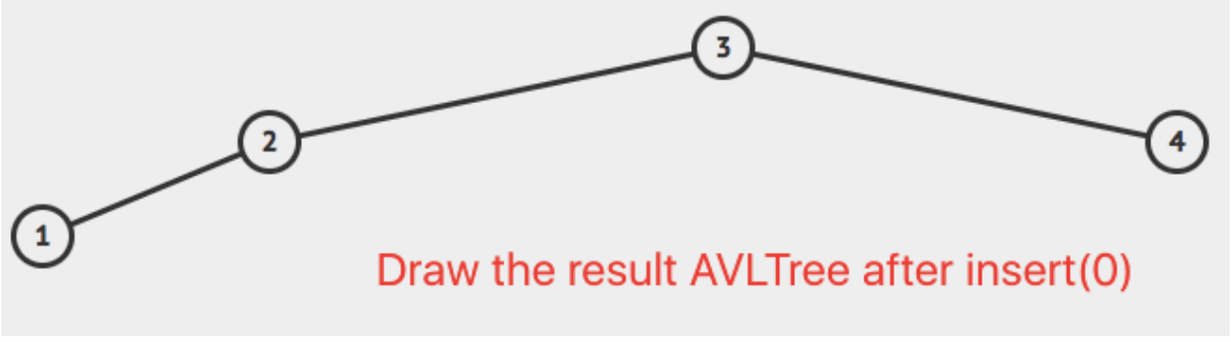
# CSCI-SHU 210 Data Structures

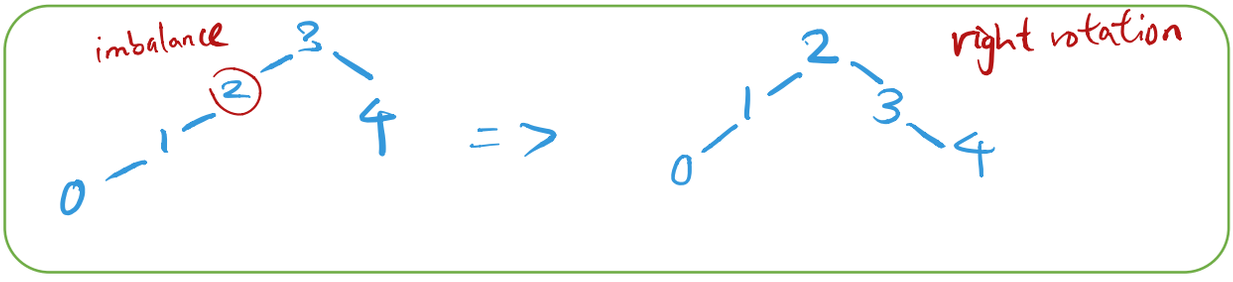
## Recitation11 Worksheet AVL Trees and Sorting Algorithms

### Part 1: AVL Tree

- **Single Rotations**

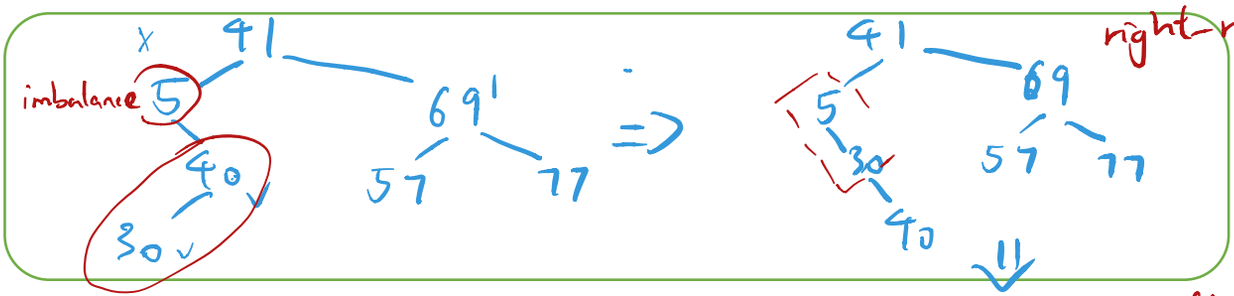


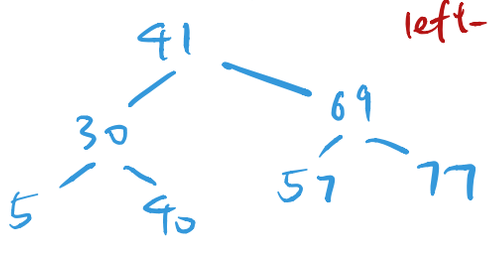

a). Suppose we have the AVLTree above. Draw the AVLTree after insert(0).




- **Double Rotations**

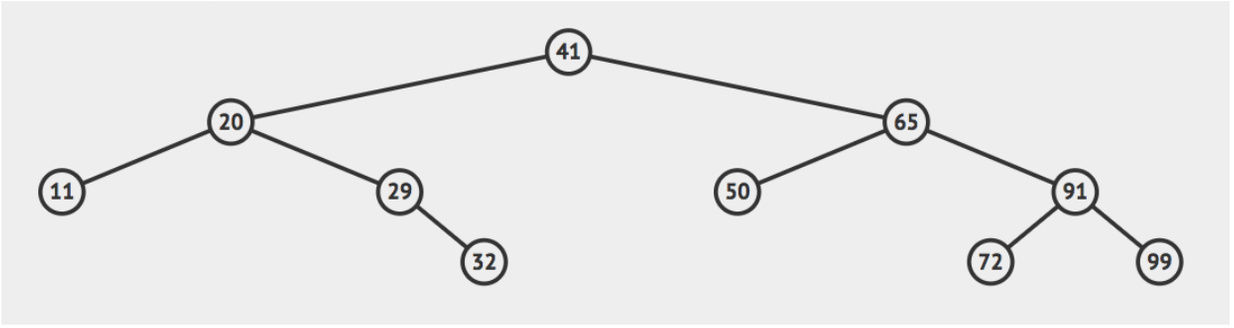

a). Suppose we have the AVLTree above. Draw the AVLTree after insert(30).



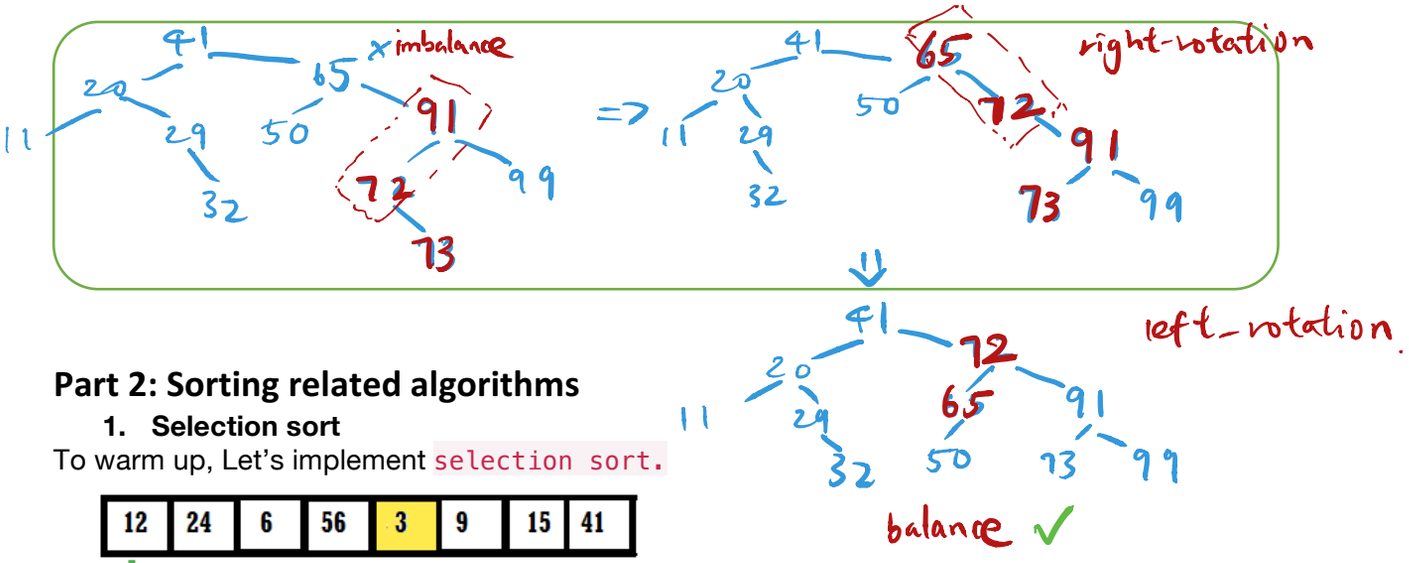




balance ✓

➢ **Challenge**



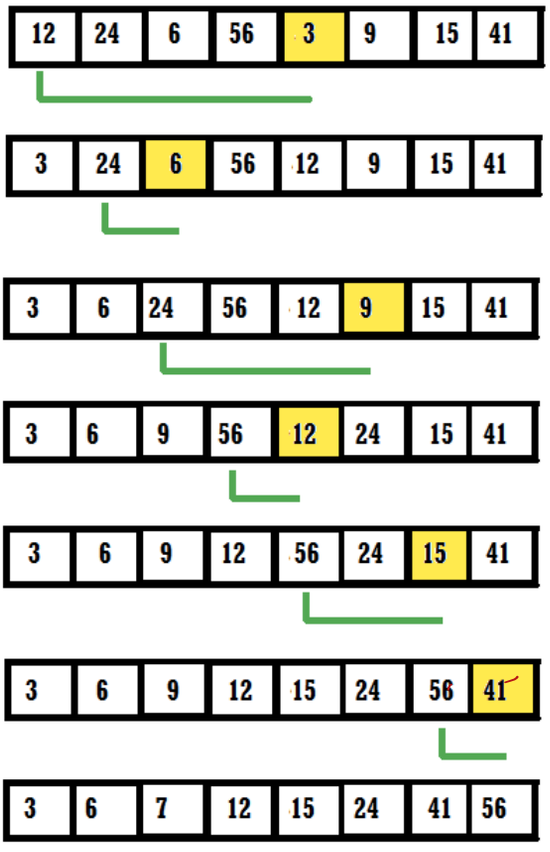

a). Suppose we have the AVLTree above. Draw the AVLTree after insert(73).




## Part 2: Sorting related algorithms

### 1. Selection sort

To warm up, Let's implement `selection sort.`




Your task 1: Implement # To do functions within selection_sort.py.

## 2. Textbook Quick Sort, and its variations

Our textbook have provided the following quick sort code:

```
1.  def inplace_quick_sort(S, a, b):
2.
3.  """Sort the list from S[a] to S[b] inclusive using the quick-sort algorithm."""
4.      if a >= b: return                          # range is trivially sorted
5.      pivot = S[b]                               # last element of range is pivot
6.      left = a                                   # will scan rightward
7.      right = b-1                                # will scan leftward
8.      while left <= right:
9.          # scan until reaching value equal or larger than pivot (or right marker)
10.         while left <= right and S[left] < pivot:
11.             left += 1
12.         # scan until reaching value equal or smaller than pivot (or left marker)
13.         while left <= right and pivot < S[right]:
14.             right -= 1
15.         if left <= right:                      # scans did not strictly cross
16.             S[left], S[right] = S[right], S[left]              # swap values
17.             left, right = left + 1, right - 1                  # shrink range
18.
19.     # put pivot into its final place (currently marked by left index)
20.     S[left], S[b] = S[b], S[left]
21.     # make recursive calls
22.     inplace_quick_sort(S, a, left - 1)
23.     inplace_quick_sort(S, left + 1, b)
```

It works, and it is inplace. (modifies original list)

Which element does this implementation use as the partition pivot?

it uses the right-most (last) element as pivot.

Your task 2: Modify the given code, so the pivot is chosen using median of three instead.

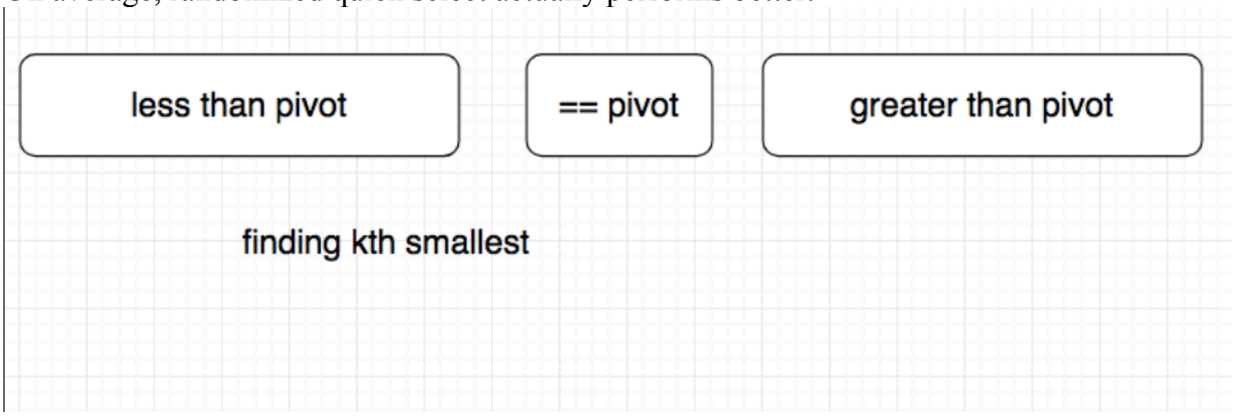
## 3. Base 10 Radix Sort

Your task 3: Implement # To do functions within radix_sort.py.

4. **Randomized quick select**

- This randomized quick select has O(n^2) worst case runtime. (The one we are going to implement)
- There's another quick select algorithm that picks median of median as pivot, with O(n) worst case runtime. (Not important in this course, but good to know)

On average, randomized quick select actually performs better.




```
Your task 4: Implement # To do functions within quick_select.py

The quick select algorithm finds kth smallest element of unsorted array.

Expected runtime is between O(n) ~ O(nlogn),
which is better than sorting, then pick index [k - 1].
```