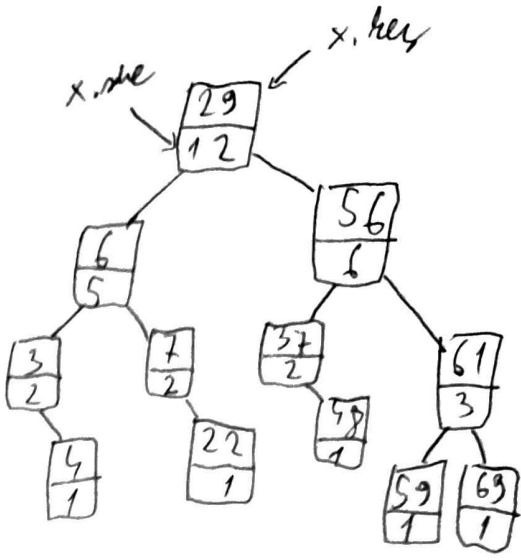
DZ 6

1.

1. key = 29, size = 12
   key = 6, size = 5
   key = 3, size = 2
   key = 4, size = 1
   key = 7, size = 2
   key = 22, size = 1
   key = 56, size = 6
   key = 37, size = 2
   key = 48, size = 1
   key = 61, size = 3
   key = 59, size = 1
   key = 69, size = 1



2.

```
OS-SELECT(x, i)
1. r = x.left.size + 1
2. if i == r
3.     return x
4. else if i < r
5.     return OS-SELECT(x.left, i)
6. else return OS-SELECT(x.right, i - r)
```

OS_SELECT prvo stavlja r na size od lijevog podstabla od x + 1 za x, ako je i = r dobivamo x i vracamo to kao i-ti cvor po redu u stablu. Ako je i < r, želimo pronaci i-ti najmanji element u T(x.left) pa rekurzivno idemo po x.left. Inace, i-ti najmanji element nalazi se u T(x.right) ali odbacimo r elemenata u T(x) koji prethode elementima u T (x.right). Na kraju dobijemo i-ti najmanji cvor u stablu T. Ako primijenimo to na T.root i 8 kao u zadatku, vidjet cemo da je r = 6, a 8 > 6 pa idemo u desno podstablo (sada je i = i - r, tj. 2). Sada je i < r, tj. 2 < 3 pa idemo u lijevo podstablo. Ovdje je r = 1 a i i = 2, pa idemo u desno podstablo s i = i - r = 1. Sada je i = r = 1 pa imamo cvor s kljucem 48 kao rjesenje.

3.

```
OS-RANK(T, x)
1. r = x.left.size + 1
2. y = x
3. while y ≠ T.root
4.     if y == y.p.right
5.         r = r + y.p.left.size + 1
6.     y = y.p
7. return r
```

OS_RANK prvo stavlja r na size od lijevog podstabla od x (+ 1 za x), seta y na x i ide gore u stablo sve dok nismo dosli do korijena s while petljom. Ako je y desno dijete svog roditelja, uvecamo r sa sizeom od lijevog djeteta roditelja od x (+ 1 za tog roditelja). Na kraju algoritma r ce biti rang od x. Ako je x cvor s kljucem 59, r je na pocetku 1 i prvo cemo ici gore na kljuc 61. Sada uvecamo r na 4 (1 + 3). Idemo na kljuc 56. Sada uvecamo r na 10 (4 + 6) i na kraju dodemo do roota. Nas rang je 10.

2.

```
OS-SELECT(x, i)
1. while (x != NIL)
2.     if x.left != NIL
3.         left_size = x.left.size
4.     else
5.         left_size = 0
6.     if i == left_size + 1
7.         return x
8.     else if i < left_size + 1
9.         x = x.left
10.    else
11.        i = (i - left_size + 1)
12.        x = x.right
13. return NIL
```

Ovaj algoritam iterira kroz stablo s while petljom i u svakom koraku ima size lijevog podstabla od trenutnog čvora. Tu varijablu uspoređuje s i. Ako su jednaki funkcija vrati pointer na trenutni čvor. Ako je left.size + 1 veći od i, funkcija ide na lijevo podstablo, a inače ide na desno podstablo i oduzima veličinu od i po pravilu algoritma.

(3.)

```
OS-KEY RANK(T, k):
    return OS_KEY_RANK_HELP(T, T.root, k)
OS-KEY_RANK_HELP(T, x, k):
if  x == NIL
    return 0
elif x.key == k
    return 1 + size(x.left)
if x.key > k
    return OS_KEY_RANK_HELP(T, x.left, k)
return 1 + size(x.left) + OS_KEY_RANK_HELP
                              (T, x.right, k)
```

(4.)

1.
```
ithSuccessor(x, i)
1. r = OS-KEY-RANK(T, x.key) + i
2. succ = OS-SELECT(T.root, r)
3. return succ
```

2.
```
ithSuccessor(x, i)
curr = T.root
numV = 0
while curr != NIL
    if numV + 1 == i
        return curr
    else if numV + 1 < i
        numV ++
        curr = curr.right
    else curr = curr.left
```

U drugom slučaju ne možemo dobiti O(log n) rješenje jer nemamo informacije o veličini podstabla ili rangu čvora. Moramo tako sami odrediti korištenjem numV varijable, pa moramo dodatno šetati po cijelom podstablu. Novo rješenje bit će O(i + h). U najgorem slučaju i će biti n (rođ. successor čvora), a h u AVL-u O(log n).

Dakle, novo rješenje je O(i + h), gdje je i u najgorem slučaju n, a h O(log n).