R-2.18

Here is an example of a max-heap having odd numbers from 1 to 59 as its elements.

The element 32 is inserted in the last place, as shown below and the steps of bubbling it up are shown.
The input AVL tree is given below.

After one left-rotation at the node 44 in the above tree, we obtain the balanced AVL tree as shown below.
Initially, splay tree is empty.

Insert 0 followed by 2.

Splay.

Insert 4.

Splay.

Repeat this insert-splay sequence until you get a chain of elements from 18 to 0, when you traverse the left child of every node starting from the root node, which would be 18.

Next, start searching for odd number from 1 to 19. Here, each search would be unsuccessful. So, at every step we must splay the parent node of the node at which search terminates which would be the nodes 2 to 18.

Search for 1 terminates unsuccessfully in 0 in the tree given below. So, splay 2. Apply zig-zig.
Continue the process.
Finally, while deleting the keys, we have to search for the key to be deleted. In general, for binary trees, we have 3 possibilities of subtrees with 0 (just remove) / 1 (swap with rightmost element of left subtree or the leftmost element of the right subtree and remove) / 2 (swap with rightmost element of left subtree or the leftmost element of the right subtree and remove) children for each node. In any case, splay the parent of the node that gets deleted.

C-3.5

Let \( h \) be an odd number representing the height of a node. Now, consider the AVL tree constructed as follows.

```
    h
   /\  /
  h-2/  \
 /   /   /
|   |   |
3   h-1
   /   \
   /    \
...    ...
```

This would require \( \Theta(h) = \Theta(log n) \) operations to maintain balance when the node labelled 1 (which also is a leaf node with height=1) is removed, as indicated.

C-3.30

We have \( k \) unique keys and are given with a total order. So, we could implement an AVL tree on the keys. When more than 1 element is categorized with the same key, we could use chaining at such nodes. Searching for a key takes \( O(log k) \) time. This will return a list of \( s \) elements, which can scanned linearly to locate the required item.