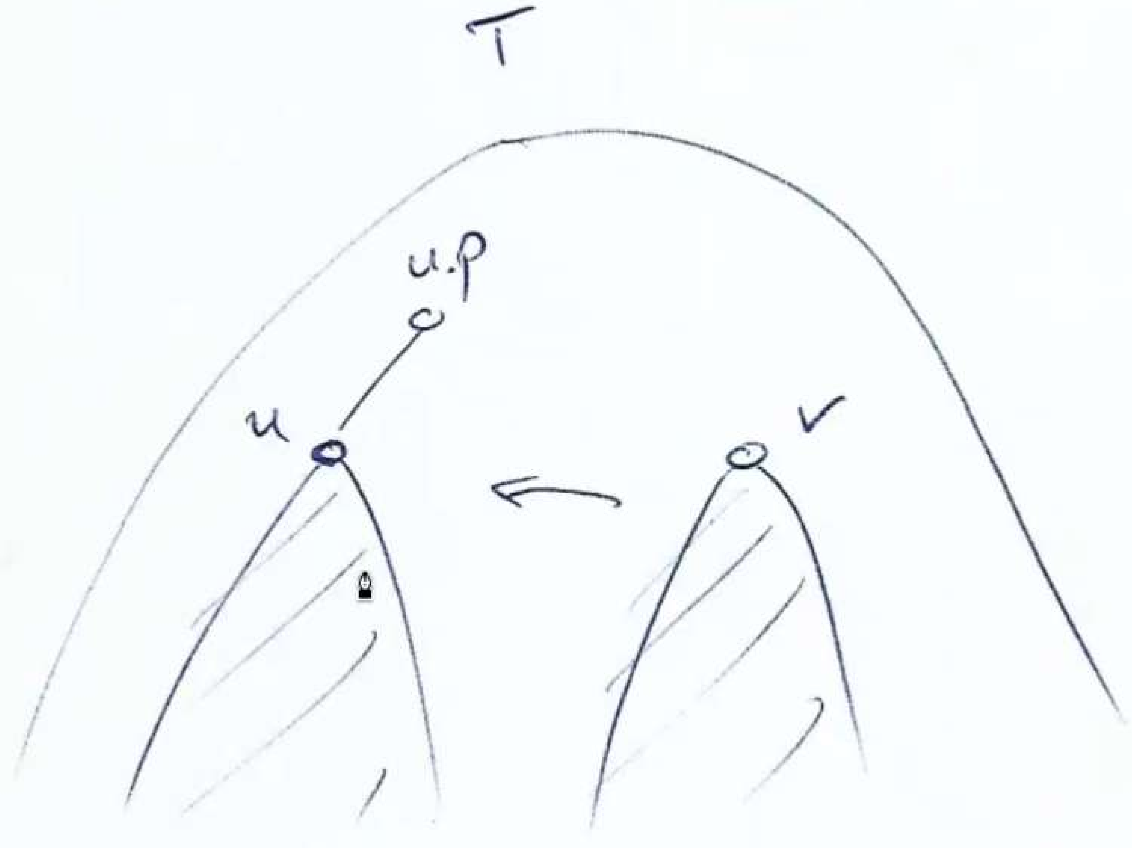
```
Transplant(T, u, v)
if u.p = NIL
    T.root = v
elseif u = u.p.left
        u.p.left = v
    else u.p.right = v
if v ≠ NIL
    v.p = u.p
```

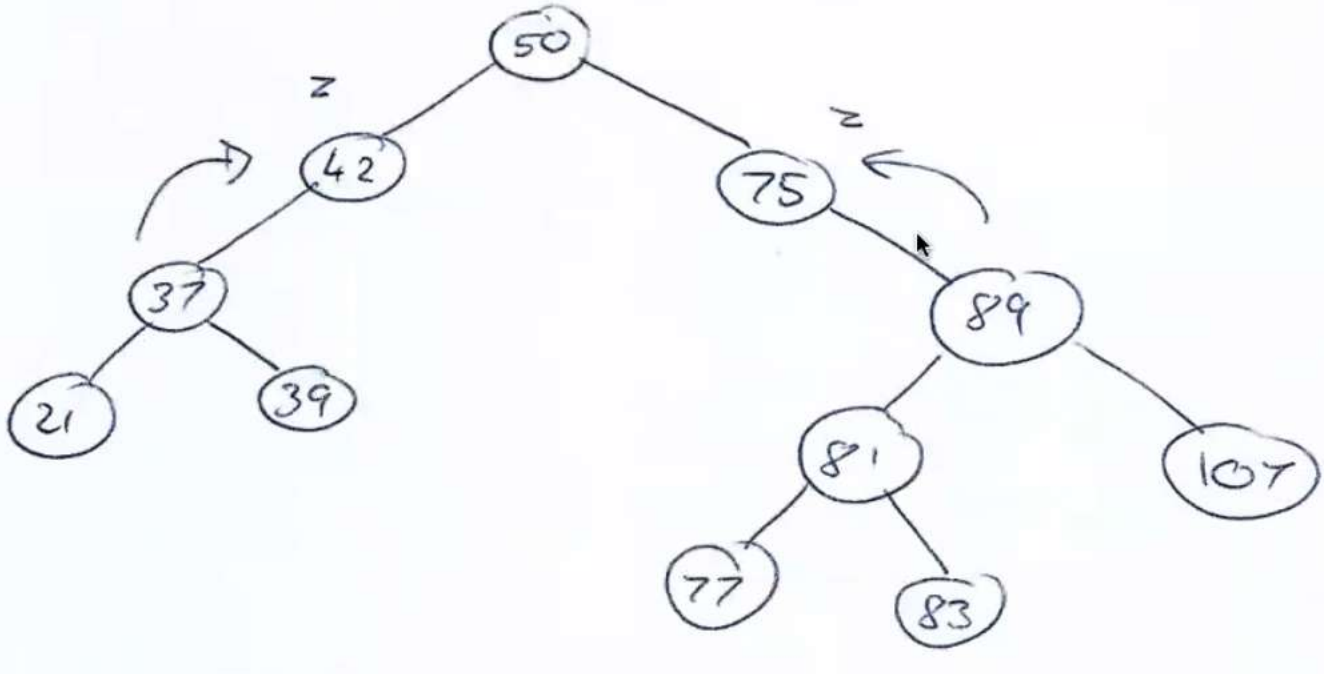

T
u.p
u
v

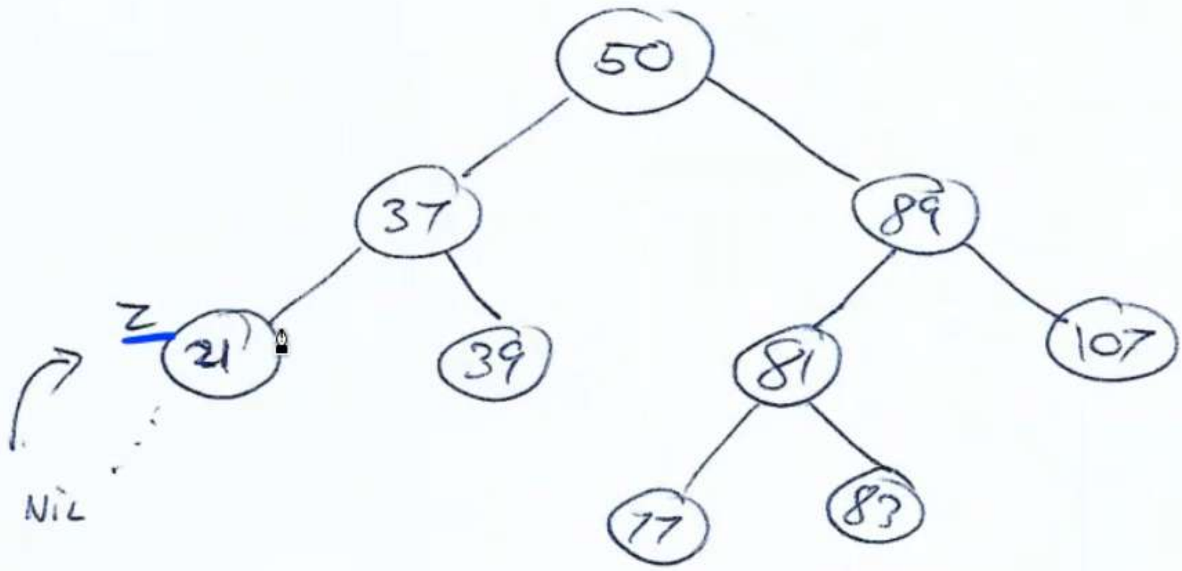
Tree-Delete (T, z)

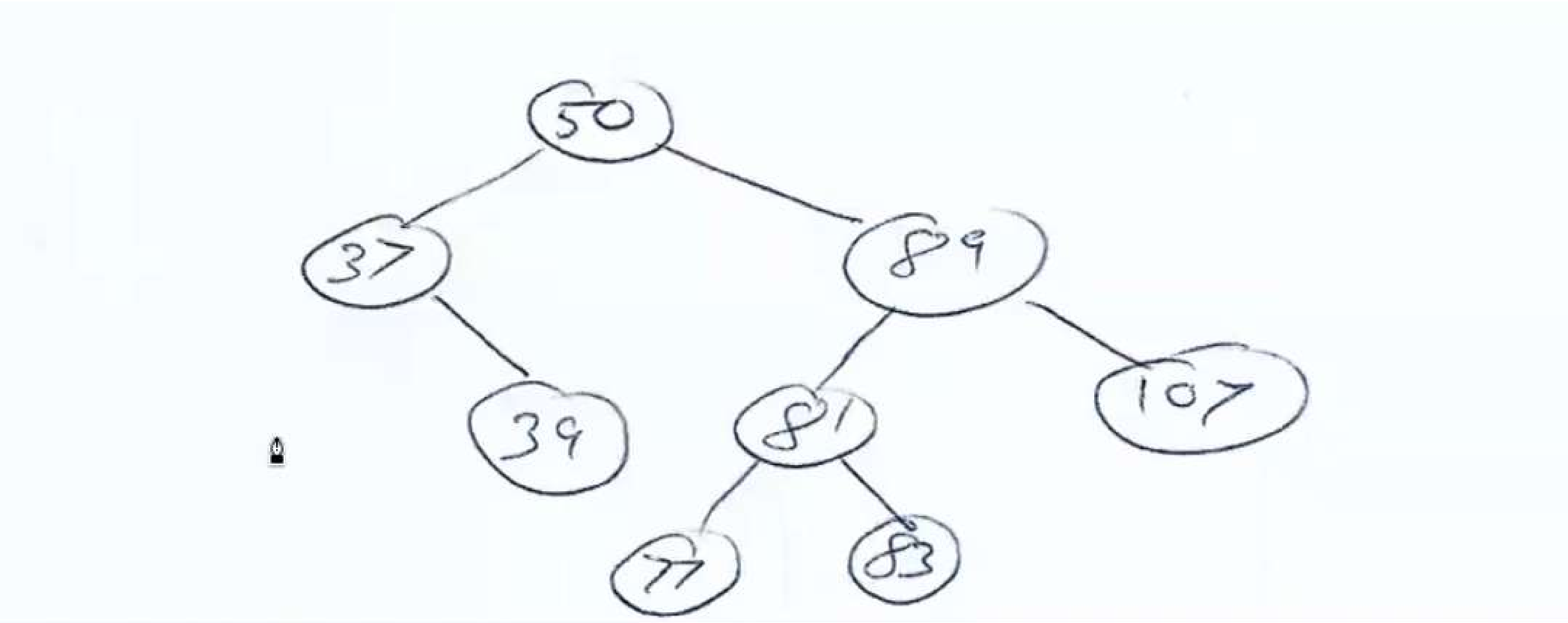
```
if z.left = NIL
    Transplant (T, z, z.right)
elseif z.right = NIL
    Transplant (T, z, z.left)
else y = Tree-Minimum (z.right)
    if y.p ≠ z
        Transplant (T, y, y.right)
        y.right = z.right
        y.right.p = y
```

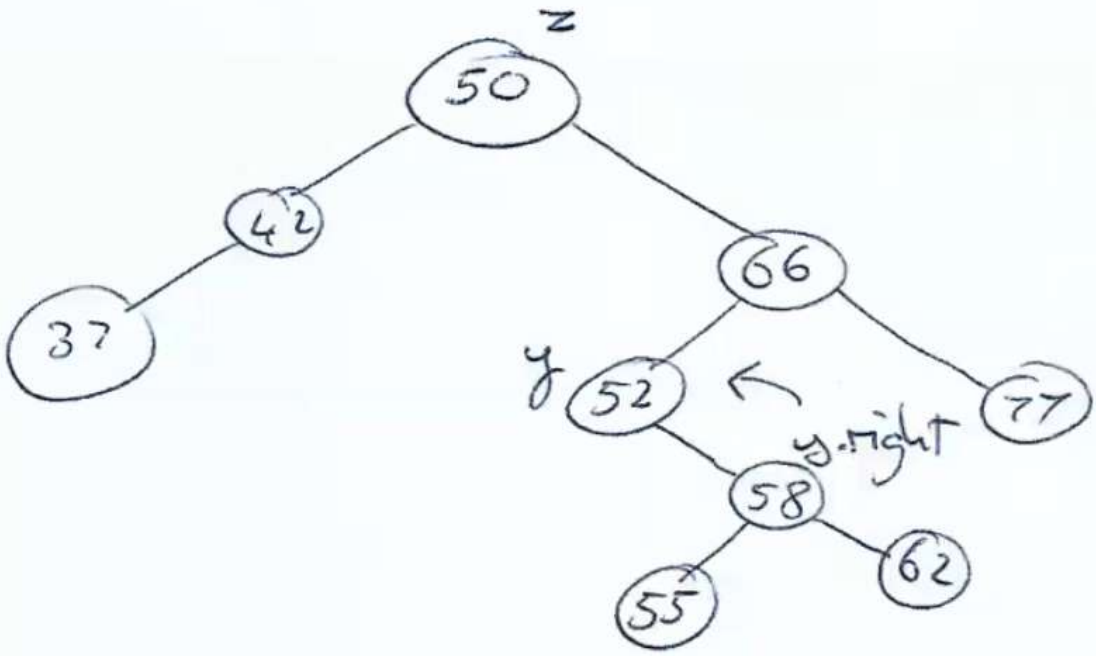
```
else  y = Tree-Minimum(z.right)
      if  y.p ≠ z
          Transplant(T, y, y.right)
          y.right = z.right
          y.right.p = y
      Transplant(T, z, y)
      y.left = z.left
      y.left.p = y
```

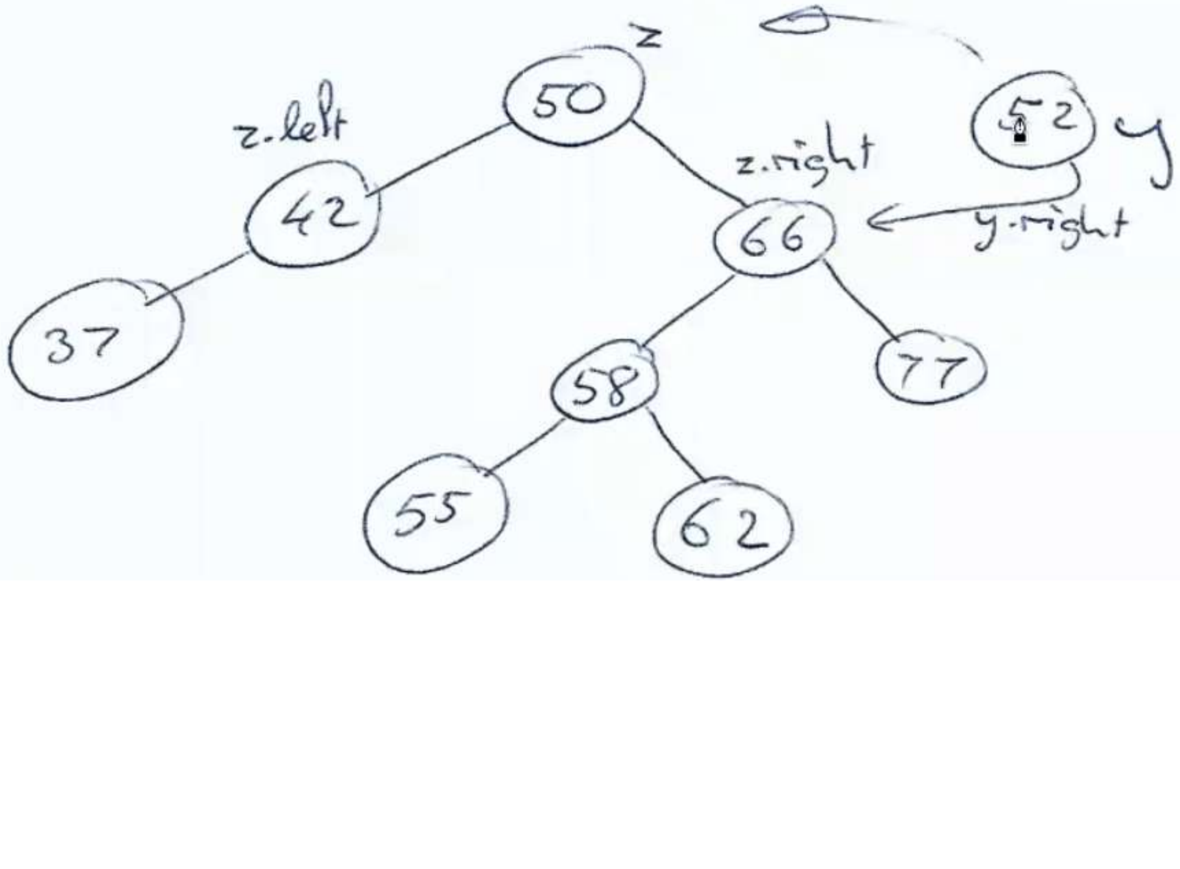
<u>Run</u>-time of Tree-Delete is $O(h)$.

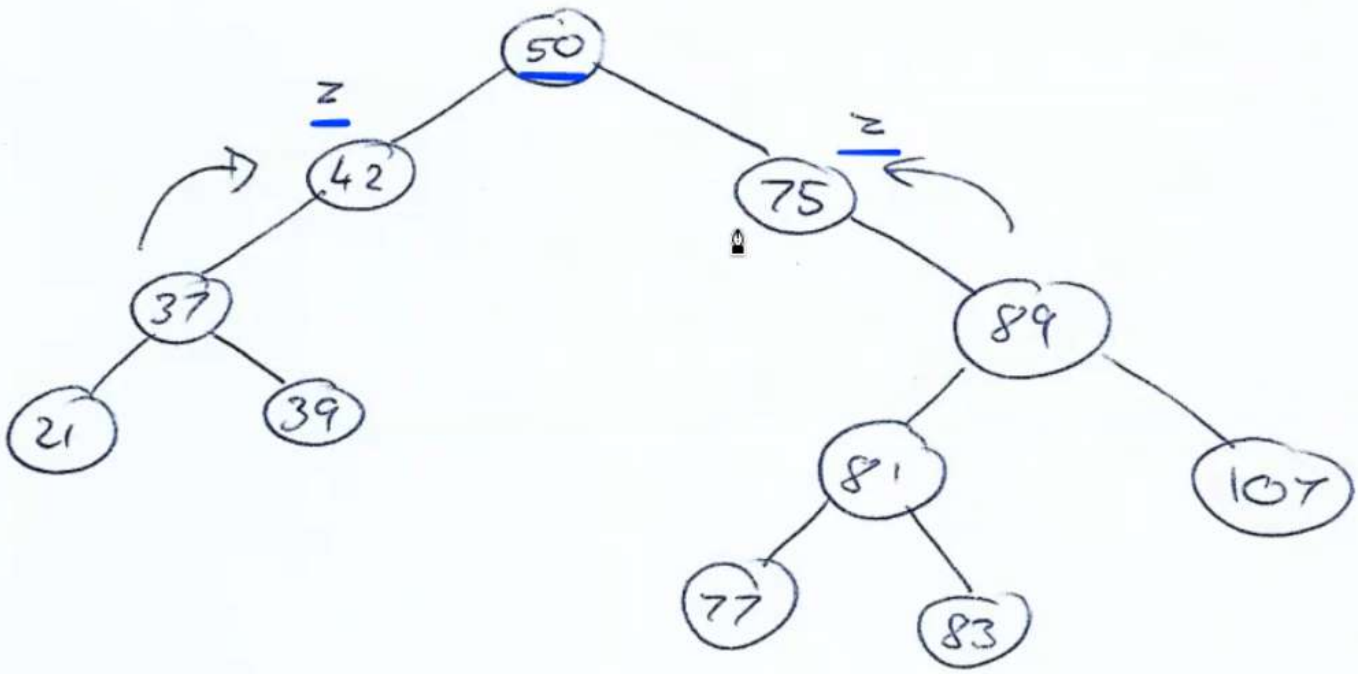

50
z
42
z
75
37
89
21
39
81
107
77
83


50
37
89
z
21
39
81
107
77
83
NIL


50
37
89
39
81
107
77
83


z
50
42
37
66
y
52
y.right
77
58
55
62

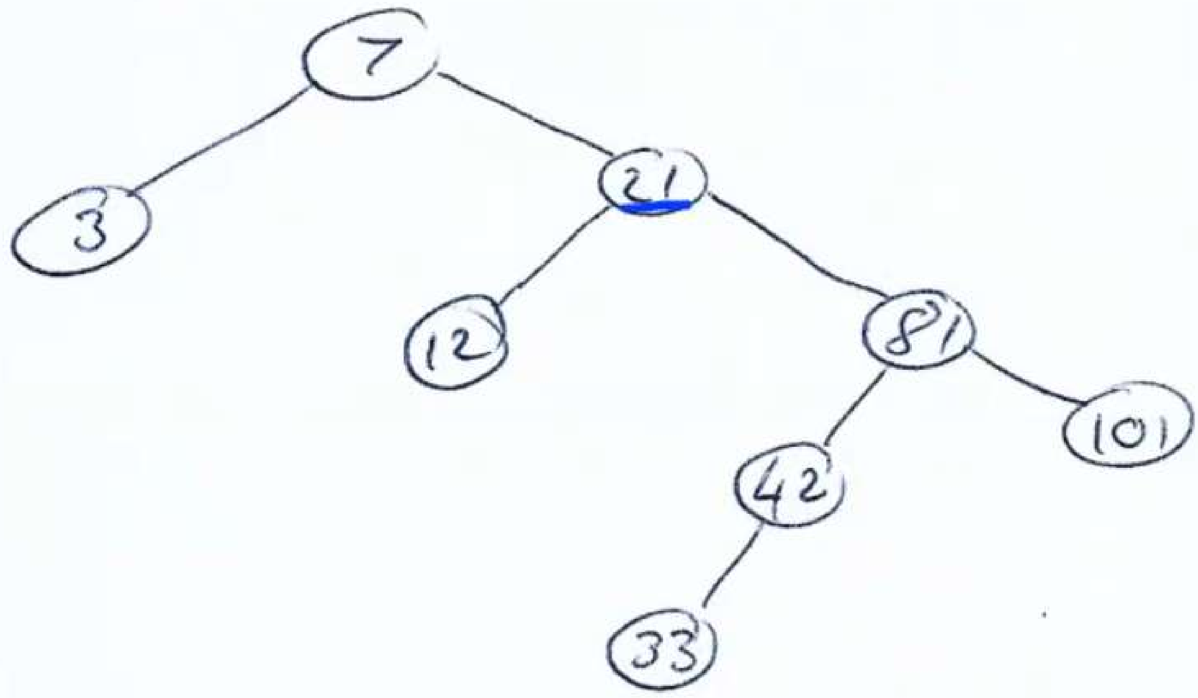

z
50
z.left
42
z.right
66
37
58
77
55
62
52
y
y.right


50
2
42
2
75
37
89
21
39
81
107
77
83

Suppose we have a list of objects that we want to store in a binary search tree. Each object has a key identifier - suppose the keys are :

7, 21, 12, 81, 3, 42, 33, 101

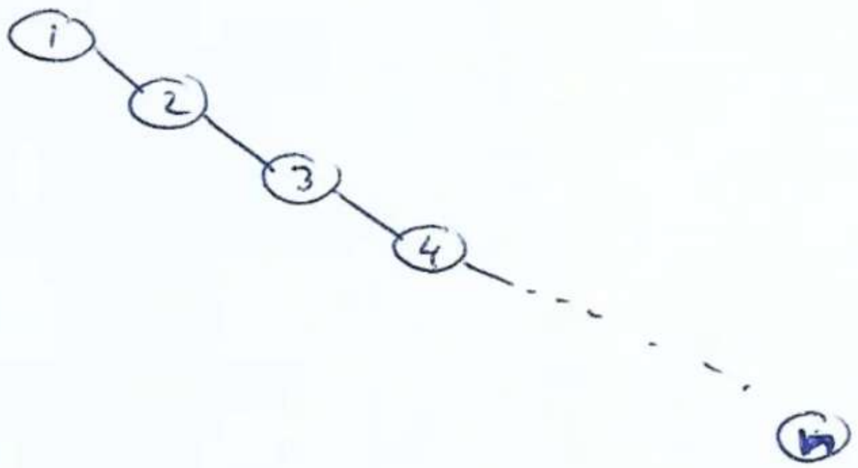
Starting from an empty Tree, call Tree-Insert repeatedly to get the tree :


7
3
21
12
81
42
101
33

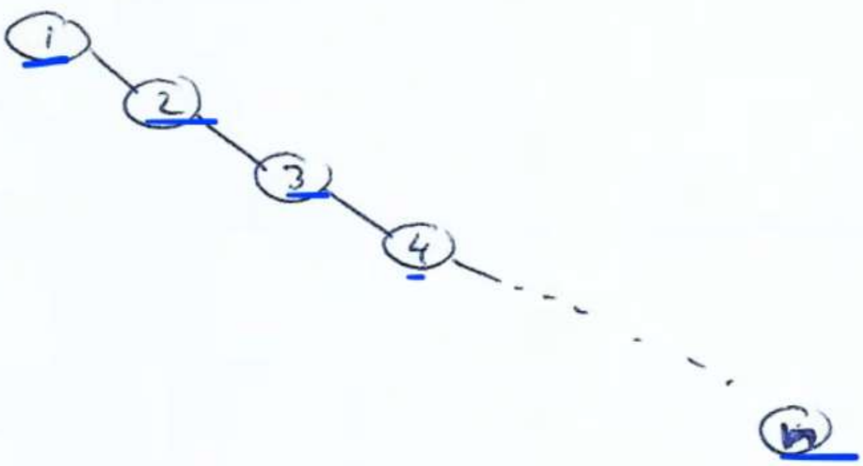
The tree is dynamic - can insert and delete and the operations Search, Insert, Delete, Max, Min all run in $O(h)$

suppose the list of objects came in this order : 1, 2, 3, 4, 5, ... n

Then the tree we build is :

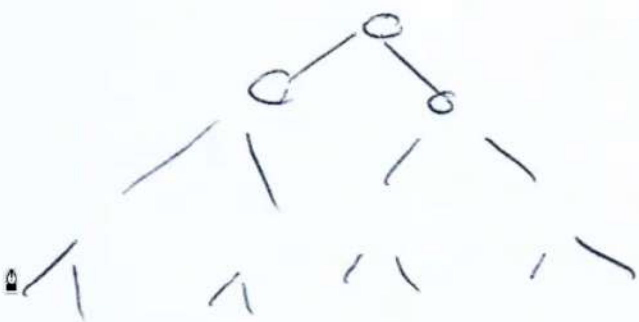


Then the Tree we build is :



Then $h = n$, so all operations run in $\Theta(n)$

It is better if the tree is balanced



Then $h \approx \log n$ so all operations run in $O(\log n)$.

One possibility to avoid $h = \Theta(n)$ is to randomly shuffle the list of keys..

Theorem 12.4 (proof omitted)

The expected height of a randomly built binary search tree on n distinct keys is $O(\log n)$.

Other methods to maintain balanced trees include:

AVL trees

treaps

Red-Black Trees