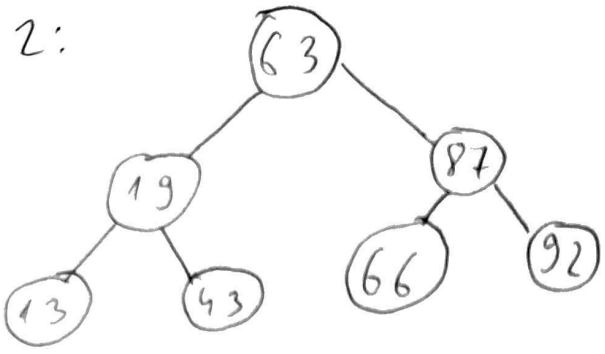
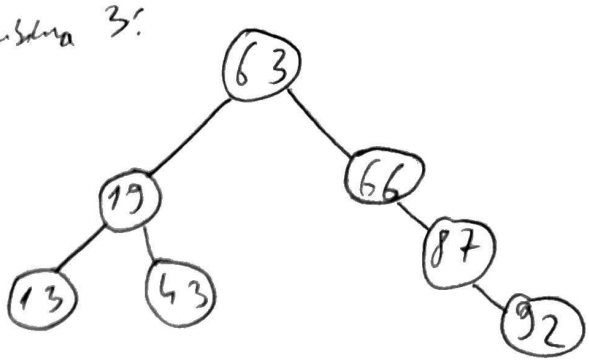
DZ 3

1.

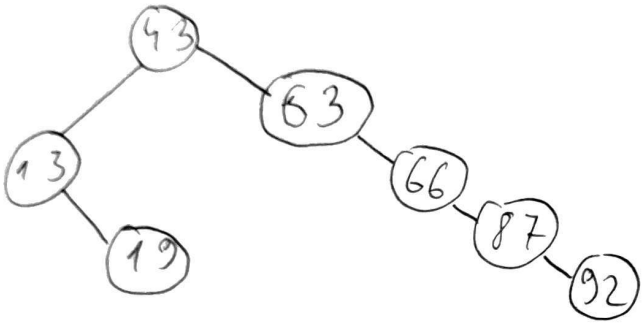
stablo 2:



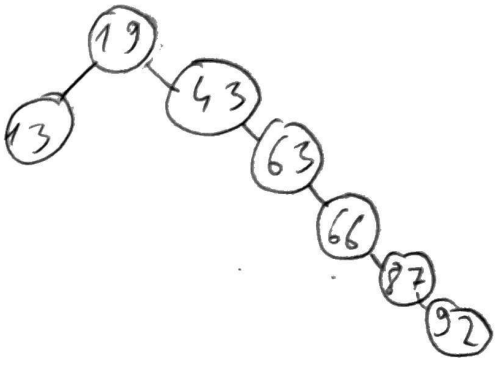
stablo 3:



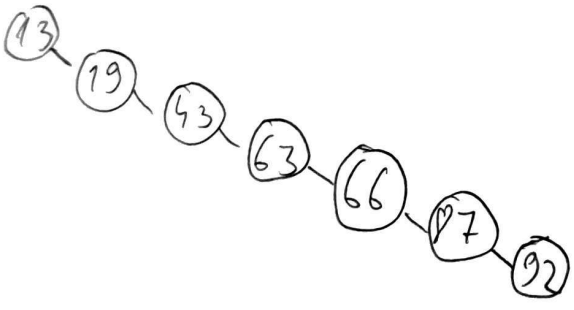
stablo 4:



stablo 5:



stablo 6:



2. y mora biti predak od x jer mu je successor (svojstvo BST-a, ako pretpostavimo suprotno i uzmemo z kao prvog zajedničkog pretka x i y slijedi da je $x < z < y$, pa y nije successor). Osim toga, y.left mora biti predak od x jer da nije, predak od x bio bi y.right pa bi vrijedilo $x > y$. Sada pretpostavimo da y nije najniži predak od x čije je djelete isto predak od x. Tada bi taj najniži predak z bio u lijevom podstablu od y pa bi vrijedilo $z < y$, što ne može biti jer je y successor od x. ↓ ■

(3.)

Ako je x lijevo dijete od y, u algoritmu successor while petlja se neće izvršiti nijednom (jer je $x == par.left$) pa će se vratiti y.
Ako je desno dijete, while petlja u algoritmu predecessor neće se izvršiti nijednom (jer je $x == par.right$) pa će se vratiti y. Dakle, točno jedna od te dvije tvrdnje biti će istinita. ■

(4.)

Neka je x node na kojem počinjemo zvati TREE-SUCCESSOR i y k-ti successor od x (na y završavamo). Udaljenost između ta dva čvora je maksimalno $2h$ i sve veze koje spajaju tih k čvorova su posjećene dva puta (tako funkcionira alg. za successor), pa treba $\leq 2k + 2h$ vremena, tj. $O(k+h)$. ■

(5.)

Neka je x proizvoljni čvor u stablu. Kada se pozove successor na $x.p$ veza između $x.p$ i x se koristi. Opet se koristi kod poziva successor na najveći element u podstablu kojem je korijen x. Još jednom se može koristiti kada pozivamo minimum stabla, no to su jedina 3 puta kada se ta veza može koristiti, pa znamo da je runtime $\leq 3n = O(n)$. ■

(6.)

```
TREE-INSERT(T, n)
    x = T.root
    y = NIL
    while x ≠ NIL
        y = x
        if n.key < x.key
            x = x.left
        else x = x.right
    n.p = y
    if y == NIL
        T.root = n
    else if n.key < y.key
        y.left = n
    else y.right = n
```

```
TREE-SEARCH(x, k)
    while x ≠ NIL and k ≠ x.key
        if k < x.key
            x = x.left
        else x = x.right
    return x
```

Po prethodnoj tvrdnji vidimo da je broj ispitanih čvorova u pretraživanju za 1 veći jer ispituje i čvor koji traži, što se u umetanju ne radi. ■

7.

```
TREE-INSERT(T, z)
1.  x = T.root
2.  y = NIL
3.  while x ≠ NIL
4.      y = x
5.      if z.key < x.key
6.          x = x.left
7.      else x = x.right
8.  z.p = y
9.  if y == NIL
10.     T.root = z
11. else if z.key < y.key
12.     y.left = z
13. else y.right = z
```

```
INORDER-WALK(T, x)
if x == NIL
    return;
inorder-walk(x.left)
do something
inorder-walk(x.right)
```

WORST CASE:
Kada umetnemo listiće u sortiranom poretku. Kada će visina stabla dostići n, pa će se while petlja na listi 3 izvršiti n puta. Ukupno vrijeme izvršavanja bit će $O(n^2)$ jer je u tom slučaju BST praktički linked lista s n čvorova. U tom slučaju, inorder-walk posjetit će čvorove u istom redoslijedu u kojem su bili insertani; pa će broj usporedbi biti proporcionalan sumi prvih n brojeva $\frac{n(n+1)}{2}$, tj. $O(n^2)$. Osim toga, insert poziva n puta a while petlja će se uvijek izvršiti n puta pa je insertanje $O(n^2)$.

(jer e x biti NIL tek nakon n poziva x = x.right ili x = x.left)

BEST CASE:
Kada je BST balansiran. Tada visina stabla ne premašuje $O(\log n)$. Pošto će se tada while petlja na listi 3 insertu izvršiti najviše $\lceil \log n \rceil$ puta, a insert također n puta, algoritam je $O(n \log n)$ best case.

(jer u while petlji imamo provjeru pa idemo ili lijevo ili desno, a pošto je visina stabla logn onda e se while petlja izvršiti O(logn) puta.

8.

Nije, npr. za: