After 2nd swap

Swap 20 + 30

2 ptr changes
(from previous tree)
α's parent's (right ptr) → α's child  root → 30
α's child's (left ptr) → α  30 → 20
α's (right ptr) → α's (prev) successor  20 → 25

Another example

Type of insert? inner
(right subtree of α's left child)
double swap 'i) swap a's child & grandchild.

(1) swap 12 & 15

14
/ \ 15
/   \
8   13
/ \  \
7   13
  /  \\  
5   9

(2) swap 12 & 5 new child.

14
/ \ 15
/   \
8   13
/ \  \
7   12 16
/   /  \
5   9 13
Suppose space wasn’t an issue.

Lazy Deletion: As we did for arrays.

Problem?: Setting the value to -1 destroys order of tree.
Instead: Leave element in tree, but mark as deleted.

Binary Node:
- element
- left
- right
- duplicates
- boolean deleted

<table>
<thead>
<tr>
<th>operation</th>
<th>AVL</th>
<th>D-LL</th>
<th>ArrayList</th>
<th>array</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains</td>
<td>O(log n)</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>insert</td>
<td>O(log n)</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>lazy+ rm</td>
<td>O(log n)</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(1) (lazy)</td>
</tr>
<tr>
<td>get</td>
<td>O(log n)</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>
When to use AVL Trees?

1. to find min/max value

```
Object findMax(BinaryNode t)
```

```
if (t != null) // make sure tree is not empty
    while (t.right != null)
        t = t.right
    return t.element
```

2. to find a range of values

```
all values here > a
```

```
all values here: > b and < a
```

3. to get values in sorted order
   inorder traversal.
Suppose we want to store a list of Student records.

**Student object**

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>year</th>
<th>courses</th>
</tr>
</thead>
</table>

Want searches to be quick, which would make a good key?

Suppose ID is 6 digits (ex: 123456)
Want to store records in a way to quickly search by ID.
Ex: courses for student with ID=123456

- Array of size $10^6$, why $10^6$? highest ID = 999,999
  - Index = 123456
  - Problem?: mostly empty, lots of wasted space!

- Array of size $10^4$, why? Closer to $10^4 = 10,000$ students.
  - Just use last 4 digits of ID
    - Ex: ID = 123456
    - Index = 3456

Problem?: Two students may have the same last 4 digits
Few issues:

1. How to choose key?
2. How to map keys to index?
3. How to deal with 2 keys that map to same index? (Collision)

0. Keys:
   - Should be unique to each record (or close to unique).
   - Should be integers
   - If string, convert to int

3. Map key to index, table size = n
   - Always want index to be in [0, n-1]
   - How to ensure? mod!

\[
\text{index} = \text{key} \mod n = \lceil \text{hash(key)} \rceil
\]

hash function.

Many many variations of hash functions.
We will mostly look at key mod n * some variations

3. Collisions:
   - What to do if 2 keys hash (map) to same index?

1. Separate chaining - keep doubly linked list of all elements that hash to the same index
ex. \( n = 10 \)

Suppose keys: 1, 4, 9, 14, 24, 36, 49, 24 → 

\[
\begin{array}{c}
0 \\
1 \\
2 \\
3 \\
4 \\
5 \\
6 \\
7 \\
8 \\
9 \\
10 \\
\end{array}
\]

\[
\begin{array}{c}
\rightarrow 11 \\
\rightarrow [29] \\
\rightarrow [49] \\
\end{array}
\]

duplicate.

\[
\begin{array}{c}
\text{Time} \quad m = \# \text{ of elements}, \; n = \text{table size}.
\end{array}
\]

\[
\begin{array}{c}
\text{worst-case.}
\end{array}
\]

\[
\begin{array}{c}
\text{Find: } O(m) \quad (\text{all elements hash to same index}).
\end{array}
\]

\[
\begin{array}{c}
\text{Insert: if duplicates OK: } O(1).
\end{array}
\]

\[
\begin{array}{c}
\text{" want to store duplicates together: } O(m) \Rightarrow \text{find}(\).
\end{array}
\]

\[
\begin{array}{c}
\text{Delete: } O(m) \quad (\text{requires find}(\)).
\end{array}
\]

\[
\begin{array}{c}
\text{average-case.}
\end{array}
\]

\[
\begin{array}{c}
\text{Find/Insert/Delete: } O(1)
\end{array}
\]

\[
\begin{array}{c}
\text{Idea: elements typically spread out evenly over } n
\end{array}
\]

\[
\begin{array}{c}
\text{indices.}
\end{array}
\]

\[
\begin{array}{c}
m \text{ elements, } n \text{ indices } \Rightarrow O\left(1 + \frac{m}{n}\right)
\end{array}
\]

\[
\begin{array}{c}
\text{But } \frac{m}{n} \text{ will be bounded by a constant } c
\end{array}
\]

\[
\begin{array}{c}
\text{ex. } \Rightarrow 10000 \text{ elements, } n = 1000
\end{array}
\]

\[
\begin{array}{c}
c \approx 10.
\end{array}
\]

\[
\begin{array}{c}
\frac{m}{n} = \text{load} = \lambda.
\end{array}
\]

\[
\begin{array}{c}
\text{worst-case is very rare}!!
\end{array}
\]