CSC 148: Introduction to Computer Science

Week 11

In-place Quicksort

Mutating the input list in a space-efficient way

University of Toronto Mississauga,

Department of Mathematical and Computational Sciences

So far ...

```
def quicksort(lst: list) -> list:
   if len(lst) < 2:
      return lst[:]
   else:
      pivot = lst[0]

      smaller, bigger = _partition(lst[1:], pivot)

      smaller_sorted = quicksort(smaller)
      bigger_sorted = quicksort(bigger)

      return smaller_sorted + [pivot] + bigger_sorted
```

- Returns the new sorted list, does not mutate the original one
- How about sorting the original list in place?

In-place Quicksort

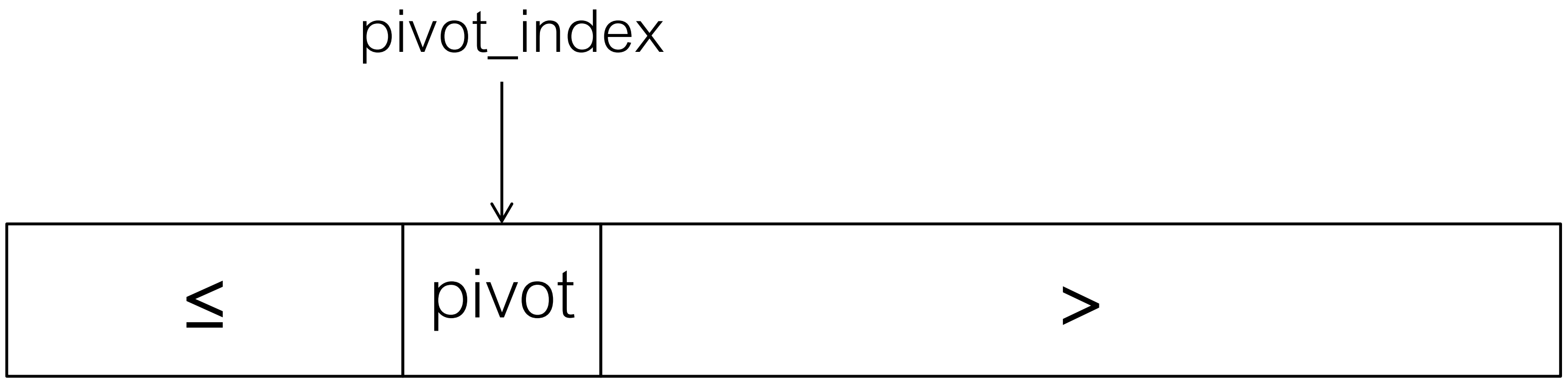
- The key helper: in place partition!

- Worksheet: Write a version of _partition that mutates lst directly
    - Follow the steps in Q1 and Q2 from the worksheet
    - Stop at Q3 for now...

Simulating Slicing with Indexes

- We often want to operate on just part of a list:
  - `f(lst[start:end])`

- Rather than create a new list object, we pass in the indexes:
  - `f(lst, start, end)`



Simulating Slicing with Indexes

```
_in_place_partition(lst)
   =>
_in_place_partition(lst, start, end)
```

```
quicksort(lst)
   =>
_in_place_quicksort(lst, start, end)
```