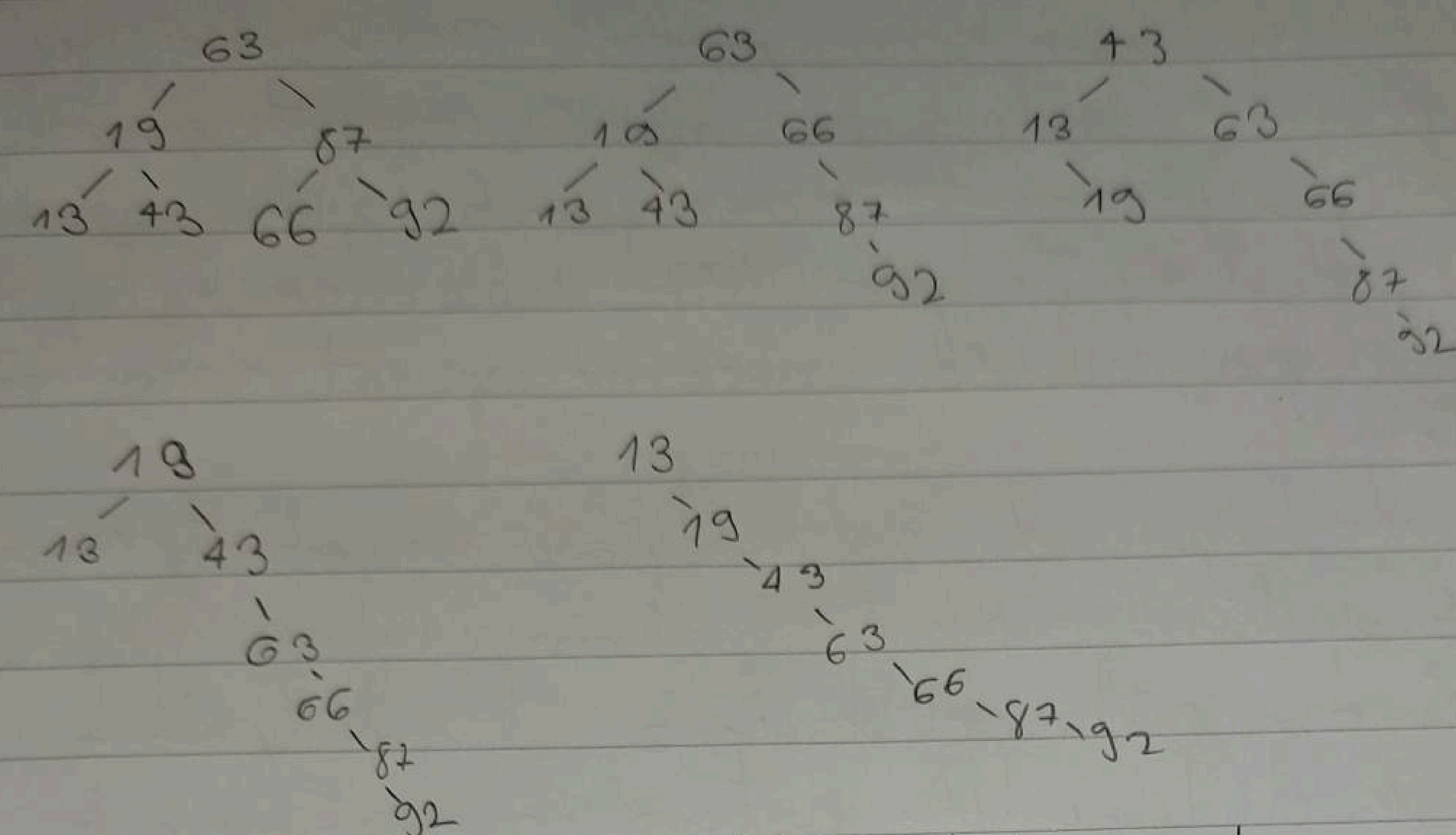
4. neka je čvor na kojem smo pozvali TREE-SUCC, a y k-ti sljedbenik x, neka je p najniži zajednički predak x i y. Uzastopni pozivi tako neće preći jednu granu stabla više od dva puta jer TREE-SUCCESSOR djeluje kao obilazak stabla, pa nikada nećemo ispisati vrh više od tri puta

osim toga bilo koji vrh čija se ključna vrijednost ne nalazi x i y bit će ispitivan najviše jednom i pojavljuje se na jedinstvenom putu od x do z ili od y do z budući da su duljine tih puteva ograničene s h

$3k + 2h = O(k + h)$

5 neka je x čvor u stablu, pozivom na successor x.right uzima se veza između x.right i x također se koristi kada pozovemo sucesor na najveći elem u podstablu u pronalasku minimuma još se može iskoristiti ta veza te se ta veza jednom 3 puta može iskoristiti $\leq 3n = O(n)$

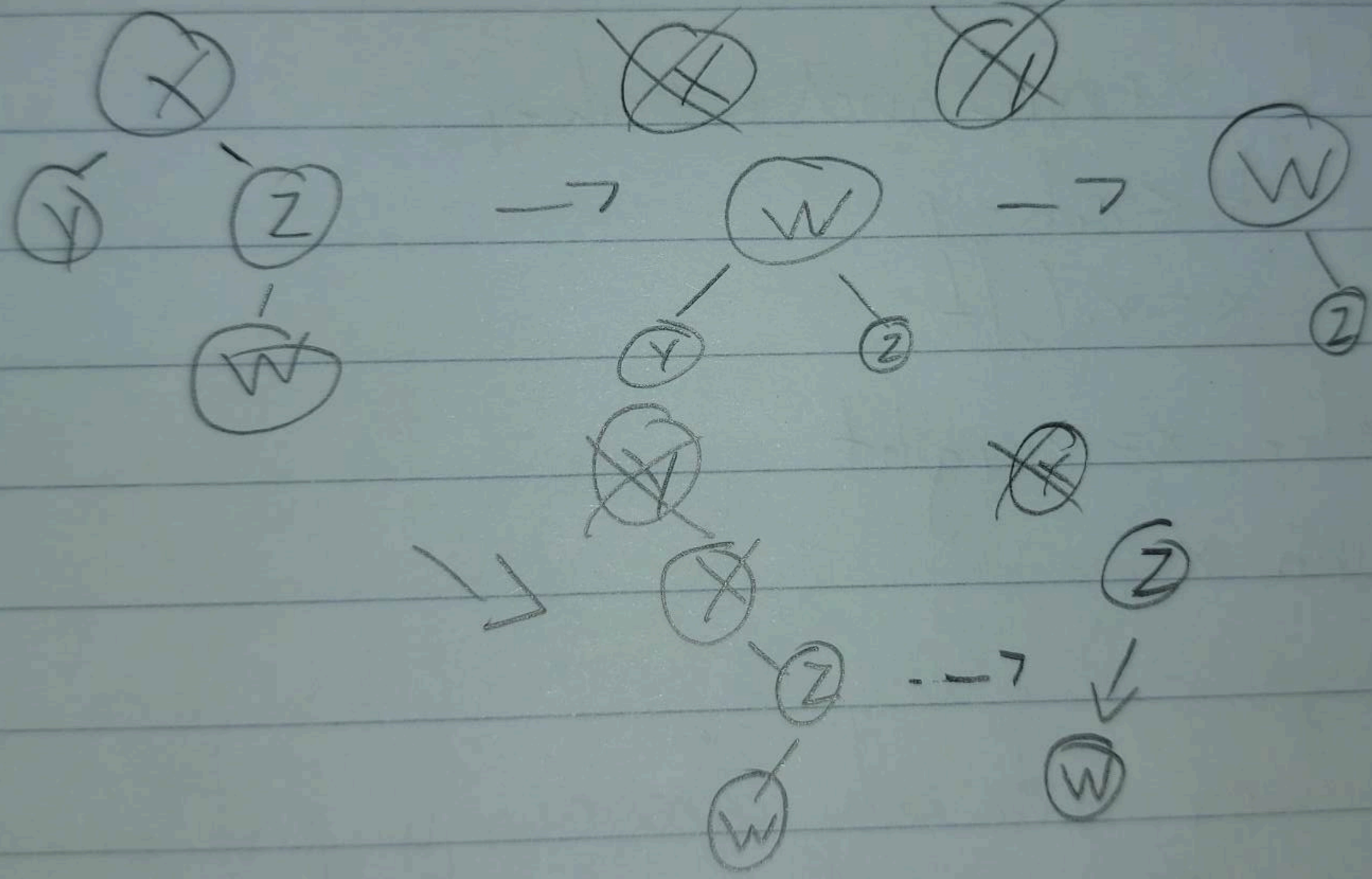
1.



2. Prvo utvrdujemo da y mora biti predak x, ako y nije predak x
tada neka z označava prvog zajedničkog pretka x i y, prema
svojstvu BTS, x < z, y → y ne može biti nasljednik x
y.left mora biti predak x jer manji bi y.right bio predak
x što implicira x > y.
Pretpostavimo da y nije najniži predak x čije je lijevi
dijete predak x neka z označava onog najnižeg
pretka, tada se z mora nalaziti u lijevom podstablu
y, što implicira z < y, što je suprotno s činjenicom
da je y nasljednik x

3. ako je x = y.left tada će pozivanje successora za x
rezultirati bez iteracije petlje while
ako je x = y.right petlja dok je za pozivanje predecessora
x neće završiti te će se vratiti y
te y je ili prethodnik ili sljedbenik x (samo jedan od te

8.



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.key
if y == NIL
    T.root = n
else if n.key < y.key
    y.left = n
else y.right = 2
```

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

broj ispitanih čvorova u pretraživanju veći je za 1 jer ispituje i čvor koji traži (insert to ne radi)

7.

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

WORST CASE

- ako ubacujemo ključeve u sortiranom poretku tada će visina stabla biti n pa će se while petlja izvršiti n-puta te će ukupno vrijeme biti O(n²)
- inorder-walk - će ići istim redoslijedom u kojem su čvorovi bili insertani pa će broj poziva biti $\frac{n(n+1)}{2} \Rightarrow O(n^2)$

BEST-CASE

- ako je BST balansirano visina stabla neće preći O(lg n) tada će se while petlja izvršiti ⌈lg n⌉ puta a insert n puta O(n lg n)