                fseek(f, adr * BLOK, SEEK_SET);
                fread (pretinac, sizeof (pretinac), 1, f);
                for(i=0; i<C; i++){
                        if(pretinac[i].sifra == 0 ||
                                pretinac[i].logickiObrisan == 1){
                                //Zapis je prazan ili je logicki obrisan
                                pretinac[i] = novi;
                                //Upis
                                fseek(f, -1 * BLOK, SEEK_CUR);
                                fwrite (pretinac, sizeof (pretinac), 1, f);
                                return 1;
                        }
                }
```

```
                //Pretinac je pun, prijeđi ciklički na sljedećega
                adr = (adr + 1) % M;
        } while(adr != poc);
        return 0;
}
```

1. zadatak (12 bodova)

Jedan zapis datoteke organizirane po načelu raspršenog adresiranja definiran je strukturom:

```
typedef struct{
        int sifra;
        char naziv[70+1];
        double cijena;
} zapis;
```

Zapis je prazan ako je na mjestu šifre vrijednost nula. Parametri za raspršeno adresiranje nalaze se u datoteci parametri.h i oni su:

- BLOK..... veličina bloka na disku
- MAXZAP..... broj zapisa
- C..... broj zapisa u jednom pretincu
- M..... broj pretinaca

Preljevi su realizirani ciklički, upisom u prvi sljedeći slobodni pretinac. Ključ zapisa je šifra artikla, a transformacija ključa u adresu obavlja se zadanom funkcijom `int adresa(int sifra)`. Napišite funkciju koja će za dani redni broj pretinca N provjeriti postoje li zapisi koji su trebali biti zapisani u njega, ali su zbog preljeva zapisani u neki drugi pretinac. Vratiti redni broj pretinca koji ima najviše takvih zapisa te vratiti ukupan broj takvih zapisa. Funkcija vraća 0 ako takvi zapisi ne postoje.

1. **(12 bodova)**

```
int broj_preljeva(FILE *f, int N, int *rbr){
      zapis pretinac[C];
      int i,j, ukupno = 0, max = 0, brojac = 0;

      for (i = 0; i < M; i++) {
            brojac =0;
            fseek (f, i*BLOK, SEEK_SET);
            fread (pretinac, sizeof (pretinac), 1, f);
            if (i == N) continue;
            for (j = 0; j < C; j++) {
                  if (( pretinac[j].sifra != 0) &&
                  adresa(pretinac[j].sifra) == N)) {
                        ukupno++;
                        brojac++;
                  }
            }
            if (brojac > max) {
                  max = brojac;
                  *rbr = i;
            }

      }
      return ukupno;
}
```

1. zadatak (18 bodova)

a) U memoriji postoji sortirana tablica ključeva `tablica`. Napisati funkciju koja će pretraživanjem po blokovima pronaći zadani ključ i vratiti pokazivač na taj ključ. Ako zadani ključ nije pronađen, potrebno je vratiti NULL pokazivač. Funkcija treba izračunati i koristiti optimalnu veličinu bloka.
Prototip funkcije je:

```
int *trazi (int *tablica, int velicinaTablice, int *trazeniKljuc);
```

1. a) (15 bodova)

Verzija 1)

```
int *trazi(int *tablica, int velicinaTablice, int trazeniKljuc){
	int velicinaBloka = (int)sqrt((float)velicinaTablice);
	int i = 0, j = 0;
	int traziUZadnjem = 1;

	//Pretrazujemo vodece zapise u blokovima
	for(i = 0; i < velicinaTablice; i+= velicinaBloka){
		if(tablica[i] > trazeniKljuc){
			//Trazeni kljuc mora biti u prethodnom bloku pa ne
			//treba pretrazivati zadnji blok
			traziUZadnjem = 0;
			//Vodeci zapis u bloku veci je od trazenog
			//kljuca - pretrazujemo prethodni blok
			for(j = i - velicinaBloka; j < i; j++){
				if(tablica[j] == trazeniKljuc){
					//Trazeni kljuc je pronadjen
					return &tablica[j];
				}
			}
			//Trazeni kljuc nije pronadjen
			return NULL;
		} else if (tablica[i] == trazeniKljuc){
			//Vodeci zapis je trazeni kljuc
			return &tablica[i];
		}
	}

	//Trazeni kljuc nije pronadjen - pretrazit cemo posljednji blok ako
	//je zadnji element u tablici veći ili jednak trazenom kljucu
	if(traziUZadnjem = 1 && tablica[velicinaTablice - 1] >= trazeniKljuc){
		for(i = (i - velicinaBloka) + 1; i < velicinaTablice; i++){
			if(tablica[i] == trazeniKljuc){
				return &tablica[i];
			}
		}
	}
	//Kljuc nije pronadjen
	return NULL;
}
```

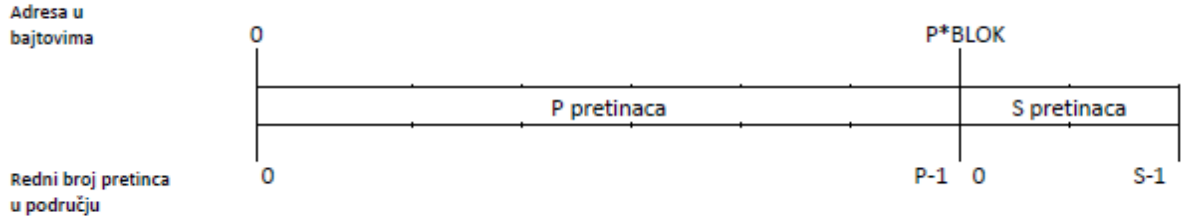
Verzija 2)

```
int *trazi2(int *tablica, int velicinaTablice, int trazeniKljuc){
	int velicinaBloka = (int)sqrt((float)velicinaTablice);
	int i = 0, j = 0;
	int traziUZadnjem = 1;

	do{
		if(tablica[i] < trazeniKljuc){
			if(i + velicinaBloka >= velicinaTablice){
				i++;
			}else{
				i += velicinaBloka;
			}
			continue;
		}else if(tablica[i] == trazeniKljuc){
			return &tablica[i];
		}
	}while(i < velicinaTablice);
}
```

1. zadatak (14 bodova)

Tablica raspršenog adresiranja sastoji se od primarnog područja veličine **P** pretinaca i preljevnog područja veličine **S** pretinaca. Oba se područja nalaze u istoj datoteci, prvo primarno područje, a zatim preljevno područje kako je prikazano na slici.



Kapacitet pretinca u oba područja je **C** zapisa. Zapis se sastoji od ključa **k** i cijelog broja **b** a definiran je strukturom

```
typedef struct{
      char k[20];
      int b;
} zapis;
```

Svi potrebni parametri i struktura definirani su u `hash.h` datoteci zaglavlja.

U oba područja zapisi su pohranjeni tehnikom raspršenog adresiranja. Transformacija ključa u adresu obavlja se zadanom funkcijom `int adresa (char *kljuc, char podrucje)`. Funkcija vraća redni broj pretinca u području, iz intervala [0, *brPretinaca* – 1] gdje je *brPretinaca* broj pretinaca u području za koje se određuje adresa. Ulazni parametar `podrucje` može biti `'P'` (primarno područje) ili `'S'` (preljevno područje).

Napisati funkciju `trazi` koja će za zadani ključ vratiti vrijednost cijelog broja **b**. Pretraživanje preljevnog područja mora se obaviti **rekurzivnim** pozivom iste funkcije (funkcije `trazi`). Ako zapis nije pronađen niti u primarnom niti u preljevnom području, funkcija vraća -1. Prototip funkcije je:

```
int trazi(char *kljuc, char podrucje, FILE *f)
```

Napomena: Nerekurzivno rješenje neće se priznavati.

Zad. 1 (14 bodova)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "hash.h"

int trazi(char *kljuc, char podrucje, FILE *f){
	zapis pretinac[C];
	int i=0;
	int adr = adresa(kljuc, podrucje);

	//Ako se radi o preljvnom području adresu tj. rbr. pretinca
	//moramo uvecati za broj pretinaca u primarnom području
	if(podrucje == 'S'){
		adr += P;
	}

	//Pozicioniraj se na trazeni pretinac i procitaj podatke
	fseek(f, adr * BLOK, SEEK_SET);
	fread(pretinac, sizeof(pretinac), 1, f);
	//Prodji kroz sve zapise u pretincu
	for(i=0; i<C; i++){
		if(strcmp(pretinac[i].k, kljuc) == 0){
			//Zapis je nadjen
			return pretinac[i].b;
		}
	}

	//Zapis nije nadjen, trazi u preljevnom podrucju rekurzivno
	//ako smo pretrazivali primarno podrucje. Inace vrati -1 jer
	//zapis nije nadjen
	if(podrucje == 'P'){
		return trazi(kljuc, 'S', f);
	}else{
		return -1;
	}
}
```