Zadaci za vježbu za završni ispit

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.

Zadatak 4.
U čvorovima sortirane dvostruko povezane liste nalaze se zapisi o knjigama: **šifra knjige** (int), **naziv knjige** (100+1 znak), **šifra autora** (int) i **cijena knjige** (float). Lista je uzlazno sortirana prema cijeni knjige. Potrebno je napisati funkciju koja će iz liste izbaciti knjige s najvišom i najmanjom cijenom Napomena: više knjiga može imati istu cijenu.

Zadatak 5.
Jednostruko povezana lista u memoriji sadrži statističke podatke o svim dosadašnjim nastupima jednog tenisača na turnirima. Element liste sadrži **šifru turnira** (int) i **broj pobjeda na tom turniru** (int). Napisati ***rekurzivnu*** funkciju koja će vratiti ukupan broj pobjeda tog tenisača na svim turnirima.
Prototip funkcije je zadan i nije ga dozvoljeno mijenjati:

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

Stabla

Zajednički dio za sve zadatke sa stablima:
Nesortirano binarno 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.

Objektno orijentirano programiranje

Zadatak 1
Napisati razred Matrica koja osim potrebnih članskih varijabli ima dva konstruktora (jedan prima jedan parametar i alocira prostor za kvadratnu matricu primljene dimenzije, a drugi dva parametra i alocira prostor za matricu danih dimenzija), destruktor, metode za vraćanje elemenata matrice, te metodu koja vraća transponiranu matricu.

Zadatak 2
Implementirati konstruktore, destruktor i metode klase `Skup` koja predstavlja skup elemenata tipa float:

```
class Skup
```

```
{
private:
   float *_elementi;
   int _brojElemenata;
public:
   Skup();
   Skup(int brojElemenata);
   void Dodaj(float element);
   int BrojElemenata();
   int BrojPojavljivanja(float element);
     ~Skup();
};
```

Defaultni konstruktor alocira memoriju za 10 elemenata, postavlja broj elemenata na 0. Drugi konstruktor alocira memoriju za zadani broj elemenata (i postavlja broj elemenata na 0).
Metoda `Dodaj` dodaje element na kraj do sada popunjenog niza elemenata, metoda `BrojElemenata` vraća broj elemenata, a metoda `BrojPojavljivanja` vraća koliko se puta zadani element pojavio u skupu.

Zadatak 3

Zadane su klase Kruznica i Tocka sa sljedećim definicijama:

```
class Kruznica
{
public:
   Kruznica() {}
   Kruznica(float inX, float inY, float inRad)
   {
         _x = inX; _y = inY; _radius = inRad;
   }

   float GetCenterX() { return _x; }
   float GetCenterY() { return _y; }
   float GetRadius()  { return _radius; }

private:
   float _x, _y;
   float _radius;
};

class Tocka
{
public:
   Tocka() {}
   Tocka(float inX, float inY)
   {
         _x = inX; _y = inY;
   }

   float GetX() { return _x; }
```

```
    float GetY() { return _y; }

private:
    float _x, _y;

};
```

Potrebno je napisati funkciju koja će za zadanu kružnicu i zadano polje točaka, prebrojati koliko se točaka nalazi unutar zadane kružnice. Prototip funkcije je:

```
int KolikoUnutar(Kruznica kruz, Tocka poljeTocaka[], int n);
```

Stog – objektno orijentirana implementacija[1]

Zajednički dio za sve zadatke sa objektno orijentirano implementiranim stogom :

Za spremanje cijelih brojeva na stog definirana je klasa `Stog` koja ima jedan konstruktor i tri javne funkcije:

```
Stog::Stog();
void Stog::Stavi(int element);
void Stog::Skini(int *element);
int Stog::Prazan();
```

Funkcija `Prazan` vraća 1 ukoliko je stog prazan, a 0 inače. Za klasu `Stog` nije definiran copy konstruktor, a možete pretpostaviti da u svakom objektu klase Stog ima dovoljno mjesta za dodavanje novih elemenata za stog. Pokušaj skidanja elementa iz praznog stoga, dovest će do pogreške. Ukoliko drugačije nije navedeno, smijete koristiti pomoćne stogove, ali ne i dodatna polja.

Zadatak 1:
Napisati funkciju koja će napraviti i vratiti duplikat zadanog stoga (isti takav stog, isti elementi koji imaju isti redoslijed). Zadani stog mora na kraju ostati nepromijenjen. Funkcija mora imati prototip:

```
Stog *duplikat(Stog *zadani);
```

Zadatak 2:
Napisati funkciju koja će iz stoga izbaciti sve parne brojeve. Međusobni redoslijed neparnih brojeva mora ostati nepromijenjen. Funkcija vraća broj izbačenih brojeva te ima sljedeći prototip:

```
int izbaci_parne(Stog *stog);
```

Zadatak 3:
Napisati funkciju koja će stvoriti novi stog koji će sadržavati sve one brojeve iz zadanog stoga za koje funkcija čiji je prototip `int provjeri(int n)` vrati istinu. Poredak

[1] Implementacija stoga može biti i drugačija pri čemu se rješenja djelomično mijenjaju. Vidi napomenu iza rješenja zadataka sa stogom.

prostih brojeva u novom stogu nije bitan. Zadani stog mora na kraju ostati nepromijenjen. Funkcija mora imati prototip:

```
Stog *kopiraj_provjerene(Stog *zadani);
```

Zadatak 4:

Napisati funkciju koja će na osnovu ulaznih stogova `stog1` i `stog2` stvoriti novi stog koristeći pritom sljedeće pravilo da se za novi element u novom stogu uvijek odabere manji od elemenata na vrhu stogova `stog1` i `stog2`. Ukoliko se jedan od stogova isprazni, onda uzeti preostale elemente iz drugog.
Ulazni stogovi moraju nakon završetka funkcije ostati nepromijenjeni. Funkcija mora imati prototip:

```
Stog *spoji(Stog *stog1, Stog *stog2);
```

Zadatak 5*:

Napisati funkciju koja će iz stoga izbaciti sve one brojeve koji se pojavljuju dva ili više puta. Funkcija ima sljedeći prototip:

```
void izbaci_duple(Stog *stog);
```

Zadatak 6:

Napisati funkciju koja će iz stoga izbaciti sva pojavljivanja broja n. Funkcija, mora u novi stog, koji je inicijalno prazan, ubaciti brojeve koji predstavljaju koliko je broj bio udaljen od originalnog vrha stoga. Funkcija ima sljedeći prototip:

```
void izbaci_broj(Stog *stog, int n, Stog *novi);
```

Primjer: Ukoliko je ulazni stog bio (vrh je naveden krajnje desno) bio:

```
8 1 4 9 9 2 5 9 6 7 9
```

i ukoliko je funkcija pozvana s `izbaci_broj(stog, 9, novi)`
nakon završetka funkcije, početni stog će biti

```
8 1 4 2 5 6 7
```

a novi stog će imati vrijednosti `0 3 6 7` (može i `7 6 3 0`)

Rješenja

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 *prethodni, *pomocni = *glava;

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

Zadatak 3.

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

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

Zadatak 4.

```
typedef struct s {
	int sifraKnjige;
	char nazivKnjige[100+1];
	int sifraAutora;
	float cijena;
	struct s *sljed;
	struct s *preth;
} cvor;

void izbaci(cvor **glava, cvor **rep) {
	cvor *p = *glava;
	float minCijena, maxCijena;

	if (!p) { // ako je lista prazna
		return;
	}

	// Lista je sortirana po cijeni
	minCijena = p->cijena;

	// izbaciti one na početku liste s najmanjom cijenom
	p = *glava;
	while (p && p->cijena == minCijena) {
		*glava = p->sljed;
		if (p->sljed) {
			p->sljed->preth = NULL;
		}
		free(p);
		p = *glava;
	}

	// ako je ispraznjena cijela lista
	if (*glava == NULL) {
		*rep = NULL;
	}
	// ako je jos ostalo elemenata u listi
	else {
		p = *rep;
		maxCijena = p->cijena;
```

```
            // izbaciti one s najvisom cijenom
            while (p && p->cijena == maxCijena) {
                  *rep = p->preth;
                  if (p->preth) {
                        p->preth->sljed = NULL;
                  }
                  free(p);
                  p = *rep;
            }
            // ako je ispraznjena cijela lista
            if (*rep == NULL) {
                  *glava = NULL;
            }
      }
   }

   void main() {
      cvor *glava = NULL, *rep = NULL;
      ...
      izbaci(&glava, &rep);
   }
```

Zadatak 5.

```
struct s1 {
      int sifturn;
      int brpobj;
      struct s1 *sljed;
};
typedef struct s1 cvor;

int broj_pobjeda(cvor *glava){
  if (glava)
    return glava->brpobj + broj_pobjeda(glava->sljed);
  else
    return 0;
}
```

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.

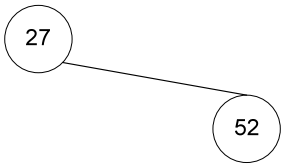
```
cvor *zrcaliStablo(cvor *korijen){
	if (korijen){
		cvor *temp;
		temp = korijen->lijevo;
		korijen->lijevo = zrcaliStablo(korijen->desno);
		korijen->desno = zrcaliStablo(temp);
	}
	return korijen;
}
```

Binarno sortirano stablo i gomila

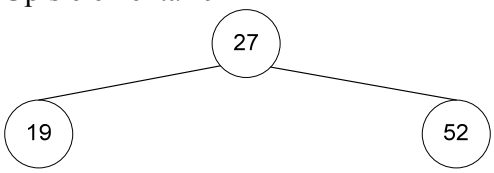
Zadatak 1

Upis elementa 27

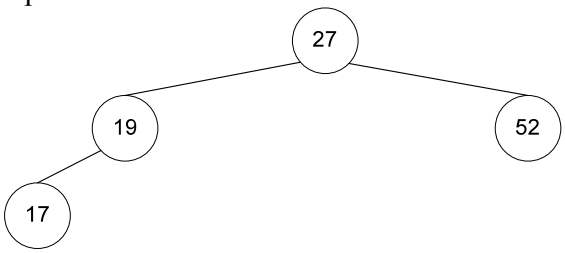
Upis elementa 52



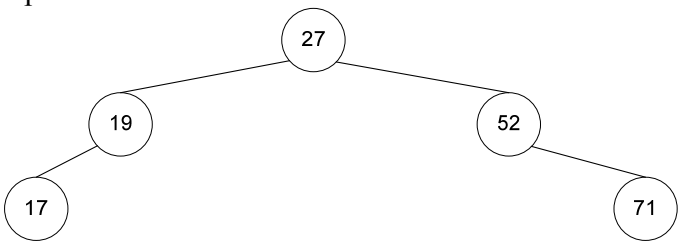
Upis elementa 19



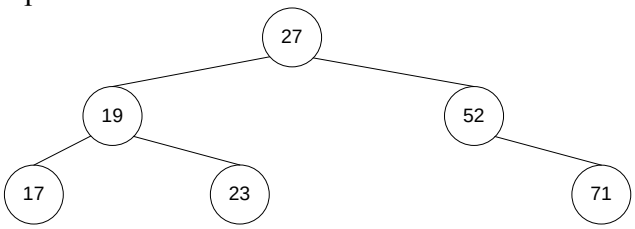
Upis elementa 17



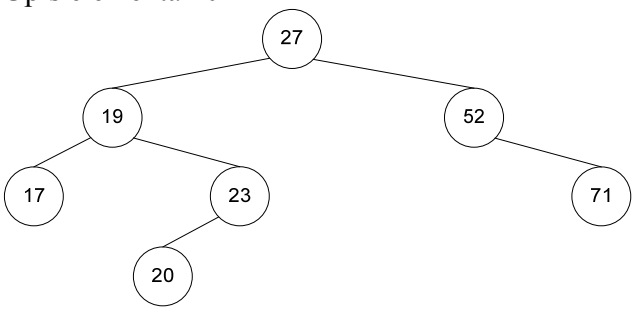
Upis elementa 71



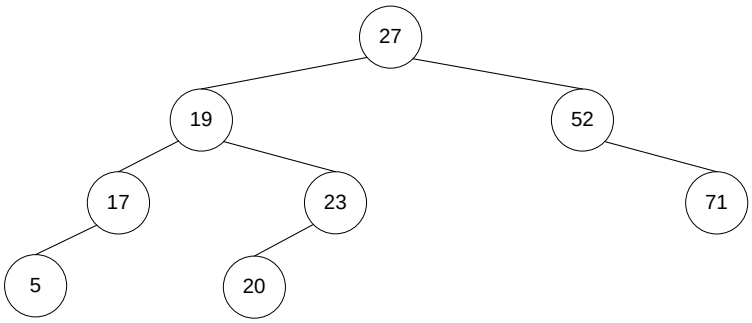
Upis elementa 23



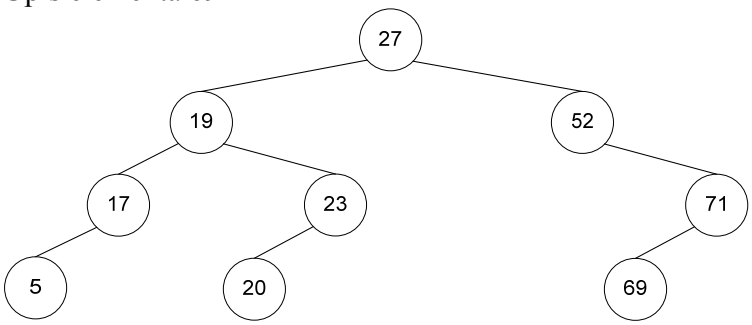
Upis elementa 20



Upis elementa 5



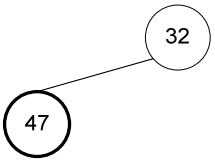
Upis elementa 69

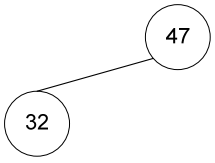


<u>Zadatak 2</u>

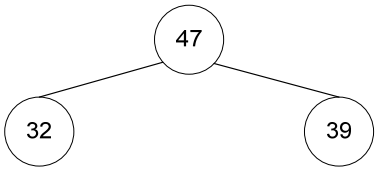
- ubaci 32

- ubaci 47

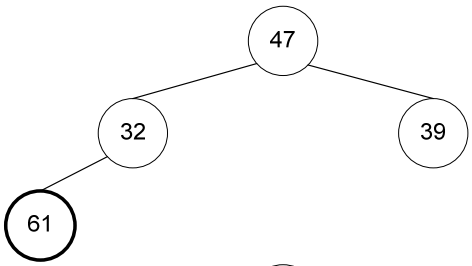


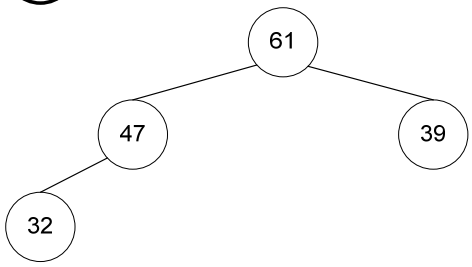


- ubaci 39

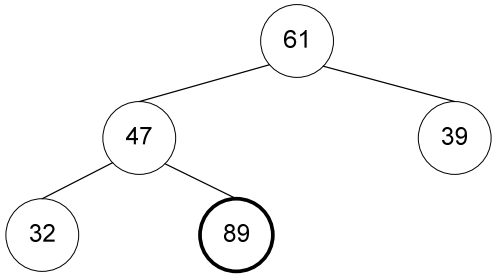


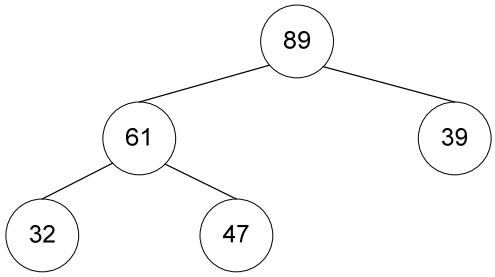
- ubaci 61



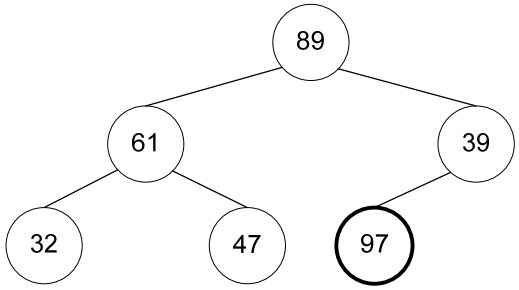


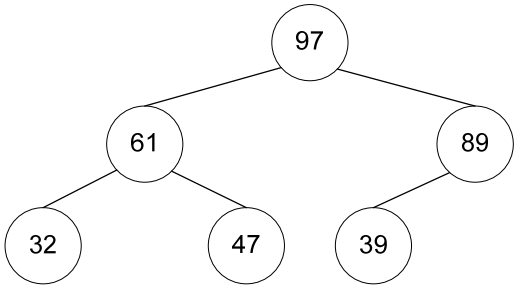
- ubaci 89



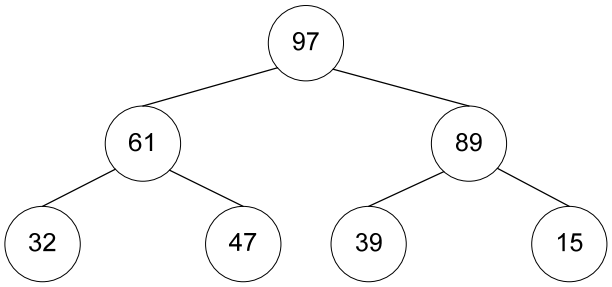


- ubaci 97

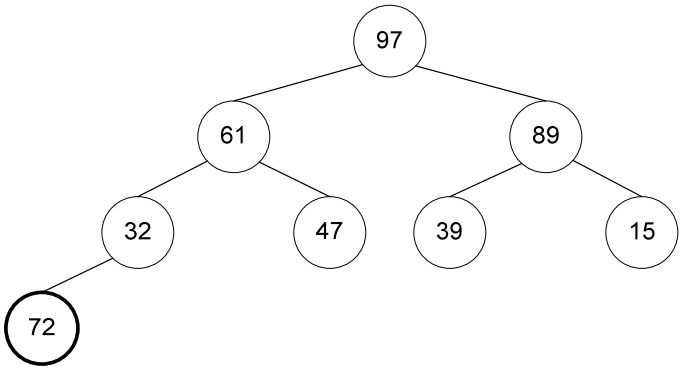




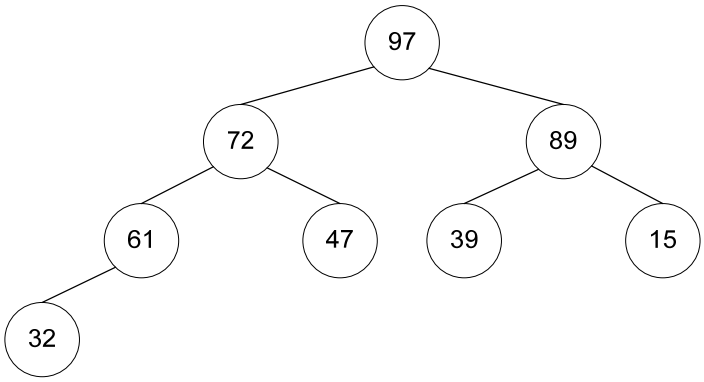
- ubaci 15



- ubaci 72

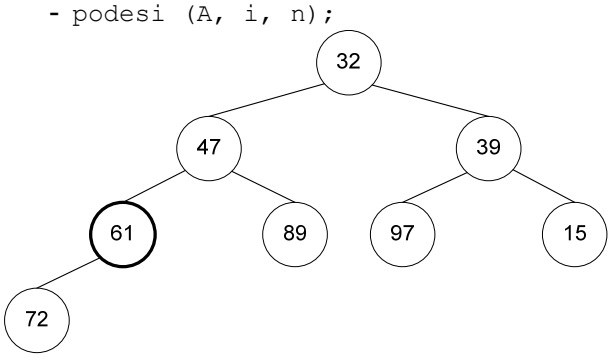


Gomila:

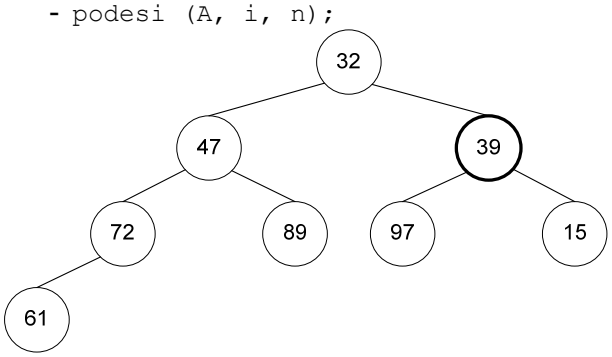


Zadatak 3

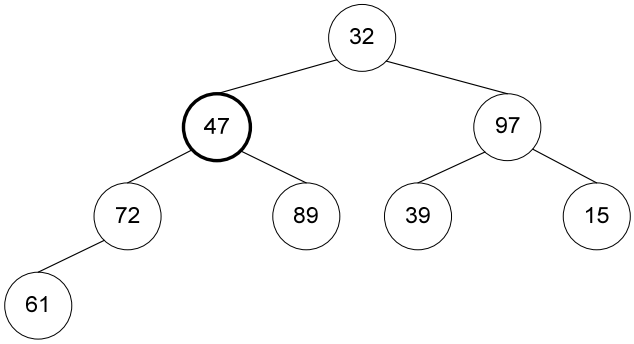
- `i = n/2;  // i = 4`
- `podesi (A, i, n);`



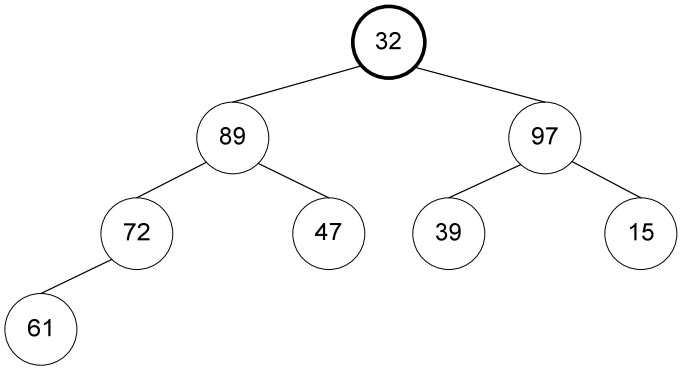
- `i = i-1;  // i = 3`
- `podesi (A, i, n);`



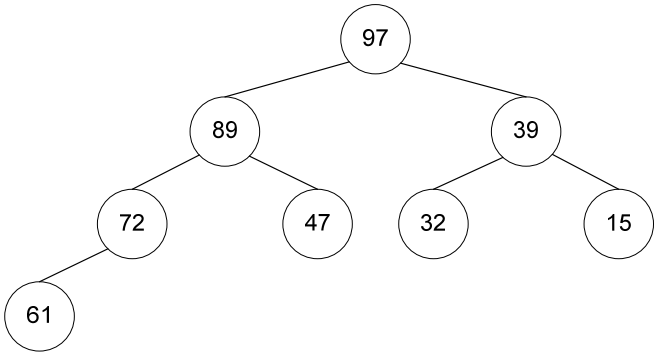
- `i = i-1;  // i = 2`
- `podesi (A, i, n);`



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

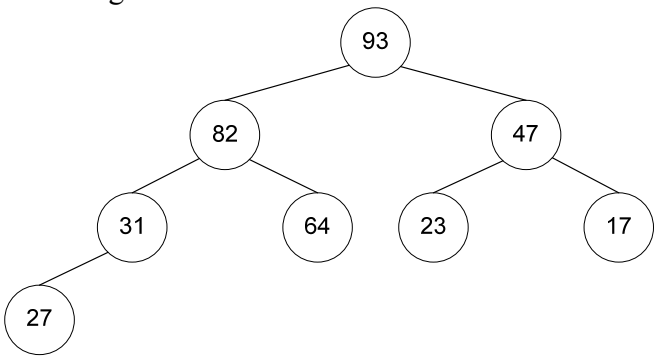


Gomila:



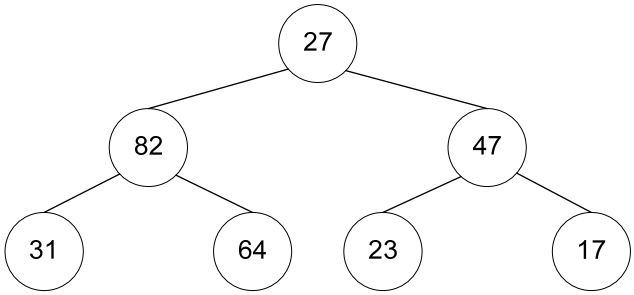
Zadatak 4

Početna gomila:



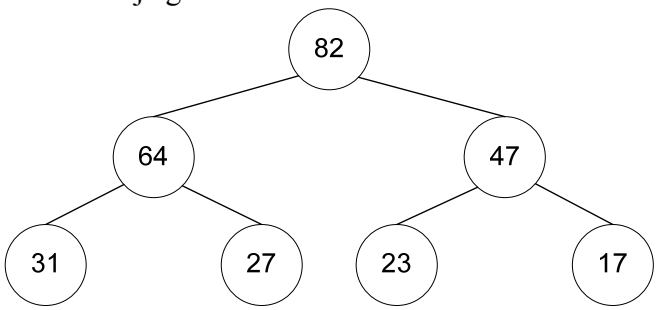
93	82	47	31	64	23	17	27

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



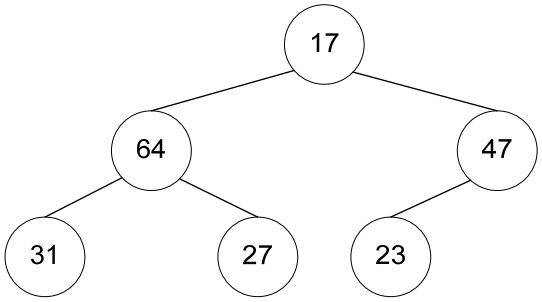
27	82	47	31	64	23	17	93

Podešavanje gomile:



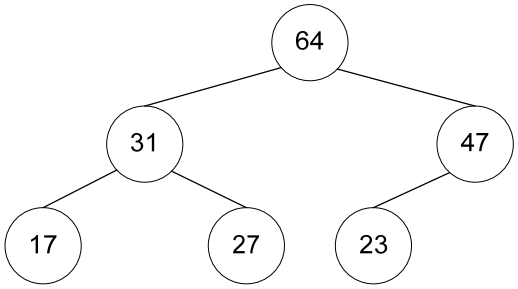
82	64	47	31	27	23	17	93

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



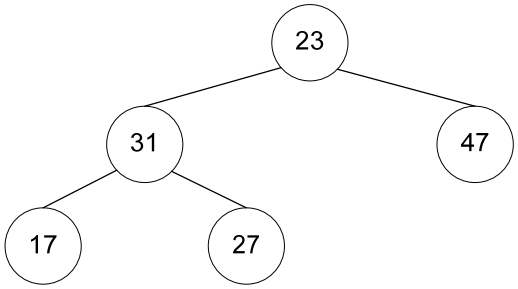
17	64	47	31	27	23	82	93

Podešavanje gomile:



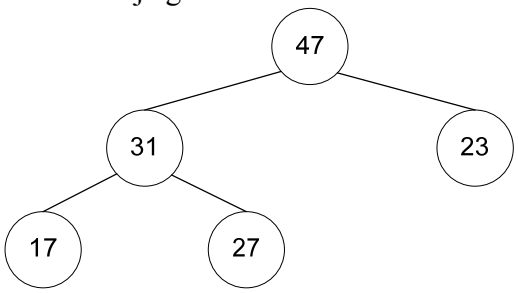
64	31	47	17	27	23	82	93

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



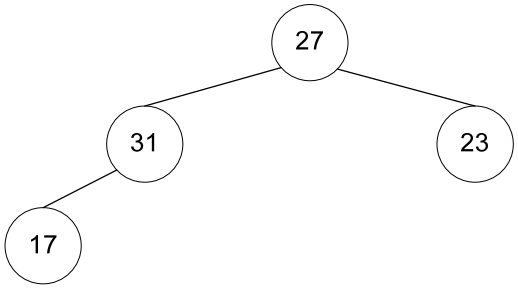
23	31	47	17	27	64	82	93

Podešavanje gomile:



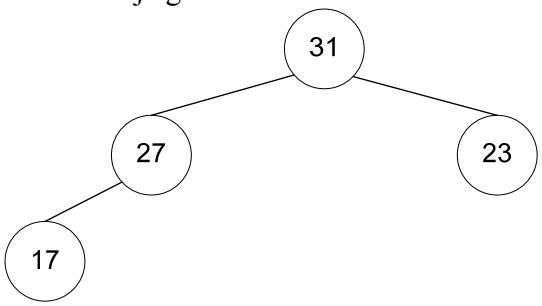
47	31	23	17	27	64	82	93

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



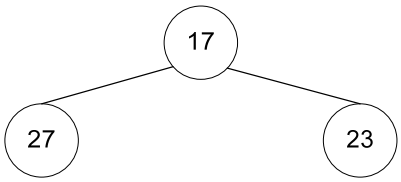
27	31	23	17	47	64	82	93

Podešavanje gomile:



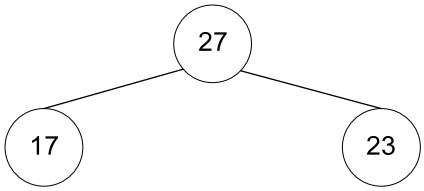
31	27	23	17	47	64	82	93

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



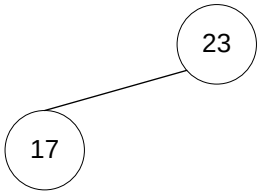
17	27	23	31	47	64	82	93

Podešavanje gomile:



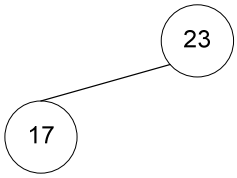
27	17	23	31	47	64	82	93

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

Objektno orijentirano programiranje

Zadatak 1

```
class Matrica {
  private:
    float *elementi;
            int brRedaka;
            int brStupaca;
  public:
    Matrica (int n) {
                  brRedaka = brStupaca = n;
                  elementi = (float*) malloc(n*n*sizeof(float));
    }
            Matrica (int m, int n) {
```

```
                brRedaka = m;
                brStupaca = n;
                elementi = (float*) malloc(n*m*sizeof(float));
    }

            Matrica(const Matrica &mat)
            {
                int i, j;
                brRedaka = mat.brRedaka;
                brStupaca = mat.brStupaca;
                elementi = (float*)
            malloc(brRedaka*brStupaca*sizeof(float));

                for (i=0; i < brRedaka; i++){
                    for (j=0; j < brStupaca; j++){
                        elementi[i*brStupaca + j] =
mat.elementi[i*mat.brStupaca + j];
                    }
                }
            }
    int getbrojRedaka(){
                return brRedaka;
            }
            int getbrojStupaca(){
                return brStupaca;
            }

            float getElement(int i, int j){
                if (i < brRedaka && j < brStupaca)
                    return elementi[i*brStupaca+j];
                else return -1;
            }
            void setElement(int i, int j, float vrijednost){
                if (i < brRedaka && j < brStupaca)
                    elementi[i*brStupaca+j] = vrijednost;
            }

            Matrica Transponirana(){
                Matrica mat(brStupaca, brRedaka);
                int i, j;
                for (i=0; i < brStupaca; i++){
                    for (j=0; j < brRedaka; j++){
                        mat.setElement(i, j, this-
>getElement(j,i));
                    }
                }
                return mat;
            }

    ~Matrica () {
              free(elementi);
      }
};
```

Zadatak 2

```
class Skup
{
private:
     float *_elementi;
     int _brojElemenata;
     int _maxBrojElemenata;
public:
     Skup();
     Skup(int brojElemenata);
     void Dodaj(float element);
     int BrojElemenata();
     int BrojPojavljivanja(float element);
~Skup();
};

Skup::Skup()
{
     _elementi = new float[10];
     _maxBrojElemenata = 10;
     _brojElemenata = 0;
}
Skup::Skup(int brojElemenata)
{
     _elementi = new float[brojElemenata];
     _maxBrojElemenata = brojElemenata;
     _brojElemenata = 0;
}
void Skup::Dodaj(float element)
{
     _elementi[_brojElemenata] = element;
     _brojElemenata++;
}
int Skup::BrojElemenata()
{
     return _brojElemenata;
}
int Skup::BrojPojavljivanja(float element)
{
     int br = 0;
     for (int i = 0; i < _brojElemenata; i++)
          if (_elementi[i] == element) br++;
     return br;
}
Skup::~Skup()
{
 delete [] _elementi;
```

```
	}
```

Zadatak 3

```
int  KolikoUnutar(Kruznica kruz, Tocka poljeTocaka[],int n)
{
	Int Num = 0;
	for( int i=0; i<n; i++ )
	{
		double	x1 = kruz.GetCenterX();
		double	y1 = kruz.GetCenterY();
		double	x2 = poljeTocaka[i].GetX();
		double	y2 = poljeTocaka[i].GetY();

		double	Dist = sqrt((x2 - x1) * (x2 - x1) + (y2
- y1)*(x2 - y1));

		if( Dist < kruz.GetRadius() )
			Num++;
	}

	return Num;
}
```

Stog – objektno orijentirana implementacija

Zadatak 1:

```
Stog *duplikat(Stog *zadani){
	Stog *novi = new Stog();
	Stog pomocni;int element;
	while(!zadani->Prazan()){
		zadani->Skini(&element);
		pomocni.Stavi(element);
	}
	while(!pomocni.Prazan()){
		pomocni.Skini(&element);
		zadani->Stavi(element);
		novi->Stavi(element);
	}
	return novi;
}
```

Zadatak 2:

```
int izbaci_parne(Stog *stog){
	Stog pomocni;int element, br=0;
	while(!stog->Prazan()){
		stog->Skini(&element);
```

```
            if (element % 2)
                  pomocni.Stavi(element);
            else
                  br++;
      }
      while(!pomocni.Prazan()){
            pomocni.Skini(&element);
            stog->Stavi(element);
      }
      return br;
}
```

Zadatak 3:

```
Stog *kopiraj_provjerene(Stog *zadani){
      Stog *novi = new Stog();
      Stog pomocni;int element;
      while(!zadani->Prazan()){
            zadani->Skini(&element);
            pomocni.Stavi(element);
      }
      while(!pomocni.Prazan()){
            pomocni.Skini(&element);
            zadani->Stavi(element);
            if (provjeri(element))
                  novi->Stavi(element);
      }
      return novi;
}
```

Zadatak 4:

```
Stog *spoji(Stog *stog1, Stog *stog2){
      int el1, el2;
      Stog pom1, pom2, *novi;
      novi = new Stog();
      while(!stog1->Prazan() && !stog2->Prazan()){
            stog1->Skini(&el1);
            stog2->Skini(&el2);
            if (el1 < el2){
                  novi->Stavi(el1);
                  pom1.Stavi(el1);
                  //vrati većeg(neiskorištenog) na njegov stog,
                  //tako da opet bude na vrhu
                  stog2->Stavi(el2);
            }
            else{
                  novi->Stavi(el2);
                  pom2.Stavi(el2);
                  stog1->Stavi(el1);
            }
      }
      while(!stog1->Prazan()){
            stog1->Skini(&el1);
            novi->Stavi(el1);
```

```
            pom1.Stavi(el1);
        }
        while(!stog2->Prazan()){
            stog2->Skini(&el2);
            novi->Stavi(el2);
            pom2.Stavi(el2);
        }
        while(!pom1.Prazan()){
            pom1.Skini(&el1);
            stog1->Stavi(el1);
        }
        while(!pom2.Prazan()){
            pom2.Skini(&el2);
            stog2->Stavi(el2);
        }
        return novi;
}
```

Zadatak 5*:

Ideja: Uzeti prvi element na stogu i spremiti ga u x. Zatim ga uspoređivati sa preostalim elementima u stogu. Ukoliko pojedini element nije jednak x sačuvati ga privremeno u pomoćnom stogu `pom1`*. Ukoliko je taj element jednak x, onda označiti da x ima duplikate i taj broj odbaciti (ne čuvati ga u pomoćnom stogu). Nakon prolaska kroz sve elemente stoga, vrijednost duplikat će označavati da li x ima duplikata ili ne. Ako nije imao duplikata, onda x sačuvati premještanje u pomoćni stog* `pom2`*. Sve elemente iz* `pom1` *vratiti u originalni stog i ponoviti postupak. Nakon što postupak bude gotov, tj. na originalnom stogu više ne bude nikakvih elemenata, na stogu* `pom2` *će se nalaziti samo oni elementi koji nisu imali duplikata u stogu te ih treba vratiti na originalni stog.*

```
void izbaci_duple(Stog *stog){
        Stog pom1, pom2;int duplikat=1, x, element;
        while(!stog->Prazan()){
            stog->Skini(&x);
            duplikat = 0;
            while(!stog->Prazan()){
                stog->Skini(&element);
                if (x == element)
                    duplikat = 1;
                else
                    pom1.Stavi(element);
            }
            if (!duplikat)
                pom2.Stavi(x);
            while(!pom1.Prazan()){
                pom1.Skini(&element);
                stog->Stavi(element);
            }
        }
        while(!pom2.Prazan()){
            pom2.Skini(&element);
            stog->Stavi(element);
        }
```

```
}
```

Zadatak 6:

```
void izbaci_broj(Stog *stog, int n, Stog *novi){
	Stog pom;int element, udaljenost=0;
	while(!stog->Prazan()){
		stog->Skini(&element);
		if (element == n)
			novi->Stavi(udaljenost);
		else
			pom.Stavi(element);
		udaljenost++;
	}
	while(!pom.Prazan()){
		pom.Skini(&element);
		stog->Stavi(element);
	}
}
```

Napomena:
Implentacija stoga je mogla biti ponešto drugačija. Primjerice, mogla je izgledati ovako:

```
	Stog::Stog();
	int Stog::Stavi(int element);
	int Stog::Skini(int *element);
```

pri čemu bi funkcija Skini vraćala 1 ukoliko je skidanje elementa sa stoga uspjelo, a 0 ako skidanje nije uspjelo (npr. iz razloga jer je stog bio prazan)
U tom slučaju rješenja zadataka bi bila ponešto drugačija, pa bi primjerice zadatak 6. imao sljedeće rješenje:

```
void izbaci_broj(Stog *stog, int n, Stog *novi){
	Stog pom;int element, udaljenost=0;
	while(stog->Skini(&element)){
		if (element == n)
			novi->Stavi(udaljenost);
		else
			pom.Stavi(element);
		udaljenost++;
	}
	while(pom.Skini(&element)){
		stog->Stavi(element);
	}
}
```

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.

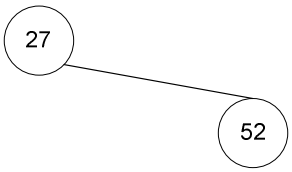
```
cvor *zrcaliStablo(cvor *korijen){
      if (korijen){
            cvor *temp;
            temp = korijen->lijevo;
            korijen->lijevo = zrcaliStablo(korijen->desno);
            korijen->desno = zrcaliStablo(temp);
      }
      return korijen;
}
```

Binarno sortirano stablo i gomila

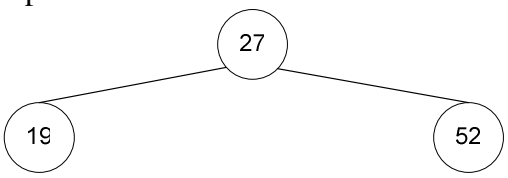
Zadatak 1

Upis elementa 27

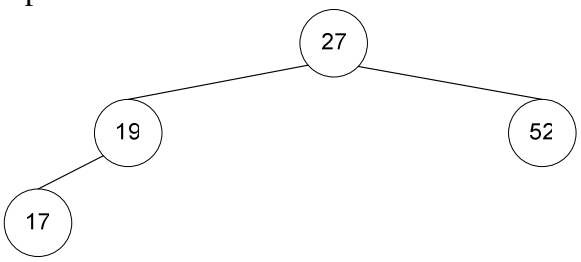
Upis elementa 52



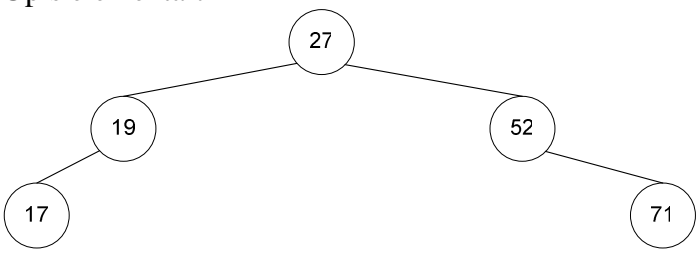
Upis elementa 19



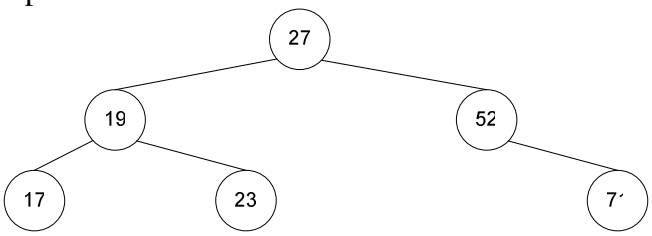
Upis elementa 17



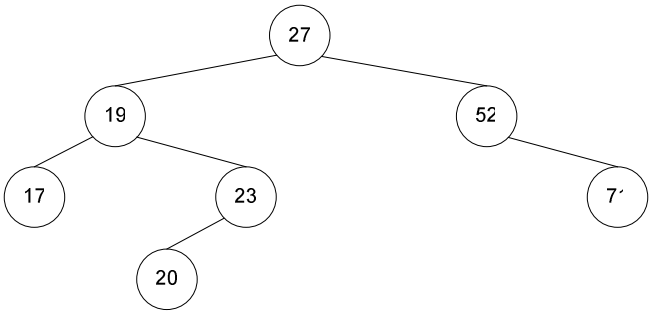
Upis elementa 71



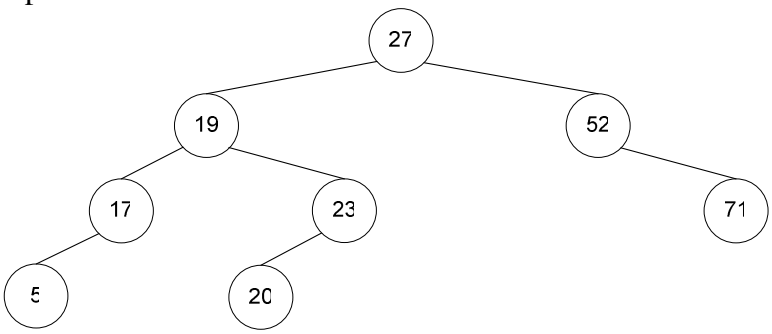
Upis elementa 23



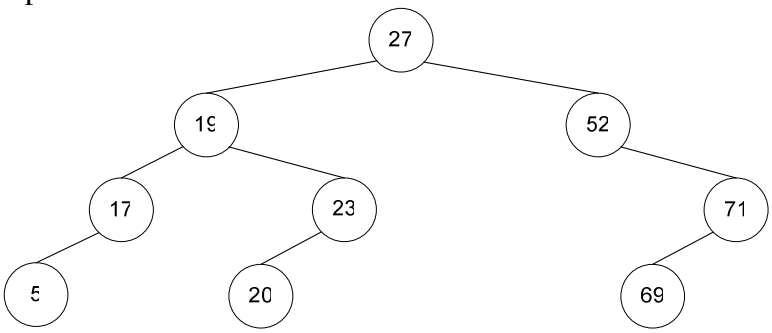
Upis elementa 20



Upis elementa 5



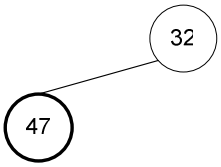
Upis elementa 69

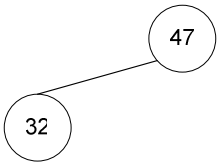


<u>Zadatak 2</u>

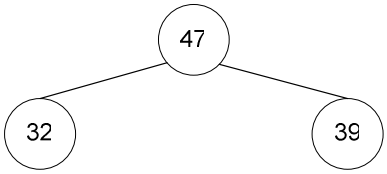
- ubaci 32

- ubaci 47

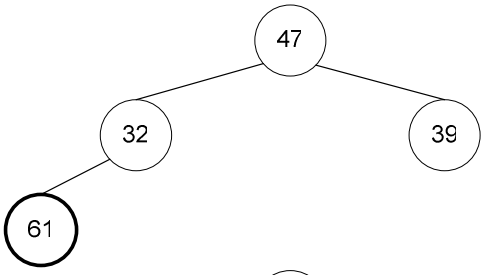


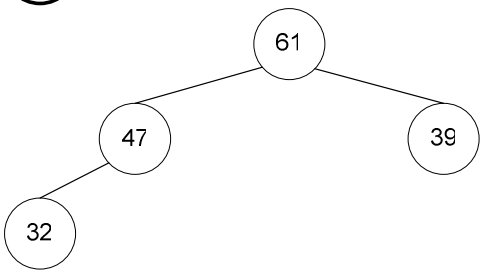


- ubaci 39

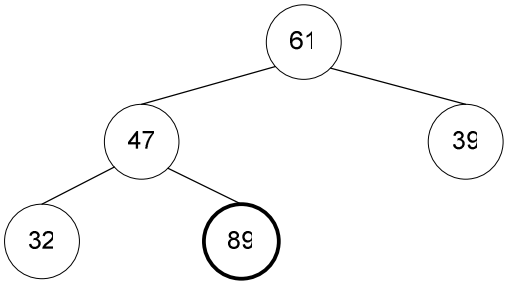


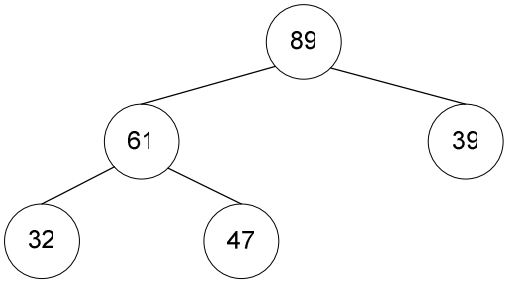
- ubaci 61



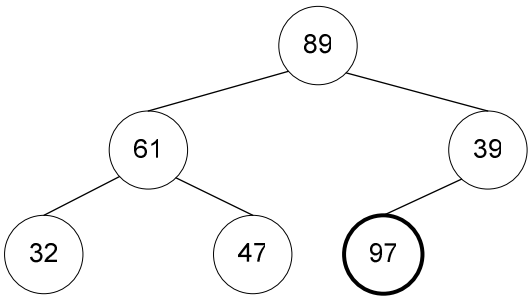


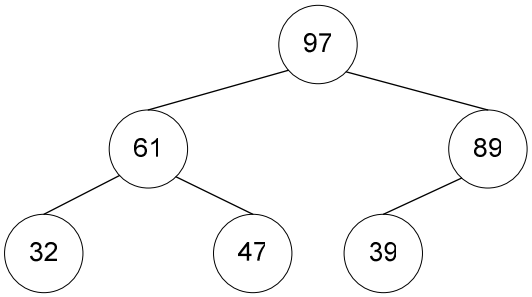
- ubaci 89



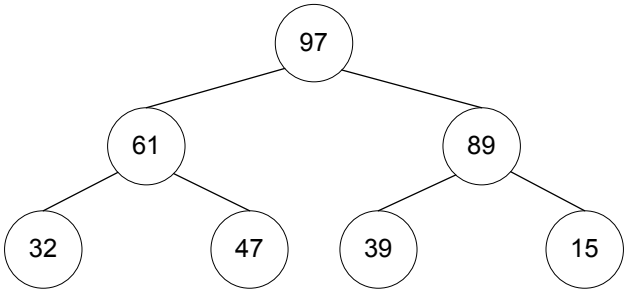


- ubaci 97

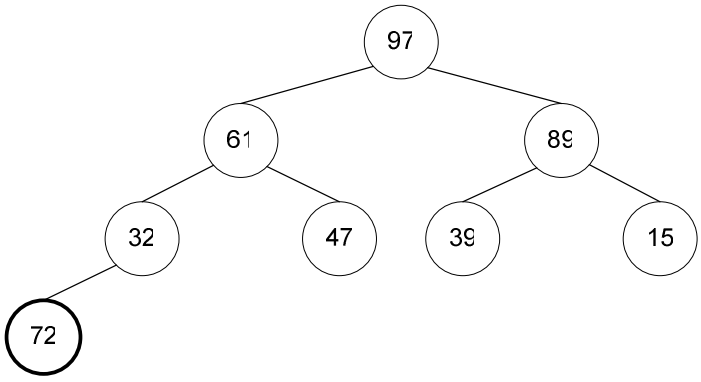




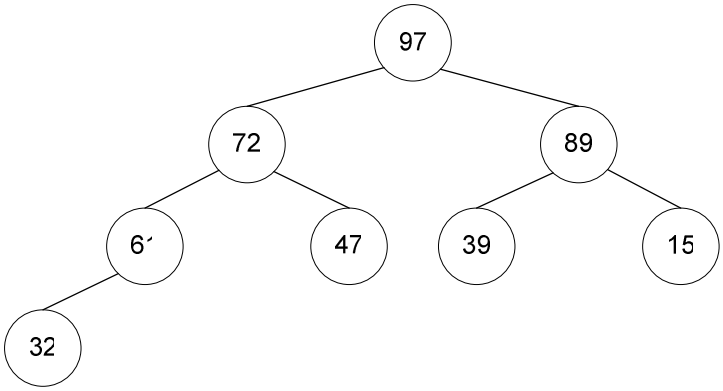
- ubaci 15



- ubaci 72

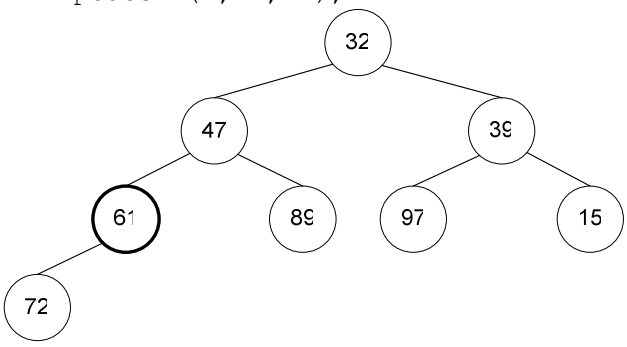


Gomila:

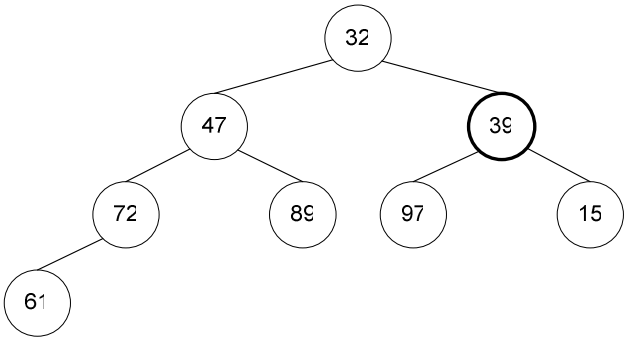


Zadatak 3

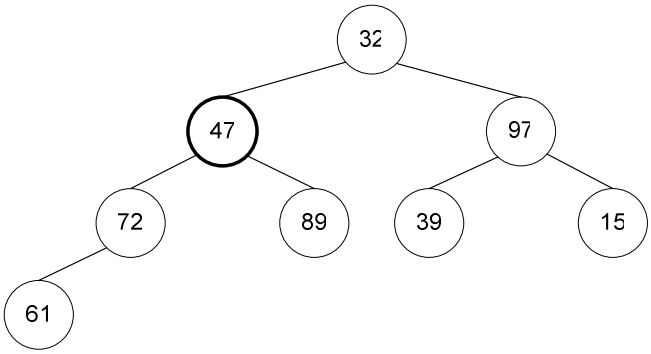
- `i = n/2;  // i = 4`
- `podesi (A, i, n);`



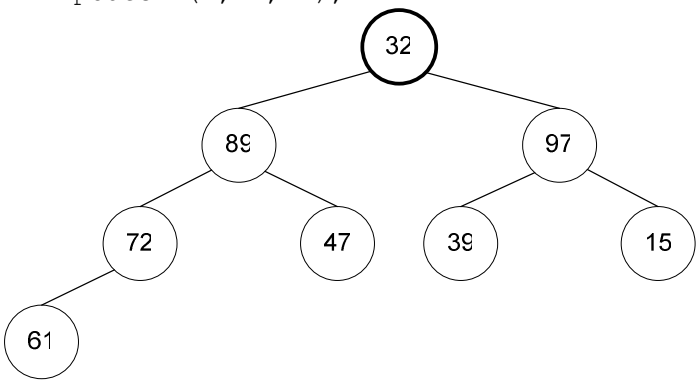
- `i = i-1;  // i = 3`
- `podesi (A, i, n);`



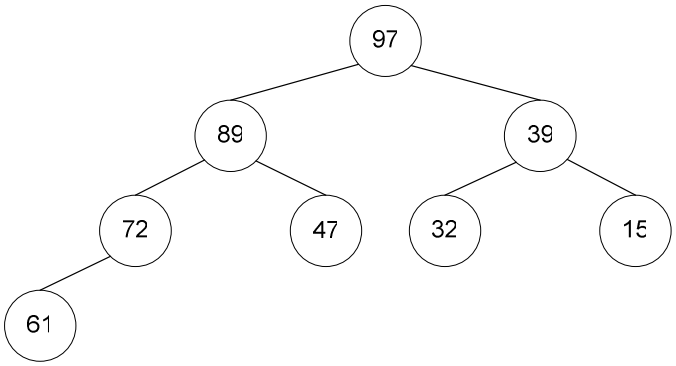
- `i = i-1;  // i = 2`
- `podesi (A, i, n);`



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

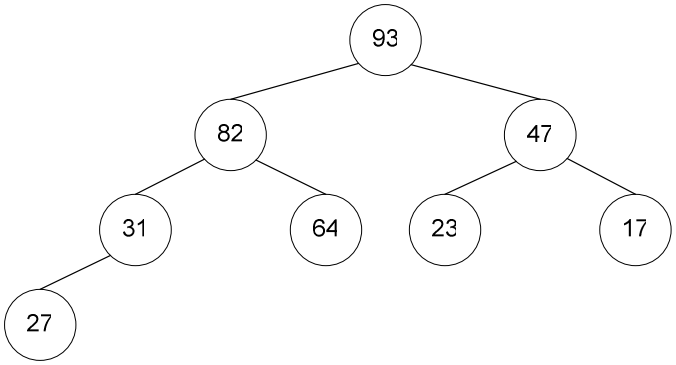


Gomila:



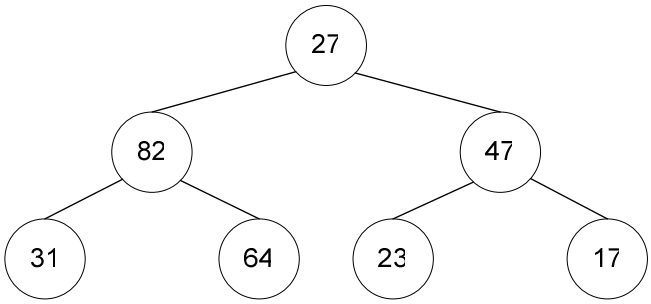
Zadatak 4

Početna gomila:



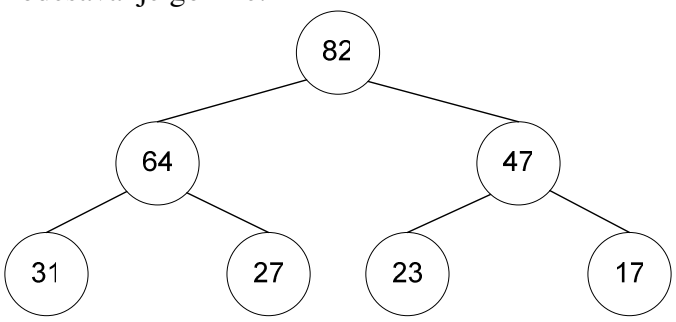
93	82	47	31	64	23	17	27

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



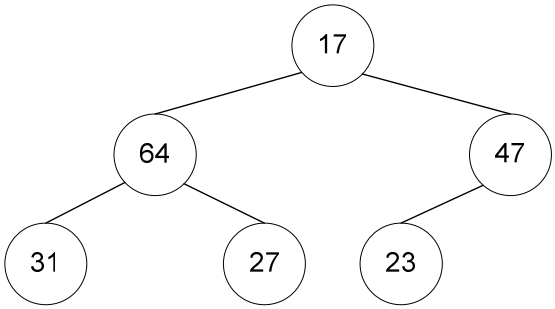
27	82	47	31	64	23	17	93

Podešavanje gomile:



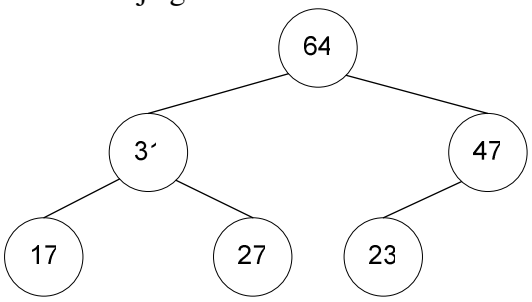
82	64	47	31	27	23	17	93

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



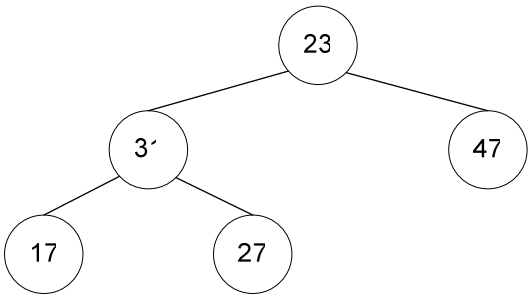
17	64	47	31	27	23	82	93

Podešavanje gomile:



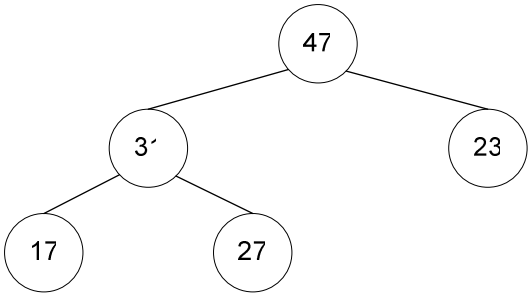
64	31	47	17	27	23	82	93

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



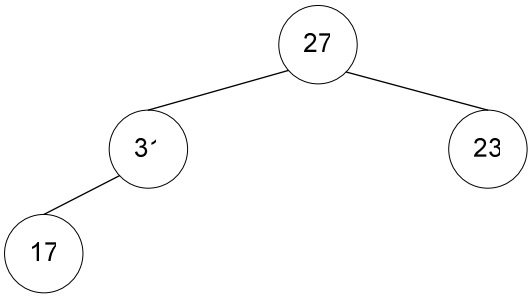
23	31	47	17	27	64	82	93

Podešavanje gomile:



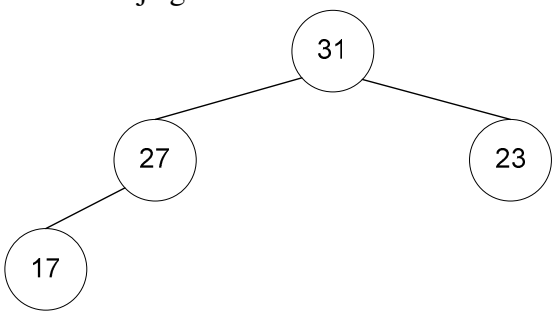
47	31	23	17	27	64	82	93

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



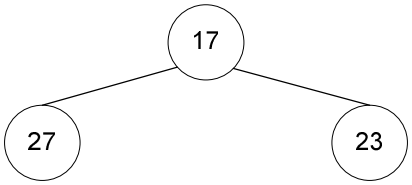
27	31	23	17	47	64	82	93

Podešavanje gomile:



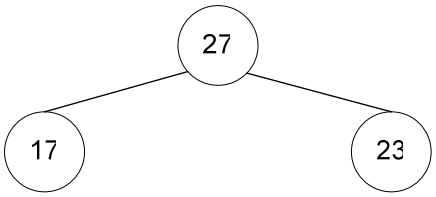
31	27	23	17	47	64	82	93

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



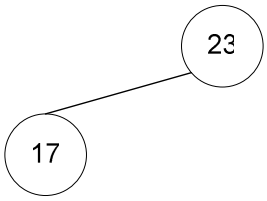
17	27	23	31	47	64	82	93

Podešavanje gomile:



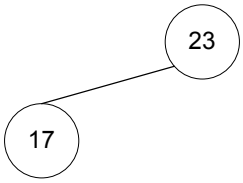
27	17	23	31	47	64	82	93

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