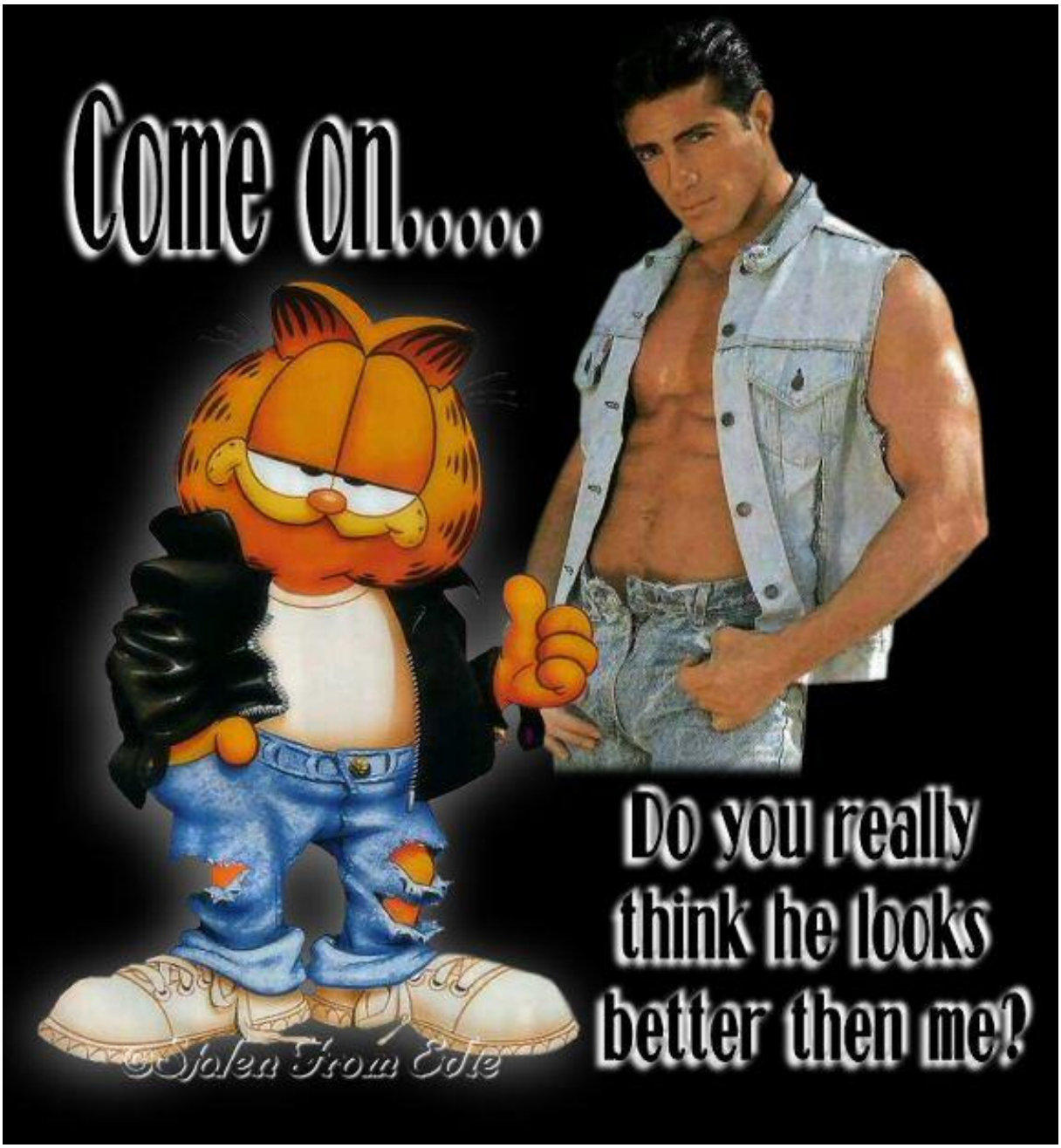
ASP – ZI 2008/09

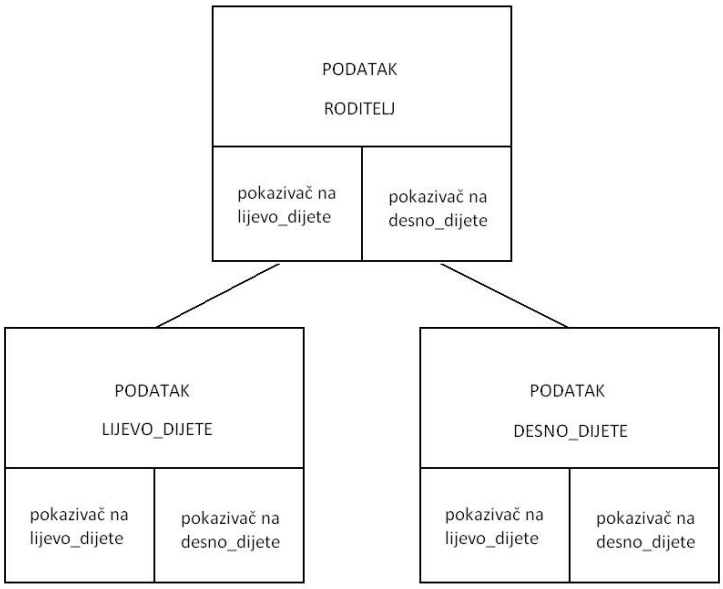
Binarna stabla i gomila



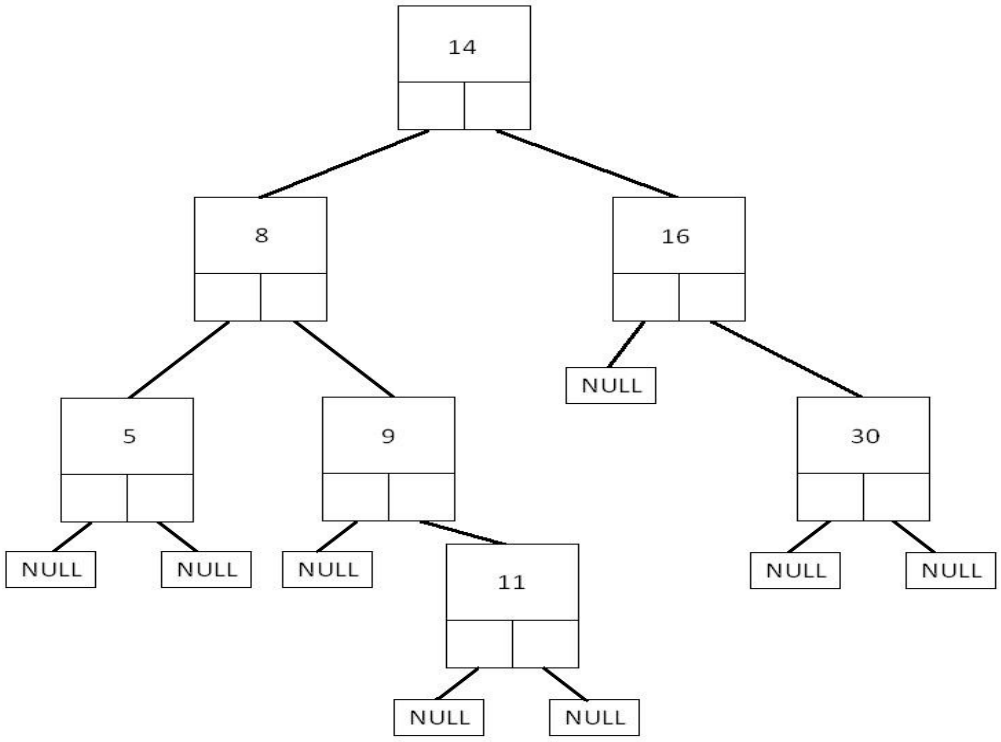
by Vedax

1. BINARNA STABLA

Binarno stablo je takvo stablo koje ima uvijek samo dvije grane koje se nastavljaju iz njega. Vrh stabla, ili početni element stabla naziva se **korijen**. Taj korijen sastoji se od nekog podatka (i obično se naziva **roditelj**) i pokazivača na svoju djecu – **lijevo** i **desno dijete**. Evo slike:



Znači, korijen je roditelj svojoj djeci, lijevo/desno dijete je roditelj svojoj djeci itd., itd.
Taj korijen je samo jedan u nizu **čvorova** koji se mogu pojaviti u binarnom stablu. Recimo da imamo sljedeće binarno stablo:



Svaka ova mala „kućica“ koja se sastoji od podatka i dva pokazivača je zapravo jedan čvor. Opišimo tu strukturu.

```
struct cvor {
        tip podatak; /*imamo podatak */
        struct cvor *lijevo_dijete; /*imamo pokazivač na lijevo dijete */
        struct cvor *desno_dijete; /*imamo pokazivač na desno dijete */
};
```

Znači, opisali smo strukturu **čvor**. Sljedeći problem je umetanje novoga čvora u postojeće binarno stablo. Kao što se iz gornje slike vidi, kada smo na čvor s podatkom 9 dodavali čvor s podatkom 11, najprije smo:

1. trebali **nacrtati „kućicu“ (tj. alocirati memoriju)**,
2. zatim smo **upisali podatak**,
3. i konačno smo stavili da pokazivači na lijevo i desno dijete pokazuju na **NULL** jer nakon 11 nismo umetali novi podatak.

```
struct cvor *NoviCvor (int elem) {
        struct cvor *novi = (cvor *) malloc(sizeof(struct cvor)); /*alociranje memorije za novi čvor */
        novi->podatak =elem; /*upisivanje podatka */
        novi->lijevo = NULL; /*pokazivač na lijevo dijete pokazuje na NULL */
        novi->desno = NULL; /*pokazivač na desno dijete pokazuje na NULL */
        return(novi); /*vraćamo pokazivač na novi čvor */
    }
```

1.1. SORTIRANO BINARNO STABLO

Dvije stvari potrebne za slaganje ovakvog stabla:

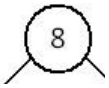
1. s **lijeve** strane idu **manji ili jednaki** elementi u odnosu na roditelja
2. s **desne** strane idu **veći** elementi u odnosu na roditelja

1. Kada dodajemo novi element u stablo, najprije provjeravamo da li je stablo prazno. Ako jest, jednostavno vratimo novi čvor (tu funkciju smo prethodno naveli). Znači napravimo „kućicu“, upišemo podatak (**elem**), i napravimo pokazivače na lijevo i desno dijete, koji pokazuju na NULL.
2. Ako se ne pokazuje na NULL, provjerimo da li je element koji unosimo **manji ili jednak** korijenu stabla. Ako jest, znači da idemo u **lijevi** dio stabla,
3. inače idemo u **desni** dio stabla. Ta dva dijela rekurzivno ponavljamo ulazeći sve dublje u stablo dok opet ne dođemo do NULL pokazivača, gdje umećemo novi element.

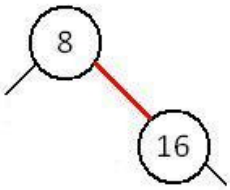
```
struct cvor *dodaj (struct cvor *cvor, tip elem) {
        if (cvor == NULL) { /*ispitujemo da li je stablo prazno, tj. da li se pokazuje na NULL */
                return (NoviCvor (elem)); /*ako jest vraćamo novi čvor */
        }
        else {
                if (elem <= cvor->podatak) /*ispitujemo da li je elem manji ili jednak roditelju*/
                        cvor->lijevo = dodaj(cvor->lijevo, elem); /*ako jest, idemo lijevo */
                else
                        cvor->desno = dodaj(cvor->desno, elem); /*ako nije, idemo desno */
        return(cvor);}
}
```

ZADATAK 1.: Neka imamo zadani niz 8, 16, 32, 4, 1, 0, 23, 17, 5. Od toga treba napraviti sortirano binarno stablo.

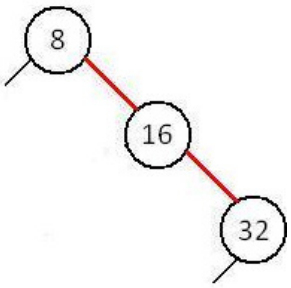
Pa krenimo redom. Najprije upisujemo prvi element niza, 8. to je korijen našeg sortiranog binarnog stabla.



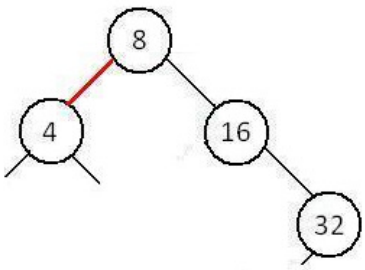
Zatim umećemo sljedeći element, 16. Provjeravamo da li je 16 manji ili jednak, ili je veći od 8. On je **veći** od 8 pa ide na **desnu** stranu.



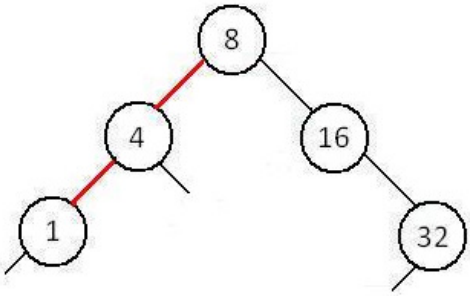
Zatim slijedi 32. Krećemo od korijena. 32 je **veći** od 8 pa idemo **desno**. Dolazimo do 16. 32 je **veći** i od 16 pa opet idemo **desno** i tamo stavljamo naš element.



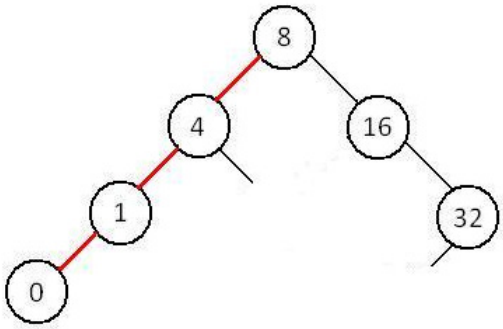
Slijedi 4. Ponovno krećemo od korijena. 4 je **manji** od 8 pa idemo **lijevo**. Tu stavljamo naš element.



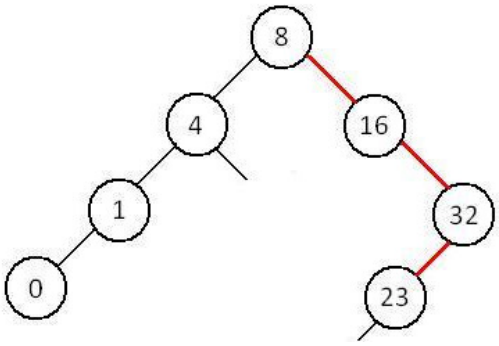
Zatim imamo 1. 1 je **manji** od 8 pa idemo **lijevo**. 1 je **manji** i od 4 i opet idemo **lijevo**. Tu stavljamo naš element.



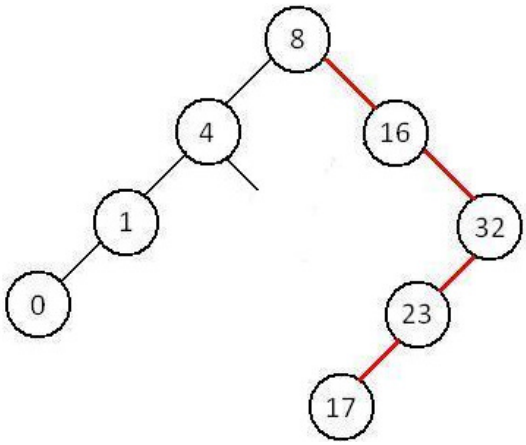
Slijedi 0. 0 je **manja** od 8 pa idemo **lijevo**. **Manja** je i od 4 i opet idemo **lijevo**. Također je **manja** i od 1 te idemo **lijevo** i tu stavljamo 0.



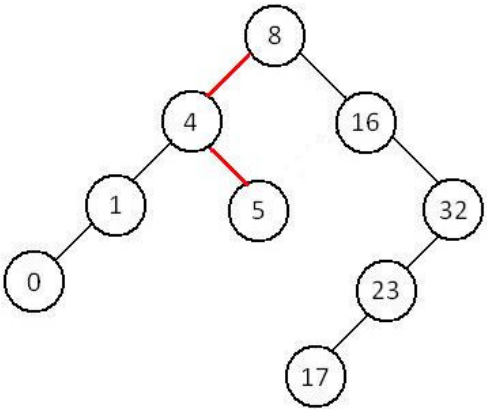
Dolazi nam broj 23. 23 je **veći** od 8 pa idemo **desno**. 23 je **veći** i od 16 pa idemo **desno**. Međutim, **manji** je od 32 i tada idemo **lijevo** i tu umećemo naš element.



Slijedi 17. 17 je **veći** od 8 pa idemo **desno**. **Veći** je i od 16 pa idemo **desno**. Međutim **manji** je od 32 i idemo **lijevo**. I dolazimo do 23. 17 je **manji** od 23 i idemo **lijevo** i tu umećemo naš element.



I konačno, ostao nam je broj 5. 5 je **manji** od 8 pa idemo **lijevo**. I **veći** je od 4 pa idemo **desno** i tu umećemo naš element.



Time je sortiranje gotovo.

Napravimo sada funkcije za traženje elementa po stablu i traženje najmanjeg elementa u stablu.

1. Traženje elementa po stablu. Sve što je potrebno jest da uspoređivati vrijednost koju tražimo sa vrijednošću u trenutnom članu. Ako smo došli do člana koji je NULL (tj. ušli u praznu granu, koja više nema člana), znači da traženi član nije pronađen i vraćamo 0. Ako se, naprotiv, traženi element podudario s trenutnim elementom, našli smo član i vraćamo 1.
Međutim, najčešće se treba putovati dublje u stablo. To se radi tako da rekurzivno ulazimo u stablo i to u granu koja nam odgovara - ako je traženi broj manji od onog koji je u čvoru u kojem smo trenutno, treba otići u lijevu granu, odnosno u desnu ako je veći.

```
int trazi (struct cvor *cvor, int trazeno) {
        if(cvor == NULL) {
                return 0; /*ako nismo uspjeli pronaći element vraćamo 0 */
        }
        else {
                if(trazeno == cvor->podatak) return1; /*ako se traženi element podudara s trenutnim, vraćamo 1 */
                else {
                        if(trazeno < cvor->podatak) /*ako je traženi broj manji od trenutnog, idemo lijevo */
                                return(trazi(cvor->lijevo, trazeno));
                        else return(trazi(cvor->desno, trazeno)); /*inače idemo desno */
                }
        }
}
```

2. Traženje najmanjeg elementa. Ova funkcija će vraćati najmanji član iz sortiranog binarnog stabla. **Najmanji član** je upravo onaj koji se nalazi na **najnižoj razini skroz lijevo**. Iskoristimo tu činjenicu.
Najlakše ostvarenje toga jest da putujemo po stablu samo po lijevoj grani i svaki puta bilježimo sadržaj trenutnog člana sve dok ne dođemo do NULL pokazivača (jer ovdje grana prestaje) te zadnju zapamćenu vrijednost vratimo, a upravo ta vrijednost bit će vrijednost najmanjeg člana.

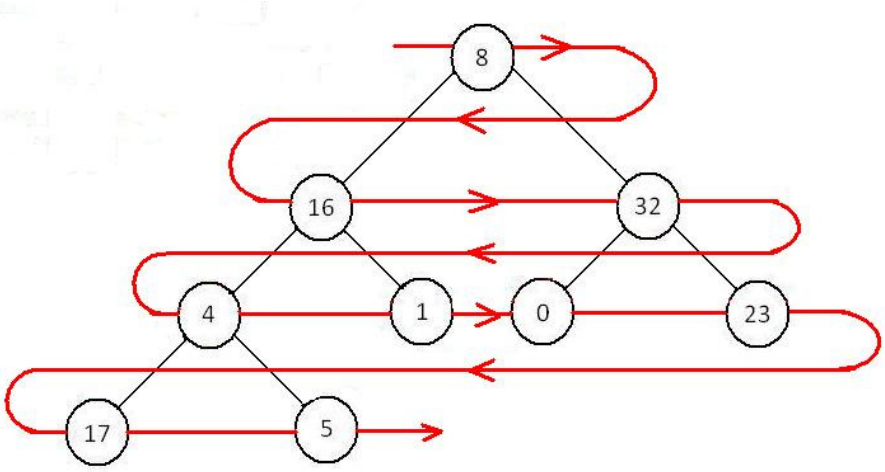
```
int MinClan (struct cvor *cvor) {
        int pom; /*pomoćna varijabla u koju ćemo pamtiti sadržaj trenutnog člana*/
        pom = cvor->elem;
        while (cvor->lijevo != NULL) { /*dok nismo došli do NULL, idemo lijevo */
                cvor = cvor->lijevo;
                pom = cvor->elem; /*pamtimo sadržaj trenutnog člana */
        }
        return pom; /*vraćamo najmanji član */
}
```

2. GOMILA

Gomila je **potpuno binarno stablo** gdje se čvorovi mogu uspoređivati nekom uređajnom relacijom (npr. <=) i gdje je bilo koji **čvor** u smislu te relacije **veći ili jednak od svoje djece** (ako postoje). **Najveći član** mora biti **prvi** član gomile.

ZADATAK 2.: Neka imamo zadani niz 8, 16, 32, 4, 1, 0, 23, 17, 5. Od toga treba napraviti gomilu.

Gomila se popunjava tako da uzmemo prvi element i stavimo ga na početak a svaki sljedeći element upisujemo s lijeva na desno pazeći pri tome da je stablo potpuno.



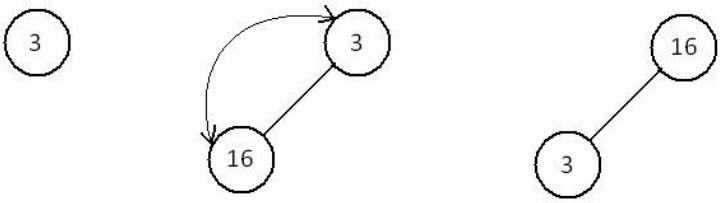
2.1. OBLIKOVANJE GOMILE

2.1.1. SLOŽENOST *O (n log n)*

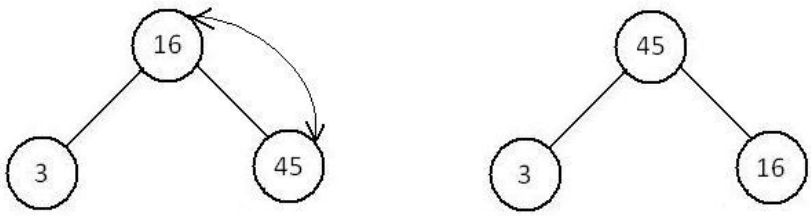
To oblikovanje ilustrirat ćemo preko primjera. Imamo niz 3, 16, 45, 15, 8, 27, 30, 63, 40.

Kod ovakvog oblikovanja, prilikom umetanja svakog elementa automatski se provjerava da li je umetnuti element manji od svoga roditelja, te ako nije treba ga zamijeniti, a nakon toga provjeriti da li je nakon te zamjene grana stabla koju smo dirali također dobro organizirana u smislu da je roditelj uvijek veći ili jednak od svoje djece.

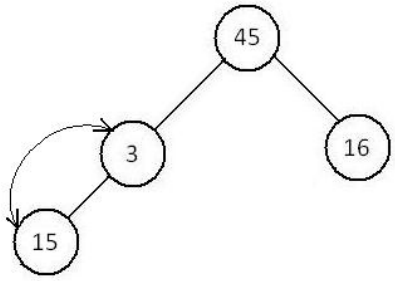
Pa krenimo redom. Najprije ubacimo 3. zatim slijedi 16 kojeg upisujemo s lijeve strane. 16 je dijete od 3, ali kako roditelj mora biti veći od svoje djece, ta dva elementa zamijenimo.

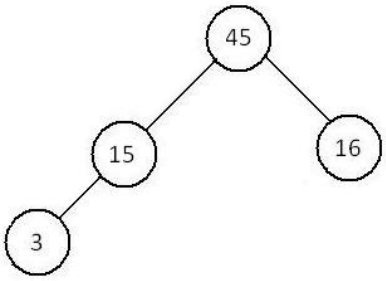


Zatim slijedi 45 koji umećemo s desne strane od 16. 45 je dijete od 16, no opet vidimo da, kako je roditelj veći od djece, moramo zamijeniti ta dva elementa.

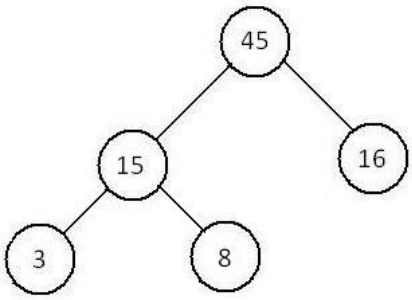


Dalje dodajemo 15. Po pravilu za stvaranje gomile stavljamo ga s lijeve strane od 3. Opet prema pravilu djeca – roditelji moramo zamijeniti 3 i 15. Eh sad, pošto smo dirali lijevu granu moramo ju čitavu provjeriti. 45 je roditelj od 15 i uočavamo da je veći od 15 što je dobro. I također, već smo popravili redoslijed 3 i 15 i zaključujemo da je ta grana u redu.

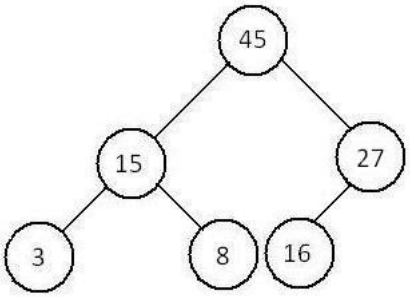




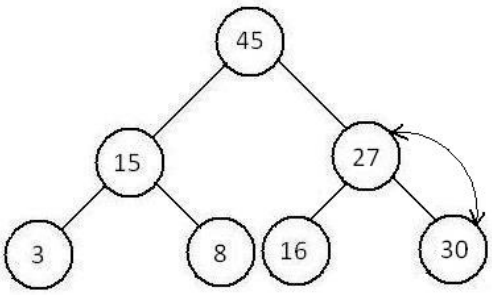
Idemo dalje. Sada je na redu 8. Njega sada stavljamo s desne strane od 15. 8 je manji od 15, odnosno od svoga roditelja i to je u redu.

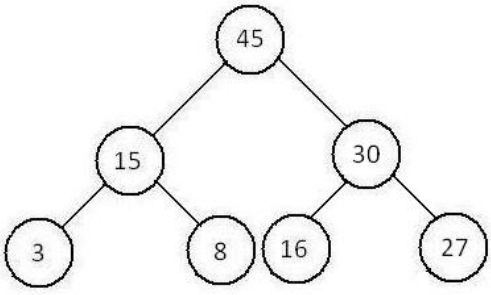


Zatim dodajemo 27. Dodajemo ga s lijeve strane od 16. Po pravilu dijete – roditelj vidimo da tva elementa moramo zamijeniti. Također, kako smo dirali desnu granu, moramo provjeriti da li su sva djeva manja ili jednaka svojim roditeljima na toj strani. Uočavamo da je sve u redu i prelazimo na dodavanje sljedećeg elementa.

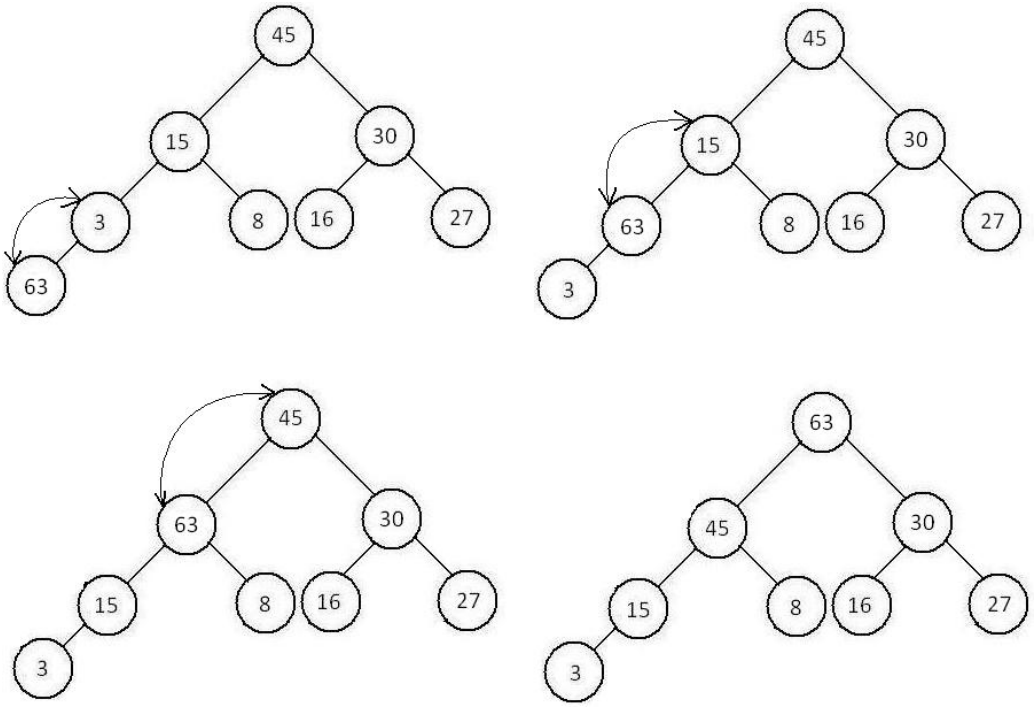


Sljedeći element je 30. Njega dodajemo s desne strane od 27. Ta dva elementa po pravilu dijete – roditelj moramo zamijeniti. I opet, kako smo dirali desnu granu, moramo ju čitavu provjeriti. Sve je u redu i prelazimo na dodavanje sljedećeg elementa.

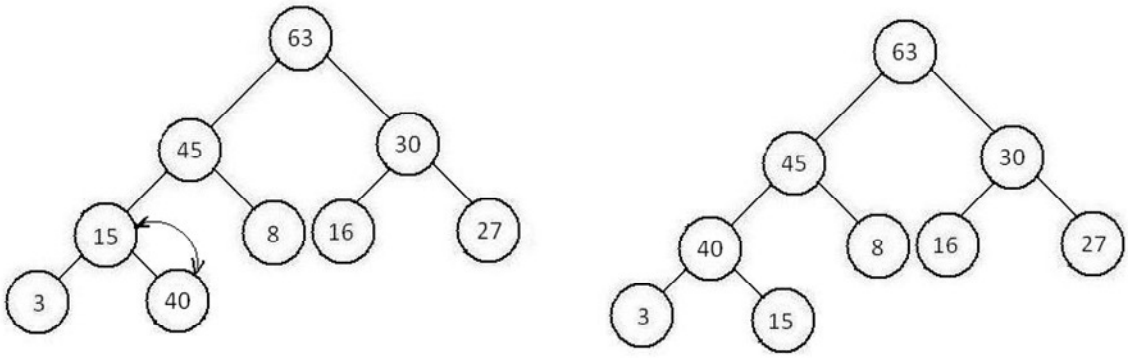




Na redu je 63. Stavljamo ga s lijeve strane od 3. Uočavamo da ta dva elementa moramo zamijeniti. Kako smo dirali lijevu granu moramo ju provjeriti. Uočavamo da je 63 veći od svog roditelja, 15, pa ih također moramo zamijeniti. Daljnjom provjerom vidimo da je 63 veći i od svog sljedećeg roditelja, 45, pa i njega zamijenimo sa 63. Daljnjom provjreom uočavamo da je sve ostalo u toj grani u redu i idemo na dodavanje sljedećeg elementa.



Sljedeći element, a ujedno i zadnji kojeg moramo dodati, je 40. Njega najprije dodajemo s desne strane od 15. prema već nekoliko puta navedenom pravilu dijete – roditelj, zamjenjujemo 40 i 15. Kako smo dirali tu granu ponovno moramo provjeriti da li je sve ostalo u tom dijelu u redu. Uočavamo da je 40 manji od svog novog roditelja 45 i time završavamo oblikovanje gomile.



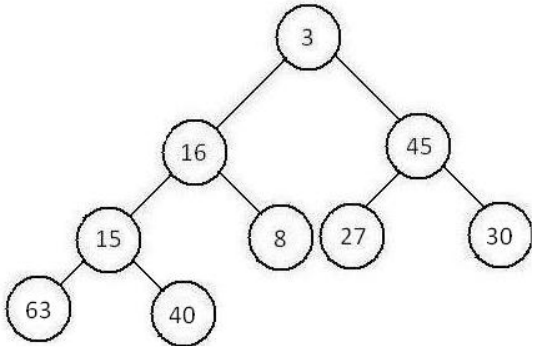
2.1.2. SLOŽENOST *O (n)*

To oblikovanje ilustrirat ćemo također preko primjera s nizom 3, 16, 45, 15, 8, 27, 30, 63, 40.

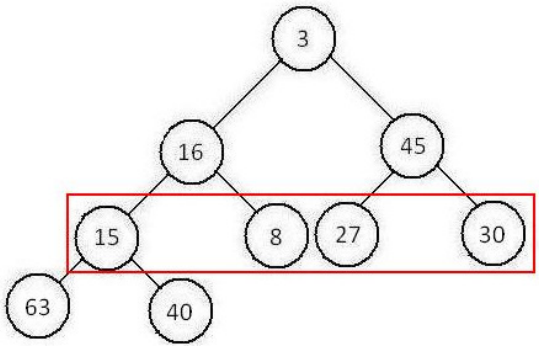
Stvaranje gomile algoritmom ove složenosti ostvaruje se tako da se prvo složi potpuno binarno stablo (kao u zadatku 2.), a nakon toga se kreće od predzadnje razine te se, član po član, s desna, gledaju djeca tih članova. Ako je koje dijete (ili oba) veće od člana kojeg gledamo, zamjenjujemo ih, te također gledamo dublje u granu treba li još što zamijeniti. Nakon što smo prošli sve članove neke razine, krećemo na visu razinu.

Pa krenimo.

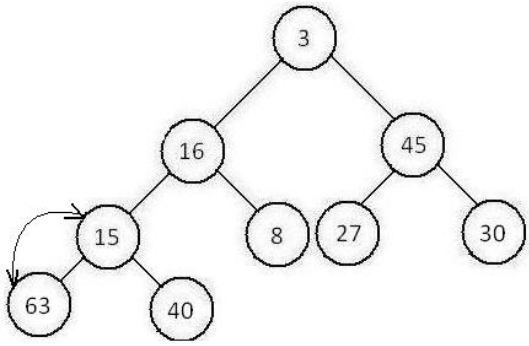
Najprije napravimo potpuno binarno stablo.

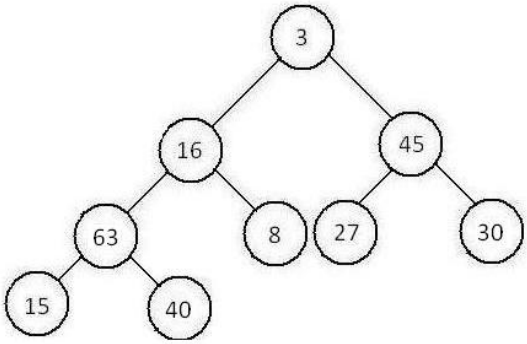


Zatim gledamo predzadnju razinu kao na slici.

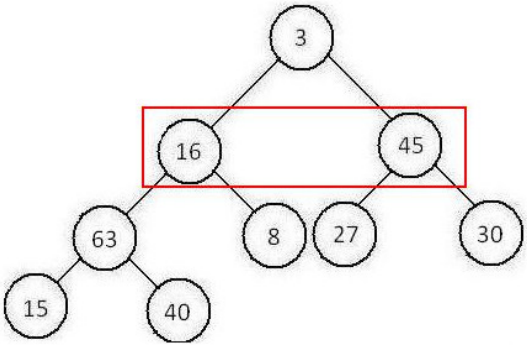


Provjeravanje se vrši s desne strane. Kako 30, 27 i 8 (redom s desna) nemaju djece, onda provjravamo za 15. Uočavamo da su obadva djeteta od 15, i 63 i 40, veći od svog roditelja, međutim, kako je 63 veći od 40, onda zamjenjujemo samo 15 i 63. Tu ne trebamo provjeravati granu jer su to zadnji elementi i uspješno smo izvršili zamjenu.

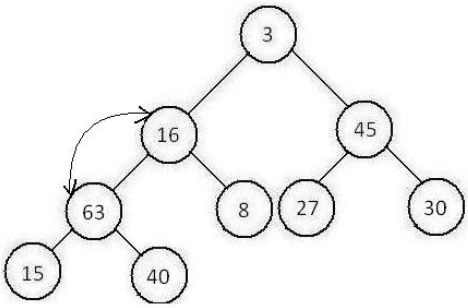


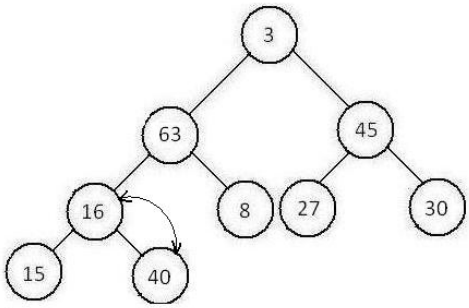


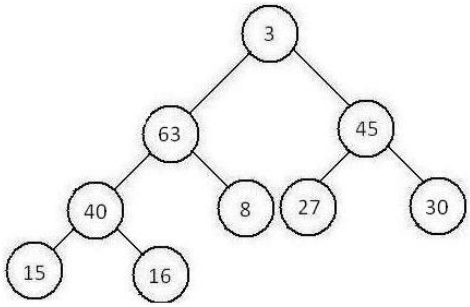
Prelazimo na sljedeću razinu.



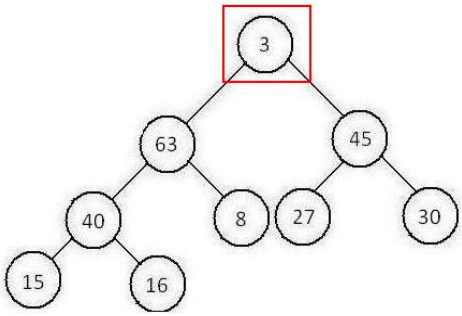
Krećemo s desna. Vidimo da su djeca od 45, 27 i 30, manja od svog roditelja pa prelazimo na 16. Tu uočavamo da je 63 veći od 16 i trebamo ih zamijeniti. Također, trebamo pogledati da li se on 16 nalazi na pravom mjestu. Vidimo da ga moramo zamijeniti sa 40 jer je on veći od njega.



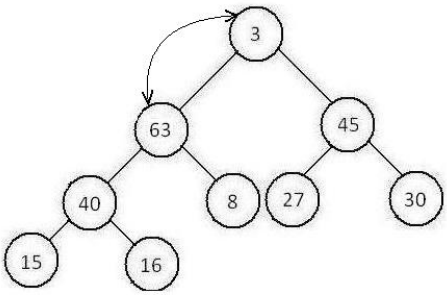


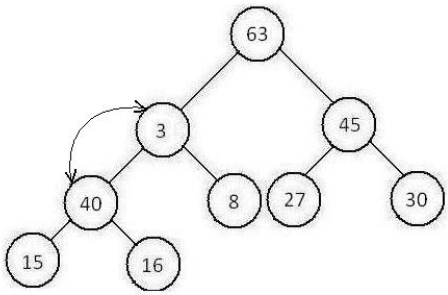


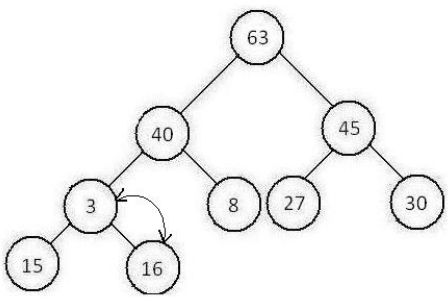
I konačno, dolazimo na posljednju razinu.

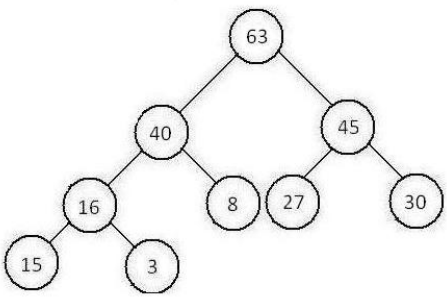


Vidimo da su i 63 i 45 veći od 3, ali kako je 63 veći od 45, zamjenjujemo samo 3 i 63. Nakon toga treba popraviti granu koju smo dirali. Uočavamo da 3 nije na dobrom mjestu jer su i 40 i 8 veći od njega, no opet, kako je 40 veći od 8, mijenjamo samo 40 i 3. i konačno dolazimo do zadnjeg dijela gdje vidimo da su i 15 i 16 veći od 3, ali opet, jer je 16 veći od 15, zamjenjujemo samo 3 i 16. Time je oblikovanje gomile završeno.



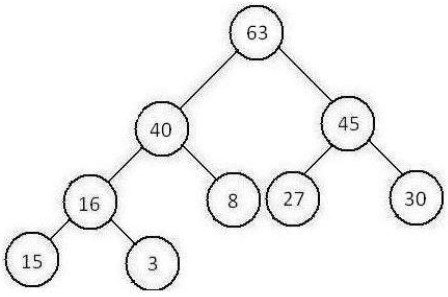






2.2. SORTIRANJE GOMILOM (HEAP SORT)

Objasnit ćemo ovo preko gornjeg primjera, ali samo određeni dio jer je ovo zaista lagano. Uzimamo, potpuno oblikovano binarno stablo s kojim smo završili.



Tada nam je ovo ulazni niz:

63	40	45	16	8	27	30	15	3

Oblikovanje gomile radit ćemo **odzgo prema dolje**.

Znači, najprije uzimamo najviši član, 63. Njega stavimo u novo, sortirano polje, a izbrišemo iz starog.

Stari niz:

	40	45	16	8	27	30	15	3

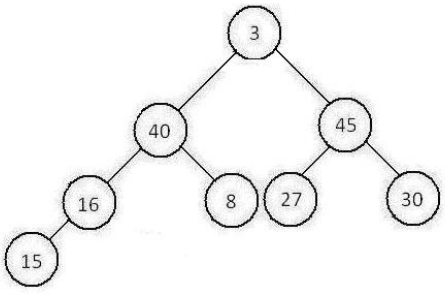
Sortirani niz:

63								

Nakon toga, zadnji član stabla, u ovom slučaju 3, stavljamo na mjesto izbrisanog elementa.

Stari niz:

3	40	45	16	8	27	30	15



Krećemo odozgo prema dolje. I 40 i 45 su veći od 3, ali na najviše mjesto stavljamo 45 jer je veći od 40 (zamijenimo 45 i 3).

Stari niz:

45	40	3	16	8	27	30	15

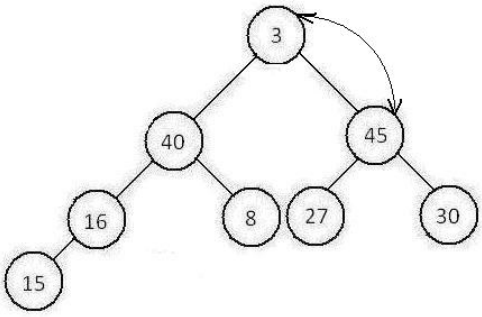
Nakon toga brišemo najviši član, 45, dodajemo ga u sortirani niz poslije 63, a zadnji element, 15, dolazi na prazno mjesto.

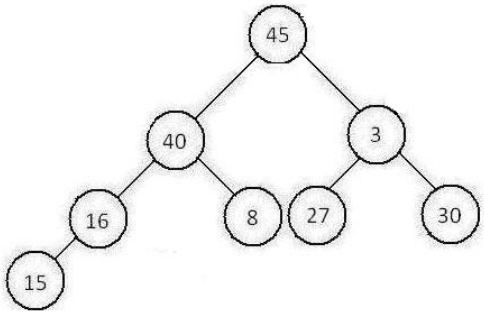
Stari niz:

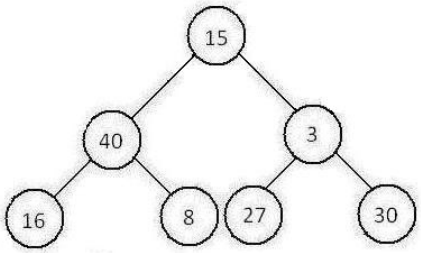
15	40	3	16	8	27	30

Sortirani niz:

63	45							







Zatim ponovno, krenuvši odzogo, oblikujemo gomilu. Od 40 i 3, 40 je veći od 15 pa ta dva člana zamijenimo.

Stari niz:

40	15	3	16	8	27	30

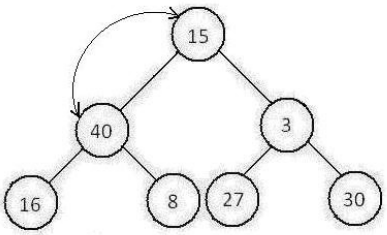
Zatim 40 brišemo i stavljamo u novi, sortirani niz poslije 45, a na prazno mjesto dolazi zadnji element, odnosno 30.

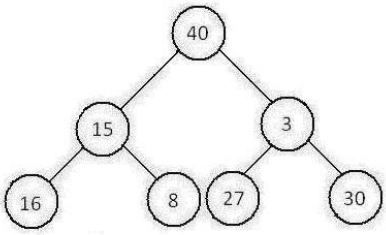
Stari niz:

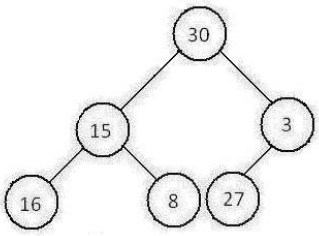
30	15	3	16	8	27

Sortirani niz:

63	45	40						







Evo već imamo 3 člana, i mislim da je bit sortiranja shvaćena.

Sretno na MI!!! ☺