Zadatak 1
Napisati funkcije za rad s redom realiziranim jednostruko povezanom listom i funkcije za rad s redom realiziranim statičkim poljem. U oba reda spremaju se podaci tipa **float**.
Napisati još jednu dodatnu funkciju u kojoj će se podaci prepisati iz reda realiziranog listom u red realiziran poljem koristeći prije napisane funkcije tako da redoslijed elemenata ostane očuvan.

Napomena: Prilikom prebacivanja elemenata iz jednog reda u drugi, elementima redova smije se pristupati samo preko prije napisanih funkcija za rad sa redovima.

Zadatak 2
U red realiziran **dvostruko** povezanom listom spremaju se cjelobrojni podaci (**long**). Napisati funkciju za skidanje jednog podatka iz reda i funkciju za stavljanje novog podatka u red. Napisati glavni program u kojem će se, koristeći prije napisanu funkciju, u red staviti brojevi od 1 do 10.

Zadatak 3
Napisati funkcije za rad s redom realiziranim jednostruko povezanom listom i funkcije za rad s stogom realiziranim jednostruko povezanom listom. U red i u stog spremaju se podaci tipa **float**.
Napisati još jednu dodatnu funkciju u kojoj će se podaci prepisati iz reda u stog koristeći prije napisane funkcije.

Napomena: Prilikom prebacivanja elemenata sa reda u stog, elementima se smije pristupati samo preko prije napisanih funkcija.

Zadatak 4
U sortirano binarno stablo spremaju se cjelobrojni podaci (**int**).
Napisati funkciju koja će zadano stablo prepisati u jednostruko povezanu listu tako da i lista bude sortirana (stablo mora ostati nepromijenjeno).
Odrediti apriornu složenost funkcije u odnosu na broj elemenata stabla.

Funkcija treba imati prototip:

```
void prepisi(cvor *korijen, zapis **glava);
```

+ korištenje objektnih realizacija (Stog, Red, Lista, Stablo) kao na 2. međuispitu
+ ponoviti prethodno gradivo!

Zadaci za vježbu za završni ispit

Redovi

Zadatak 1.
Zadane su funkcije za stavljanje i skidanje elemenata iz reda realiziranog listom:

```
int DodajURed (int element, atom **glava, atom **rep);
int SkiniIzReda (int *element, atom **glava, atom **rep);
```

Napišite funkciju, koristeći gore navedene funkcije, koja stvara (i preko argumenata funkcije vraća) novi red koji sadrži samo negativne elemente iz zadanog reda. Funkcija ima protopip:

```
void RedNegativnih(atom **glavaZadani, atom **repZadani,
                   atom **glavaTrazeni, atom **repTrazeni);
```

Početni red mora ostati očuvan. Uputa: koristiti pomoćni red.

Zadatak 2.
Zadane su funkcije za stavljanje i skidanje elemenata sa stoga, te za stavljanje i skidanje elemenata iz reda (realiziranih listom):

```
int DodajNaStog (int element, atom **glava);
int SkiniSaStoga (int *element, atom **glava);
int DodajURed (int element, atom **glava, atom **rep);
int SkiniIzReda (int *element, atom **glava, atom **rep);
```

Napišite **rekurzivnu** funkciju, koristeći gore navedene funkcije, koja prebacuje elemente sa stoga u red, protopipa:

```
void Prebaci (atom **glavaStog, atom **glavaRed, atom **repRed);
```

Početni stog mora ostati očuvan. Redoslijed elemenata u redu isti je kao na stogu (onaj element koji prvi izlazi sa stoga je prvi element koji izlazi i iz reda). Red je na početku prazan.

Vezane liste

Zadatak 1.
Napisati funkciju koja ispisuje elemente zadane sortirane liste (u koju se zapisuju cijeli brojevi int) koji su manji od zadanog broja n.

Zadatak 2.
Napisati funkciju koja iz zadane liste u koju se zapisuju cijeli brojevi (int) briše parne elemente.

Zadatak 3.

U dvostruko povezanu listu spremaju se cjelobrojni podaci (long). Napisati funkciju koja će imati prototip:

```
int izbaciN(zapis **glava, zapis **rep, int N);
```

koja će iz liste izbaciti **zadnjih N** elemenata. Ako u listi ima manje od **N** elemenata, funkcija ih treba izbaciti sve. Funkcija treba vratiti stvarni broj izbačenih elemenata.

Stabla

Zajednički dio za sve zadatke sa stablima:
Stablo sadrži podatke o artiklima (naziv i cijenu) te je zadano sljedećom strukturom:

```
typedef struct {
    char naziv[20];
    int cijena;
} Element;

struct cv{
      Element element;
      struct cv *lijevo, *desno;
};
typedef struct cv cvor;
```

Nije dozvoljeno koristiti globalne varijable.

Zadatak 1.
Napisati funkciju čiji je prototip

```
cvor *trazi(cvor *korijen, char *naziv);
```

Funkcija treba vratiti pokazivač na čvor koji sadrži artikl s navedenim nazivom ili NULL ukoliko navedeni artikl ne postoji u stablu

Zadatak 2.
Ukoliko je ranije navedenom strukturom definiranu binarno stablo traženja sortirano po cijeni, napisati funkciju čiji je prototip

```
int trazi(cvor *korijen, float cijena);
```

koja će vratiti 1 ukoliko u stablu postoji artikl s traženom cijenom, odnosno 0 ako takav artikl ne postoji.

Zadatak 3.
Čvor binarnog nesortiranog stabla sadrži cjelobrojni podatak (int).
Napisati funkciju koja vraća 1 ako su sve cijene u stablu veće od n, a inače 0.

```
int SviVeciOdN(cvor *glava, int n);
```

Zadatak 4.

Napisati funkciju čiji je prototip

```
int max_cijena(cvor *korijen);
```

koja će pronaći najveću cijenu u stablu.

Zadatak 5.

Napisati funkciju čiji je prototip

```
int ukupna_cijena(cvor *korijen);
```

koja će pronaći ukupnu cijenu svih artikala u stablu.

Zadatak 6.

Napisati funkciju čiji je prototip

```
int dubina(cvor *korijen);
```

koja će izračunati dubinu stabla.

Zadatak 7.

Napisati funkciju čiji je prototip

```
void ispis_na_razini(cvor *korijen, int razini);
```

koja će ispisati sve čvorove na navedenoj razini. Napomena: Smijete koristiti pomoćne funkcije.

Zadatak 8.

Napisati funkciju čiji je prototip

```
cvor *zrcaliStablo(cvor *korijen);
```

koja će zrcaliti stablo (svakom čvoru međusobno zamijeniti lijevo i desno dijete)

Binarno sortirano stablo i gomila

Zadatak 1

U binarno stablo pohranjuje se niz podataka:
27, 52, 19, 17, 71, 23, 20, 5, 69
Treba nacrtati sortirano binarno stablo (lijevi manji, desni veći) ako je stablo popunjavano redom kako su dolazili podaci.

Zadatak 2

Kako izgleda ispis gomile po razinama ako je gomila formirana za ulazni niz 32, 47, 39, 61, 89, 97, 15, 72 algoritmom čija je složenost za najgori slučaj $O(n\log_2 n)$?

Zadatak 3

Kako izgleda ispis gomile po razinama ako je gomila formirana za ulazni niz 32, 47, 39, 61, 89, 97, 15, 72 algoritmom čija je složenost za najgori slučaj $O(n)$?

Zadatak 4

Zadana je gomila koja je pohranjena u polju:

93	82	47	31	64	23	17	27

Prikažite postupak uzlaznog heapsorta.

Rješenja

Redovi

Zadatak 1.

```
void RedNegativnih(atom **glavaZadani, atom **repZadani, atom
**glavaTrazeni, atom **repTrazeni)
{
	atom *glavaPomocni = NULL;
	atom *repPomocni = NULL;
	int element;

	while(SkiniIzReda(&element, glavaZadani, repZadani))
	{
		if (element < 0)
			DodajURed(element, glavaTrazeni, repTrazeni);
		DodajURed(element, &glavaPomocni, &repPomocni);
	}
	while(SkiniIzReda(&element, &glavaPomocni, &repPomocni))
		DodajURed(element, glavaZadani, repZadani);
}
```

Zadatak 2.

```
void Prebaci (atom **glavaStog, atom **glavaRed, atom **repRed)
{
	int element;
	if (SkiniSaStoga(&element, glavaStog))
	{
		DodajURed(element, glavaRed, repRed);
		Prebaci(glavaStog, glavaRed, repRed);
		DodajNaStog(element, glavaStog);
	}
}
```

Vezane liste

Zadatak 1.

```
void ispisiManjeOdN(int n, atom *glava)
{
	while ((glava != NULL) && (glava->element < n))
	{
		printf("%d\n", glava->element);
		glava = glava->sljedeci;
	}
}
```

Zadatak 2.

```
void brisiParne(atom **glava)
{
	atom *stari;
	atom *pomocni = *glava;

	while ((pomocni != NULL) && (pomocni->element % 2 == 0))
	{
		pomocni = (*glava)->sljedeci;
		free (*glava);
		*glava = pomocni;
	}
	while (pomocni != NULL)
	{
		while ((pomocni->sljedeci != NULL) &&
		((pomocni->sljedeci)->element % 2 != 0))
			pomocni = pomocni->sljedeci;

		if (pomocni->sljedeci != NULL)
		{
			stari = pomocni->sljedeci;
			pomocni->sljedeci = stari->sljedeci;
			free(stari);
		}
		else pomocni = pomocni->sljedeci;
	}

}
```

Zadatak 3.

```
int izbaciN(atom **glava, atom **rep, int N)
{
	int brojIzbacenih = 0;
	atom * stari;

	while((*rep != NULL) && (brojIzbacenih < N))
	{
		stari = *rep;
		*rep->preth;
		free(stari);
		brojIzbacenih++;
	}
	if (*rep == NULL) *glava = NULL;
	return brojIzbacenih;
}
```

Stabla

Zadatak 1.

```
cvor* trazi(cvor *korijen , char *naziv){
	if (korijen){
		if (strcmp(naziv, korijen->element.naziv) == 0)
```

```
                return korijen;
            else{
                cvor *rez = trazi(korijen->lijevo, naziv);
                if (rez == NULL)
                    return trazi(korijen->desno, naziv);
                else
                    return rez;
            }
        }
        return NULL;
}
```

Zadatak 2.

```
int trazi(cvor *korijen,  int cijena){
        if (korijen){
            if (cijena == korijen->element.cijena)
                return 1;
            else if (cijena < korijen->element.cijena)
                return trazi( korijen->lijevo,cijena);
            else
                return trazi( korijen->desno,cijena);

        }
        return 0;
}
```

Budući da se radilo o binarnom sortiranom stablu gdje je sortiranje izvršeno po cijeni, prethodna funkcija se mogla napisati i nerekurzivno:

```
int trazi(cvor *korijen,  int cijena){
        while(korijen){
            if (cijena == korijen->element.cijena )
                return 1;
            else if (cijena < korijen->element.cijena)
                korijen = korijen->lijevo;
            else
                korijen = korijen->desno;
        }
        return 0;
}
```

Zadatak 3.

```
int SviVeciOdN(cvor *korijen, int n)
{
  int rezLijevo, rezDesno = 0;
  if (korijen == NULL) return 1;
  rezLijevo = SviVeciOdN (korijen ->lijevo, n);
  if (rezLijevo)
      rezDesno = SviVeciOdN (korijen ->desno, n);
  return (rezLijevo && rezDesno && (korijen->element.cijena > n));
}
```

Zadatak 4.

```
int max_cijena(cvor *korijen){
	if (korijen){
		int max, max_desno;
		max = max_cijena(korijen->lijevo);
		max_desno = max_cijena(korijen->desno);
		max = max_desno > max ? max_desno : max;
		max = korijen->element.cijena > max ?
				korijen->element.cijena : max;
		return max;
	}
	return 0;
}
```

Napomena: Kod traženje minimalne cijene potrebno je dodatno paziti da li vraćena cijena iz nekog podstabla bila 0 ili ne, što s maksimumom nije bio slučaj.

Zadatak 5.

```
int ukupna_cijena(cvor *korijen){
	if (korijen)
		return korijen->element.cijena +
			ukupna_cijena (korijen->lijevo) +
			ukupna_cijena (korijen->desno);
	else
		return 0;
}
```

Zadatak 6.

```
int dubinaStabla(cvor *korijen){
	int dubinaLijevo, dubinaDesno;
	if (korijen){
		dubinaLijevo = dubinaStabla(korijen->lijevo);
		dubinaDesno = dubinaStabla(korijen->desno);
		return 1 + (dubinaLijevo < dubinaDesno ?
				dubinaDesno : dubinaLijevo);
	}
	return 0;
}
```

Zadatak 7.

```
void ispis_na_razini(cvor *korijen, int razina){
	if (korijen){
		razina--;
		if (razina==0)
			printf ("%-15s %6d\n", korijen->element.naziv,
				korijen->element.cijena);
		else if (razina>0){
```

```
                ispis_na_razini(korijen->lijevo , razina);
                ispis_na_razini(korijen->desno , razina);
            }
        }
}
```

Rješenje uz korištenje pomoćne funkcije:

```
void ispis_na_razini(cvor *korijen, int razina){
      ispisiNaRazini(korijen, razina, 0);
}


void ispisiNaRazini(cvor *korijen, int razina, int trenutna_razina){
      if (korijen){
            if (trenutna_razina < razina){
                  ispisiNaRazini(korijen->lijevo , razina,
                        trenutna_razina+1);
                  ispisiNaRazini(korijen->desno , razina,
                        trenutna_razina+1);
            }
            if (trenutna_razina == razina)
                  printf ("%-15s %6d Razina:%d\n",
                        korijen->element.naziv,
                        korijen->element.cijena,trenutna_razina);


      }
}
```

Zadatak 8.

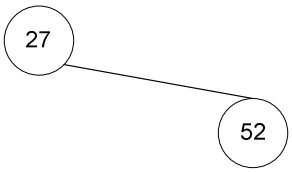
```
cvor *zrcaliStablo(cvor *korijen){
      if (korijen){
            cvor *temp;
            temp = korijen->lijevo;
            korijen->lijevo = zrcaliStablo(korijen->desno);
            korijen->desno = zrcaliStablo(temp);
      }
      return korijen;
}
```

Binarno sortirano stablo i gomila

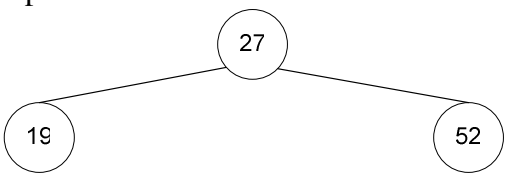
Zadatak 1

Upis elementa 27

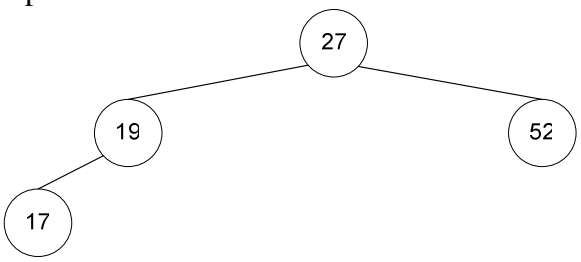
Upis elementa 52



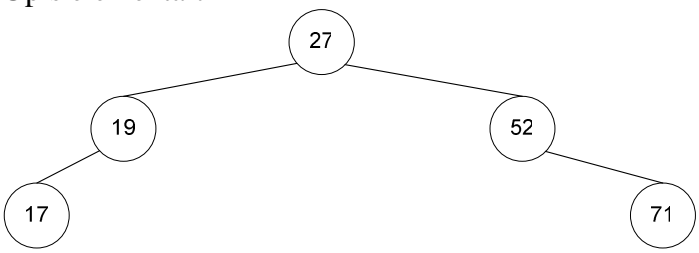
Upis elementa 19



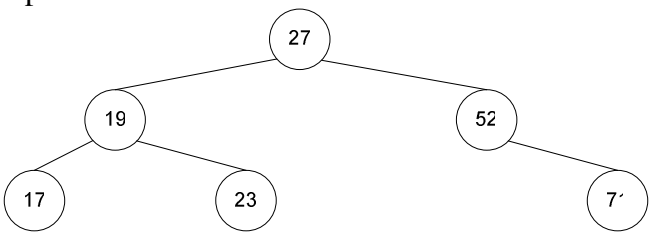
Upis elementa 17



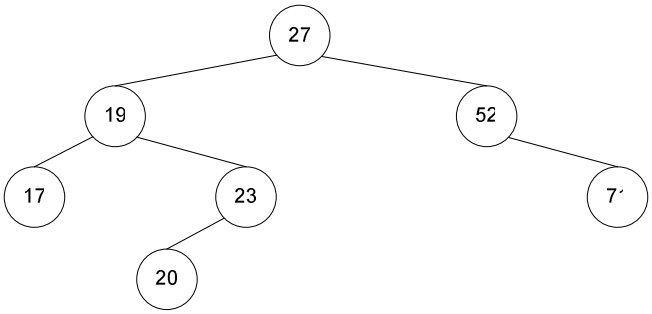
Upis elementa 71



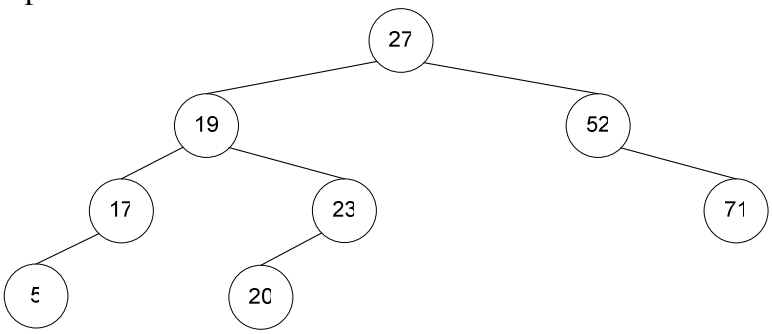
Upis elementa 23



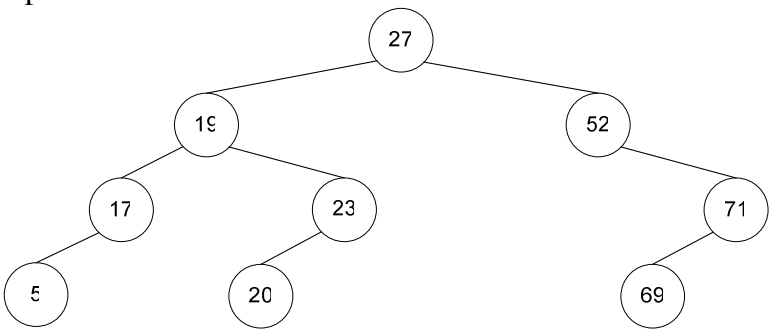
Upis elementa 20



Upis elementa 5



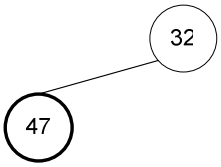
Upis elementa 69

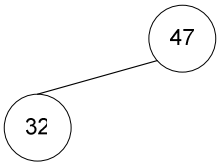


<u>Zadatak 2</u>

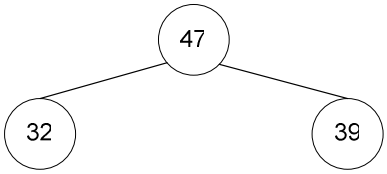
- ubaci 32

- ubaci 47

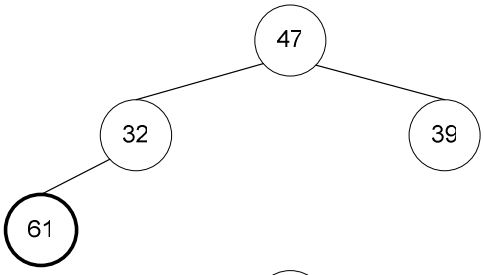


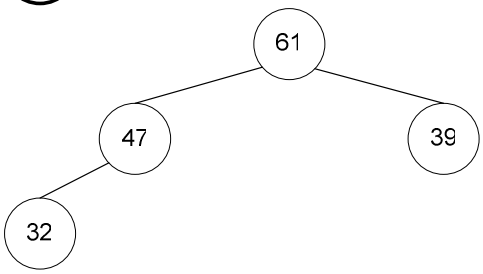


- ubaci 39

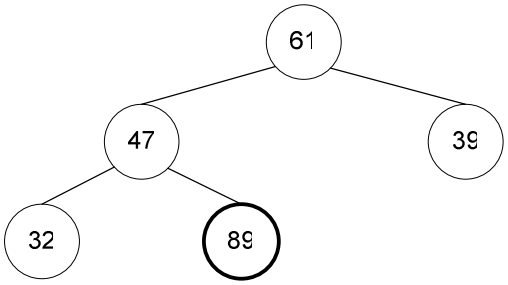


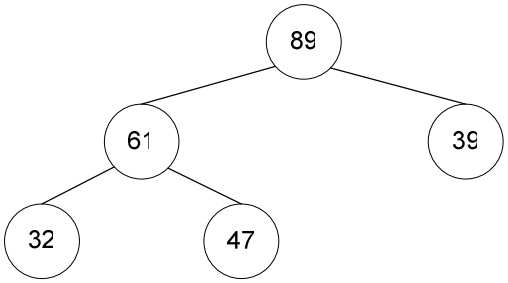
- ubaci 61



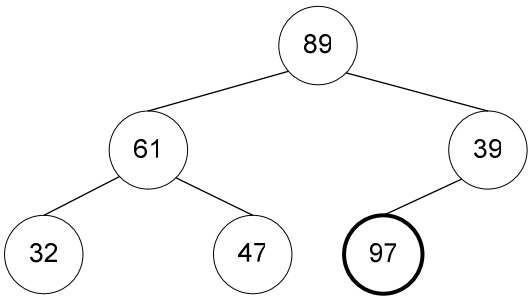


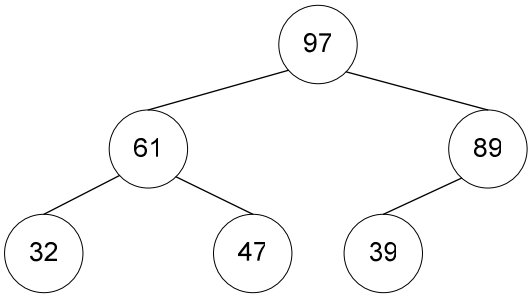
- ubaci 89



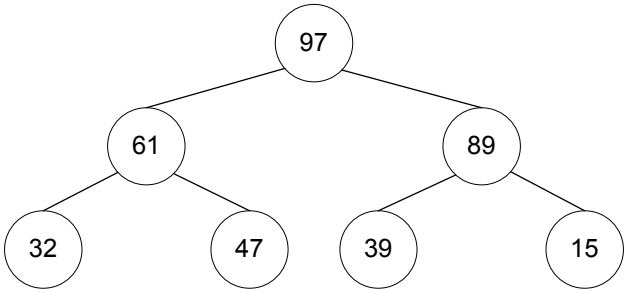


- ubaci 97

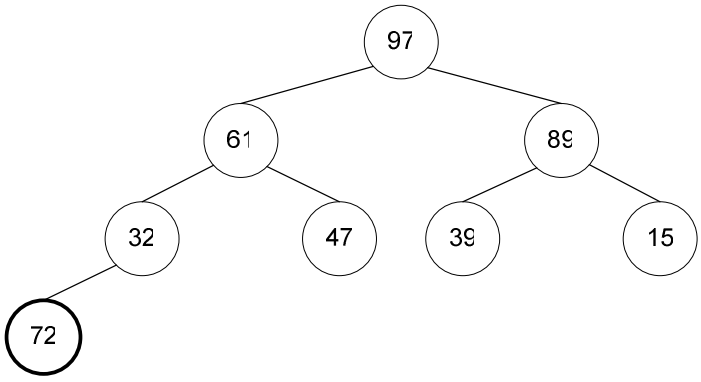




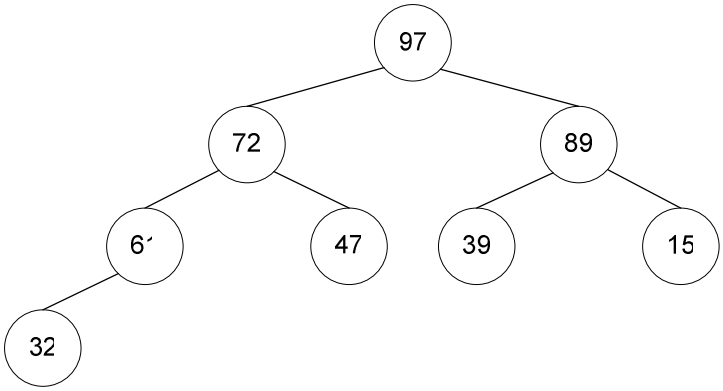
- ubaci 15



- ubaci 72

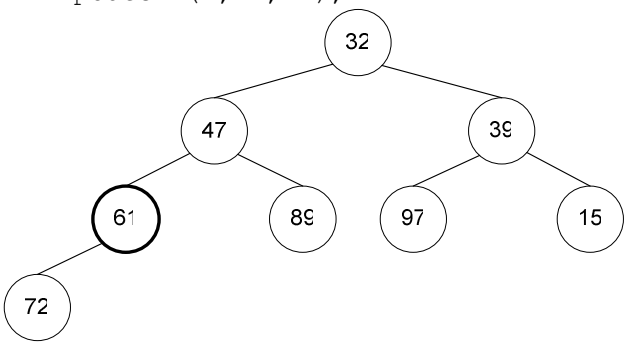


Gomila:

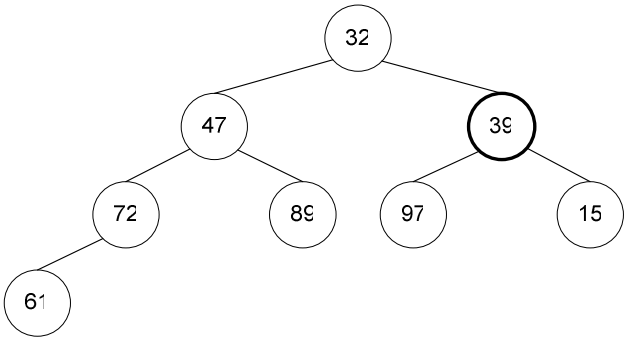


Zadatak 3

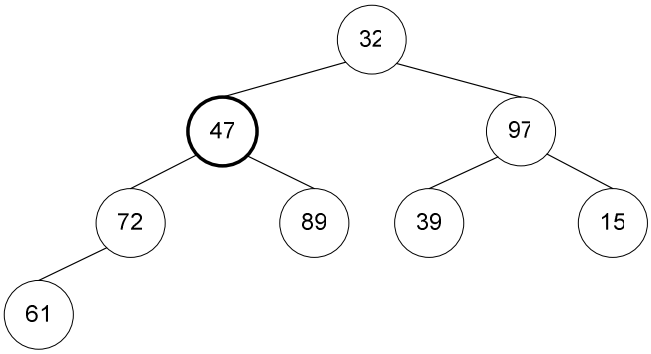
- `i = n/2;  // i = 4`
- `podesi (A, i, n);`



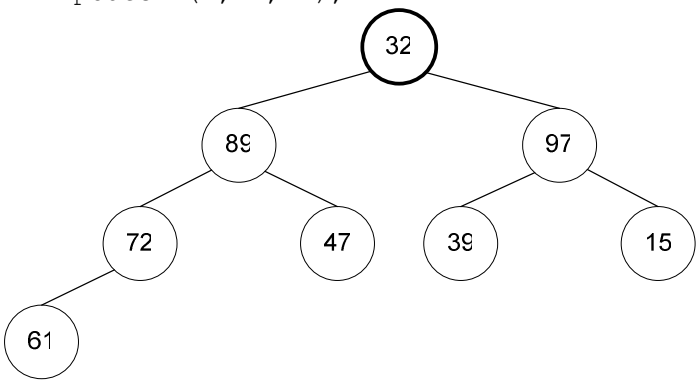
- `i = i-1;  // i = 3`
- `podesi (A, i, n);`



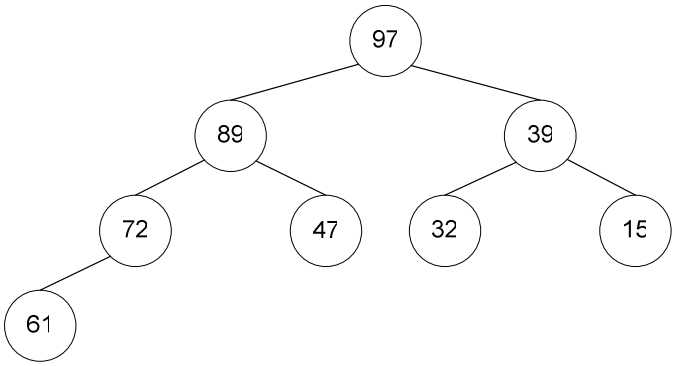
- `i = i-1;  // i = 2`
- `podesi (A, i, n);`



- `i = i-1;  // i = 1`
- `podesi (A, i, n);`

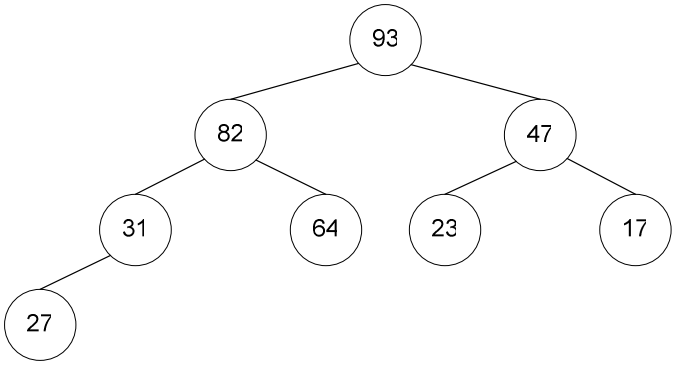


Gomila:



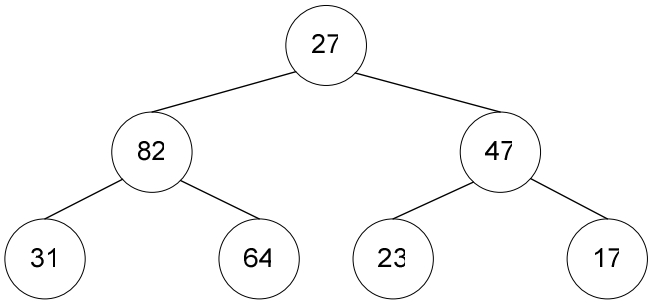
Zadatak 4

Početna gomila:



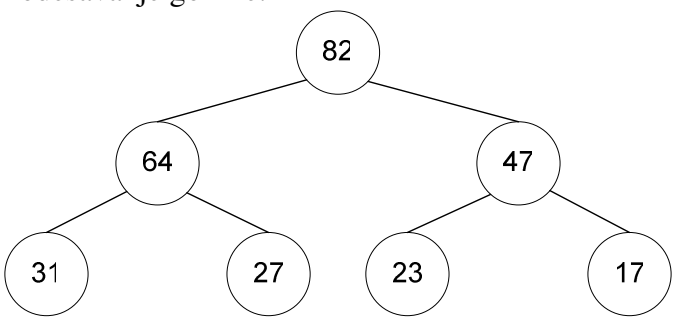
93	82	47	31	64	23	17	27

Element s vrha gomile zamjenjuje se s posljednjim elementom nesortiranog dijela polja:



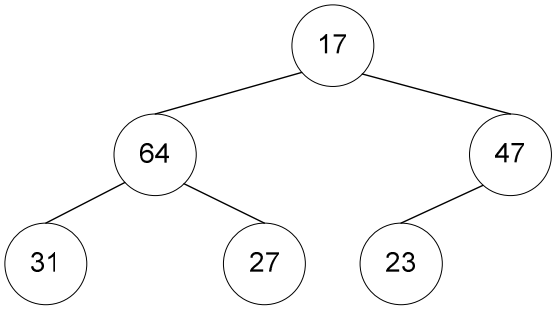
27	82	47	31	64	23	17	93

Podešavanje gomile:



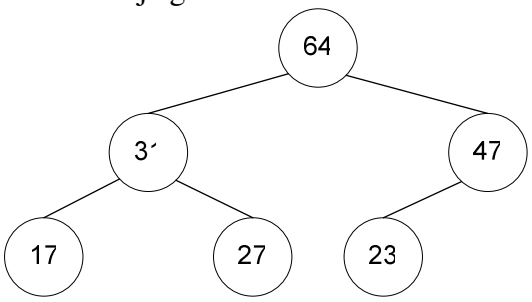
82	64	47	31	27	23	17	93

Element s vrha gomile zamjenjuje se s posljednjim elementom nesortiranog dijela polja:



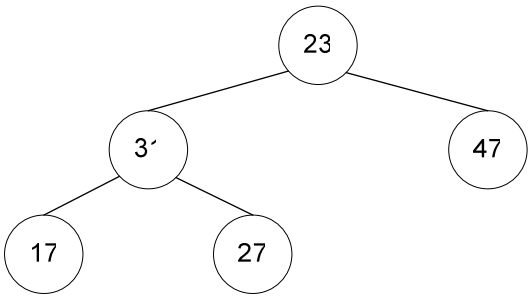
17	64	47	31	27	23	82	93

Podešavanje gomile:



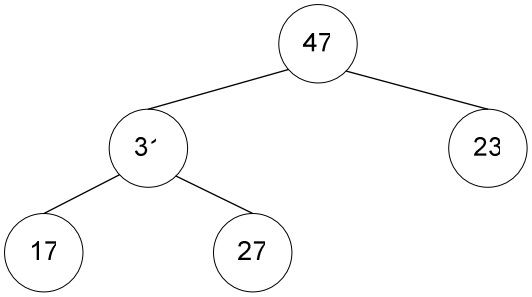
64	31	47	17	27	23	82	93

Element s vrha gomile zamjenjuje se s posljednjim elementom nesortiranog dijela polja:



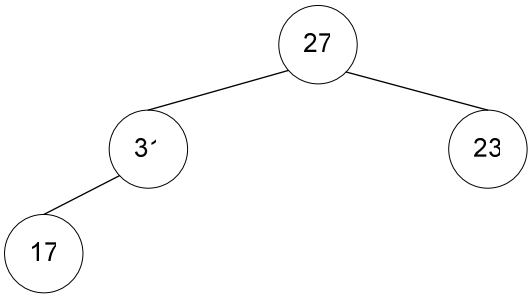
23	31	47	17	27	64	82	93

Podešavanje gomile:



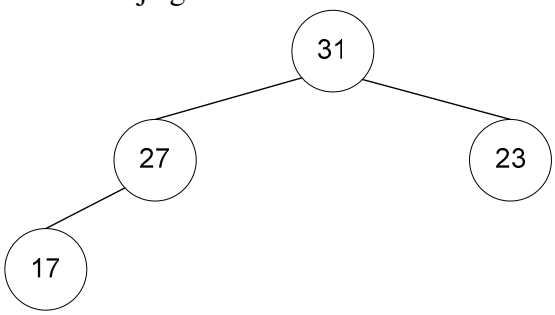
47	31	23	17	27	64	82	93

Element s vrha gomile zamjenjuje se s posljednjim elementom nesortiranog dijela polja:



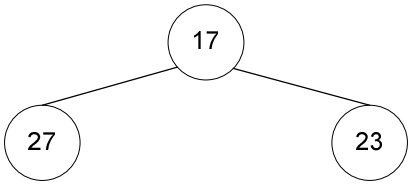
27	31	23	17	47	64	82	93

Podešavanje gomile:



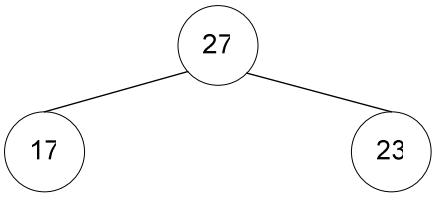
31	27	23	17	47	64	82	93

Element s vrha gomile zamjenjuje se s posljednjim elementom nesortiranog dijela polja:



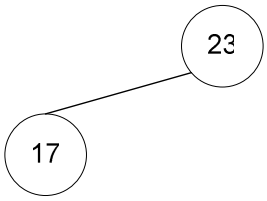
17	27	23	31	47	64	82	93

Podešavanje gomile:



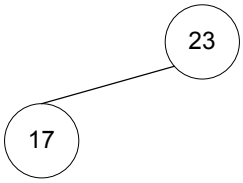
27	17	23	31	47	64	82	93

Element s vrha gomile zamjenjuje se s posljednjim elementom nesortiranog dijela polja:



23	17	27	31	47	64	82	93

Podešavanje gomile:



23	17	27	31	47	64	82	93

Element s vrha gomile zamjenjuje se s posljednjim elementom nesortiranog dijela polja:

17	23	27	31	47	64	82	93

Sortirano polje:

17	23	27	31	47	64	82	93

1. Zadatak

U binarno stablo pohranjuje se niz podataka:

12, 15, 5, 3, 7, 2, 18, 11

a) treba nacrtati sortirano binarno stablo (lijevi manji, desni veći) ako je stablo popunjavano redom kako su dolazili podaci

b) poredati ulazne podatke tako da nastupi najbolji slučaj, nacrtati

c) poredati ulazne podatke tako da nastupi neki od najgorih slučajeva, nacrtati

d) kolika su apriorna vremena izvođenja za pronalaženje pojedinog čvora za b) i za c)

2. Zadatak

Čvor binarnog nesortiranog stabla sadrži cjelobrojni podatak (int).

Napisati funkciju koja vraća broj negativnih elemenata u stablu.

```
int BrNegativnih(cvor *glava);
```

3. Zadatak

Čvor binarnog nesortiranog stabla sadrži cjelobrojni podatak (int).

Napisati funkciju koja vraća 1 ako su svi elementi u stablu pozitivni, inače 0.

```
int JesuPozitivni(cvor *glava);
```

4. Zadatak

Čvor binarnog stabla sadrži **matični broj studenta** (long) i **godinu studija** (int).

Potrebno je napisati funkciju koja provjerava da li su svi studenti o kojima se nalaze zapisi u binarnom stablu na zadanoj godini studija. Funkcija treba vratiti 1, ako su svi studenti na zadanoj godini studija, a 0 inače.

```
int funkcija(cvor *glava, int godina);
```

5. Zadatak

U binarno stablo spremaju se zapisi tipa **int**.

a) Napisati funkciju koja će pronaći najveći element u stablu ako je stablo sortirano (iskoristiti sortiranost stabla), te vratiti pokazivač na čvor u kojem se taj element nalazi.

b) Napisati funkciju koja će pronaći najveći element u stablu ukoliko stablo nije sortirano, te vratiti pokazivač na čvor u kojem se taj element nalazi.

.Obje funkcije trebaju vratiti **NULL** ako je stablo prazno i trebaju imati prototip:

```
cvor * najmanjiElement(cvor *glava);
```

6. Zadatak

U binarno se stablo spremaju elementi tipa **int**.

Napisati funkciju koja će provjeriti jesu li dva zadana stabla jednaka (i po strukturi i po vrijednosti elemenata).

Funkcija treba vratiti 1 ako su stabla jednaka, a 0 inače i treba imati prototip:

```
int jednaka(cvor *glava1, cvor *glava2);
```

7. Zadatak

U binarnom stablu nalaze se podaci tipa **int**. Napisati funkciju koja će izračunati prosječnu dubinu elemenata u stablu. Korijen stabla ima dubinu 1. Funkcija treba imati prototip:

```
float prosjDubina(cvor *korijen);
```

Napomena: Dozvoljeno je korištenje pomoćnih funkcija.

8. Zadatak

U binarno stablo spremaju se cjelobrojni podaci (**int**). Napisati funkciju koja će pronaći i vratiti koliko se elemenata stabla nalazi na zadanoj razini. Funkcija treba raditi efikasno (ne smije pretraživati čitavo stablo već samo do zadane razine) i treba imati prototip:

```
int BrNaRazini(cvor *glava, int razina);
```

Napomena: korijen stabla nalazi se na razini 1.

9. Zadatak

Napisati funkciju koja će provjeriti je li zadano binarno stablo **puno**. Funkcija mora vraćati vrijednost 1 ukoliko je binarno stablo puno ili 0 ukoliko nije puno i imati prototip:

```
int punoStablo(cvor *glava);
```

Napomene: Smiju se koristiti pomoćne funkcije. Stablo koje je visine k i ima 2^{k-1} elemenata naziva se puno binarno stablo.

10. Zadatak

Zadana je funkcija koja će provjerava da li je zadano binarno stablo potpuno (**ovu funkciju ne treba napisati**). Funkcija vraća 1 ako je stablo potpuno, a 0 inače i ima prototip:

```
int jelipotpuno(cvor *korijen);
```

Napisati novu funkciju koja će provjeriti je li zadano stablo gomila (da li je potpuno i vrijedi li za svaki čvor da je roditelj veći od djece). Prilikom provjere potpunosti zadanog stabla koristiti gornju funkciju. Nova funkcija treba imati prototip:

```
int jeligomila(cvor *korijen);
```

Napomena: Dopušteno je korištenje pomoćnih funkcija.