Week 04 - Lecture 1 Slides

Dr. Yeganeh Bahoo

Created: 2023-09-27 Wed 05:47

Table of Contents

Lecture 1: Sorting (cont.)

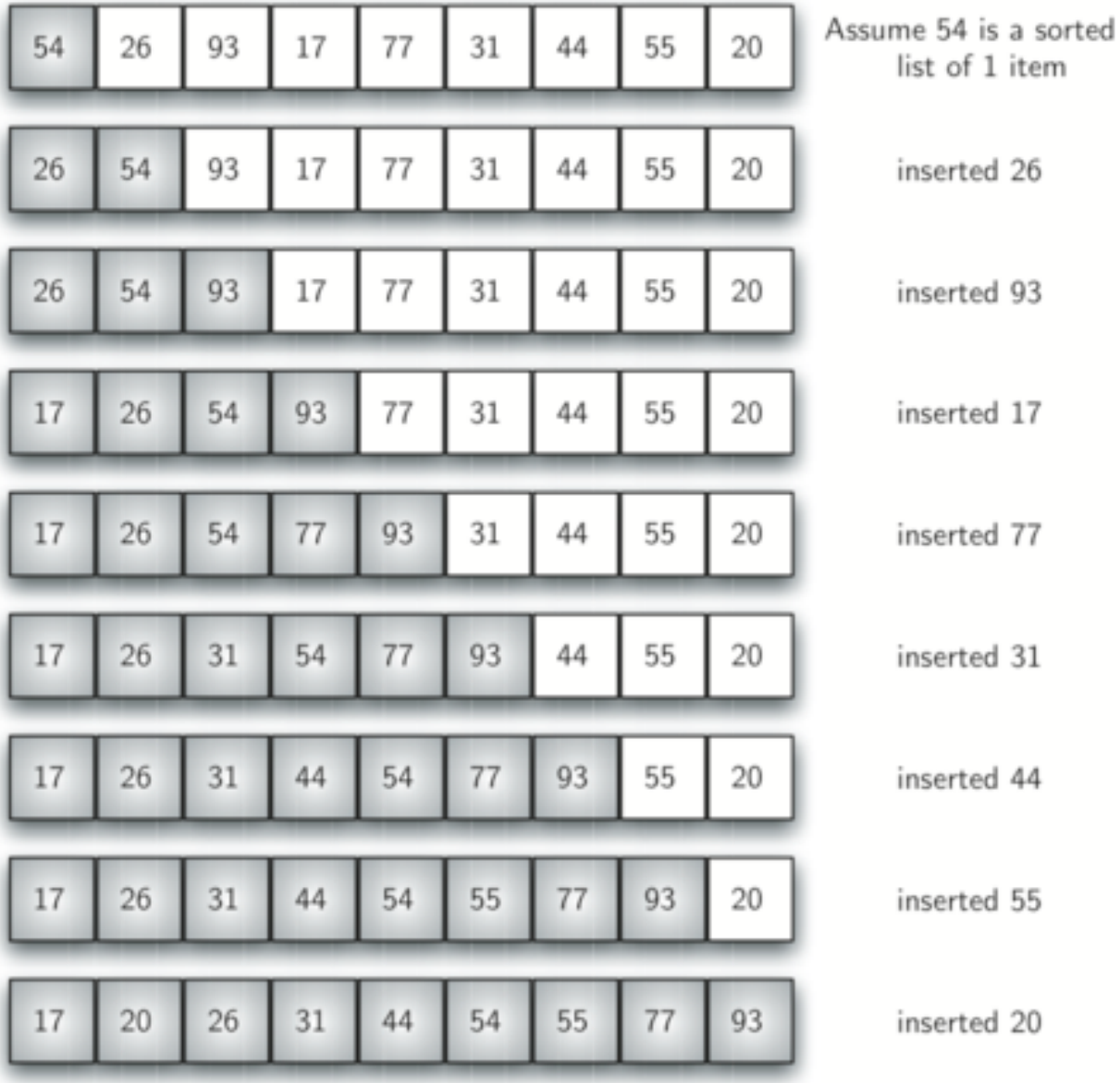
Learning objectives: By the end of this lecture you should be able to explain and implement the following sorting strategies:

- insertion sort
- quicksort

Insertion sort

- it is **online**: the left part is already sorted; unlike selection sort, it doesn't have to find the maximum element of the whole sequence in a pass
- each new item is then "inserted" back into the previous sublist such that the sorted sublist is one item larger.

https://youtu.be/ROalU379l3U



Insertion sort algorithm

```
(defun insertion-sort (vec comp)
  (dotimes (i (1- (length vec)))
    (do ((j i (1- j))) ; starts at i, decrements by 1
        ((minusp j))   ; checks if j is negative
      (if (funcall comp (aref vec (1+ j)) (aref vec j))
          (rotatef (aref vec (1+ j)) (aref vec j))
          (return))))
  vec)
```

Caption:

- i is the index of the leftmost unsorted element. i starts from 0, is incremented by ones.
- j is initially i, and $j + 1$ is index of the current element being inserted into the sorted part.
- If the comparison between the current element and its left neighbour (of index j) is true, the values are swapped, j is decremented by one, and current element and another atempt to insert current element in the sorted part takes place.
- if no insertion (swap) takes place, i is incremented

Example:

```
RTL-USER> (insertion-sort #(6 9 5 1 3 5 4 0 7 4) #'<)
#(0 1 3 4 4 5 5 6 7 9)
```

Question

```
(defun insertion-sort (vec comp)
  (dotimes (i (1- (length vec)))
    (do ((j i (1- j)))
        ((minusp j))  ; checks if j is negative
      (if (funcall comp (aref vec (1+ j)) (aref vec j))
          (rotatef (aref vec (1+ j)) (aref vec j))
          (return)))
    (format t "~a pass: ~a~%" vec (1+ i)))
  vec)
```

Given array `#(20 14 85 3 9)`, what value will be in the 0^{th} position of the array after the first pass over the outer loop.

Solution

Quiz

https://bit.ly/3PTJhP6

Analysis of Insertion Sort

```
(defun insertion-sort (vec comp)
  (dotimes (i (1- (length vec)))
    (do ((j i (1- j)))
        ((minusp j))  ; checks if j is negative
      (if (funcall comp (aref vec (1+ j)) (aref vec j))
          (rotatef (aref vec (1+ j)) (aref vec j))
          (return)))
    (format t "~a pass: ~a~%" vec (1+ i)))
  vec)
```

- # of comparisons
  - Total number of comparisons $\sum_{i=1}^{n} i = \frac{n^2+n}{2}$, the sum of the arithmetic progression from 1 to n, because, at each step, it may need to fully examine the sorted prefix to find the maximum and the prefix's size varies from 1 to n.
  - Therefore $O(n^2)$ comparisons.

Quicksort

- It is a famous sorting algorithm of the class that work in $O(n \log n)$ time.
- It relies on the divide-and-conquer approach: divides the sequence and recursively sorts its segments.
- The idea:
  - We choose a **pivot** value (there are different ways of choosing this value) which will help us split the vector in two parts: one part containing elements smaller than the pivot, and another containing those that are greater
  - The position where the pivot actually belongs in the sorted sequence is called the **ppvt**
  - The goal is to move items that are on the wrong side with respect to the pivot value while also converging on **ppvt**.

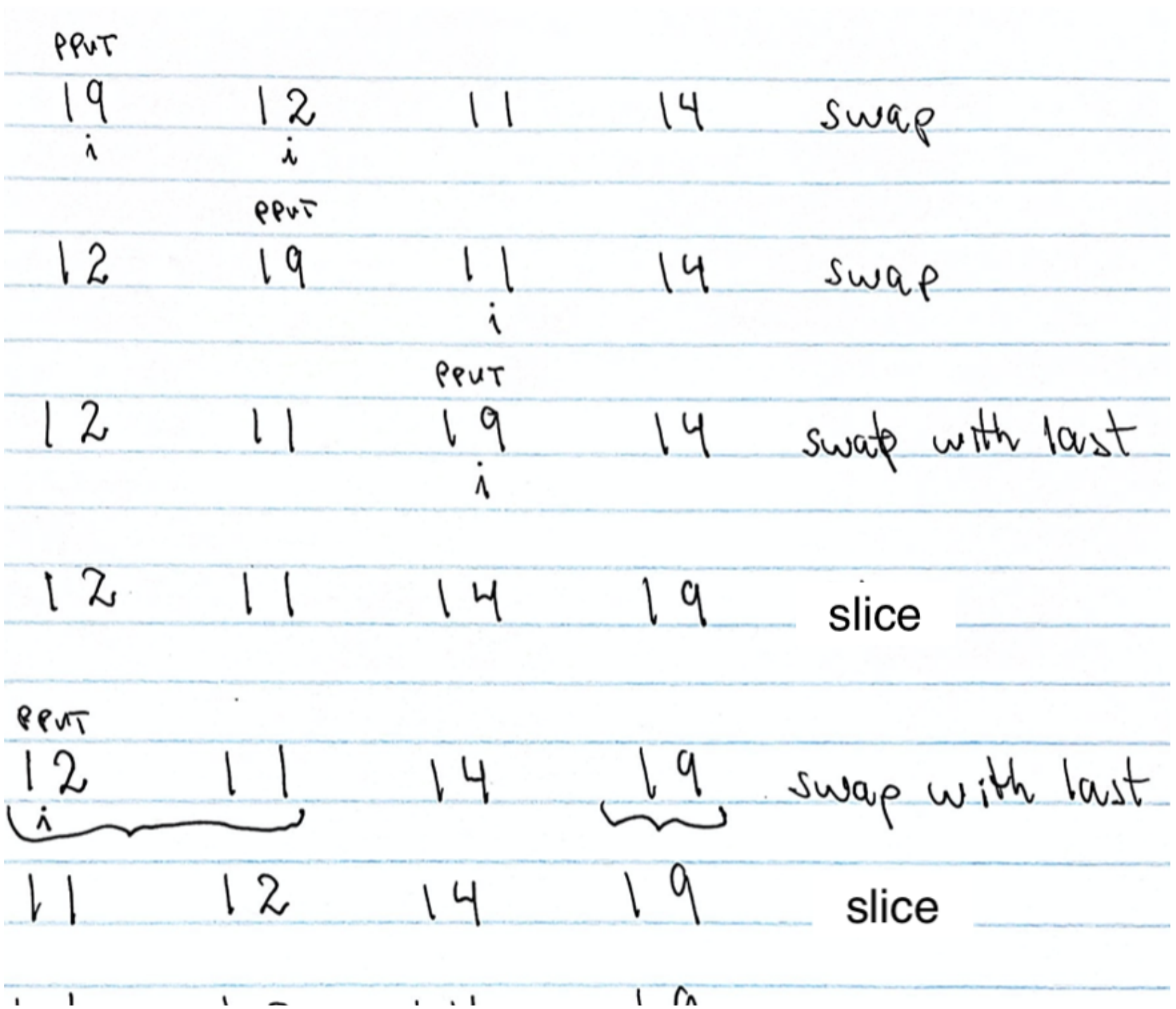
Quicksort algorithm

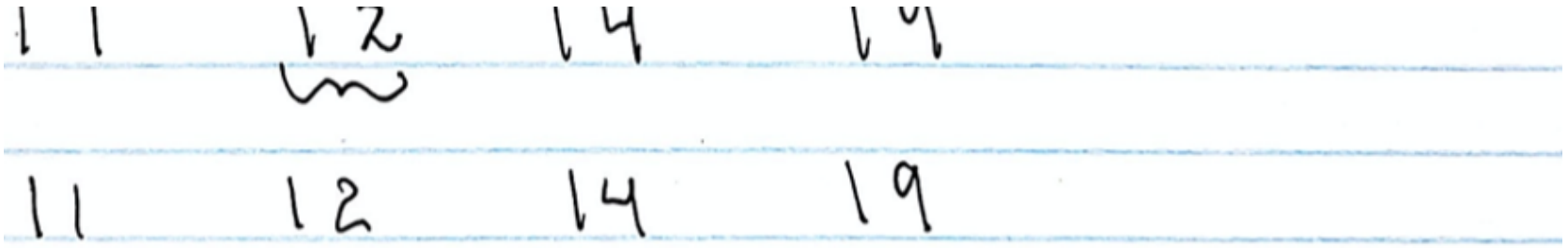
```
(defun quicksort (vec comp)
```

```
(when (> (length vec) 1)
  (let ((ppvt 0)
        (pivot (aref vec (1- (length vec))))) ; last element
    (dotimes (i (1- (length vec))) ; finds position of the pivot
      (when (funcall comp (aref vec i) pivot)
        (rotatef (aref vec i) (aref vec ppvt))
        (incf ppvt)))
    ;; swap the pivot (last element) in its proper place
    (rotatef (aref vec (1- (length vec))) (aref vec ppvt))
    (quicksort (rtl:slice vec 0 ppvt) comp)
    (quicksort (rtl:slice vec (1+ ppvt)) comp)))
vec)
```

- In-place sorting
  - RTL:SLICE does not create new copies of the subarrays
- It is not straight-forward to convert this particular recursion into looping

Tracing quicksort


PPVT
19 12 11 14 swap
i i
PPVT
12 19 11 14 swap
i
PPVT
12 11 19 14 swap with last
i
12 11 14 19 slice
PPVT
12 11 14 19 swap with last
i
11 12 14 19 slice



Analysis of quicksort

- **Best case**:
  - The pivot is in the middle of the vector
  - On every interation, (n comparisons + $n/2$ swaps + $n/2$ increments) = $2n$ operations. We'll need to do that $\log n$ times
  - Hence $T(n) = 2n \log n$, i.e., $O(n \log n)$
- **Worst case**:
  - Split point skewed to the left or right.
  - Sublists to be sorted will have 0 items and $n - 1$ items, i.e., $n + (n - 1) + (n - 2) + \cdots$ comparisons.
  - Hence $O(n^2)$.

Exercise

QUICKSORT's implementation used a functional programming technique that enables the programmer to use the function to sort arrays of **any** data type.

Suppose the structure below represents information about a course.

```
(defstruct course
  code name prereqs)
```

Let's create an array of courses

```
CL-USER>  (setf a (make-array 3 :initial-contents (list (make-course :code "cps506")
                                                        (make-course :code "cps305")
                                                        (make-course :code "cps393"))))
#(#S(COURSE :CODE "cps506" :NAME NIL :PREREQS NILx)
  #S(COURSE :CODE "cps305" :NAME NIL :PREREQS NIL)
  #S(COURSE :CODE "cps393" :NAME NIL :PREREQS NIL))
```

How would you use QUICKSORT to sort the courses in ascending order based on their course code? Fill in the blank.

```
CL-USER> (quicksort a ...)
#(#S(COURSE :CODE "CPS305" :NAME NIL :PREREQS NIL)
  #S(COURSE :CODE "CPS393" :NAME NIL :PREREQS NIL)
  #S(COURSE :CODE "CPS506" :NAME NIL :PREREQS NIL))
```

Solution

```
(quicksort a #'(lambda (x y)
                 (string< (course-code x) (course-code y))))
```

Quiz

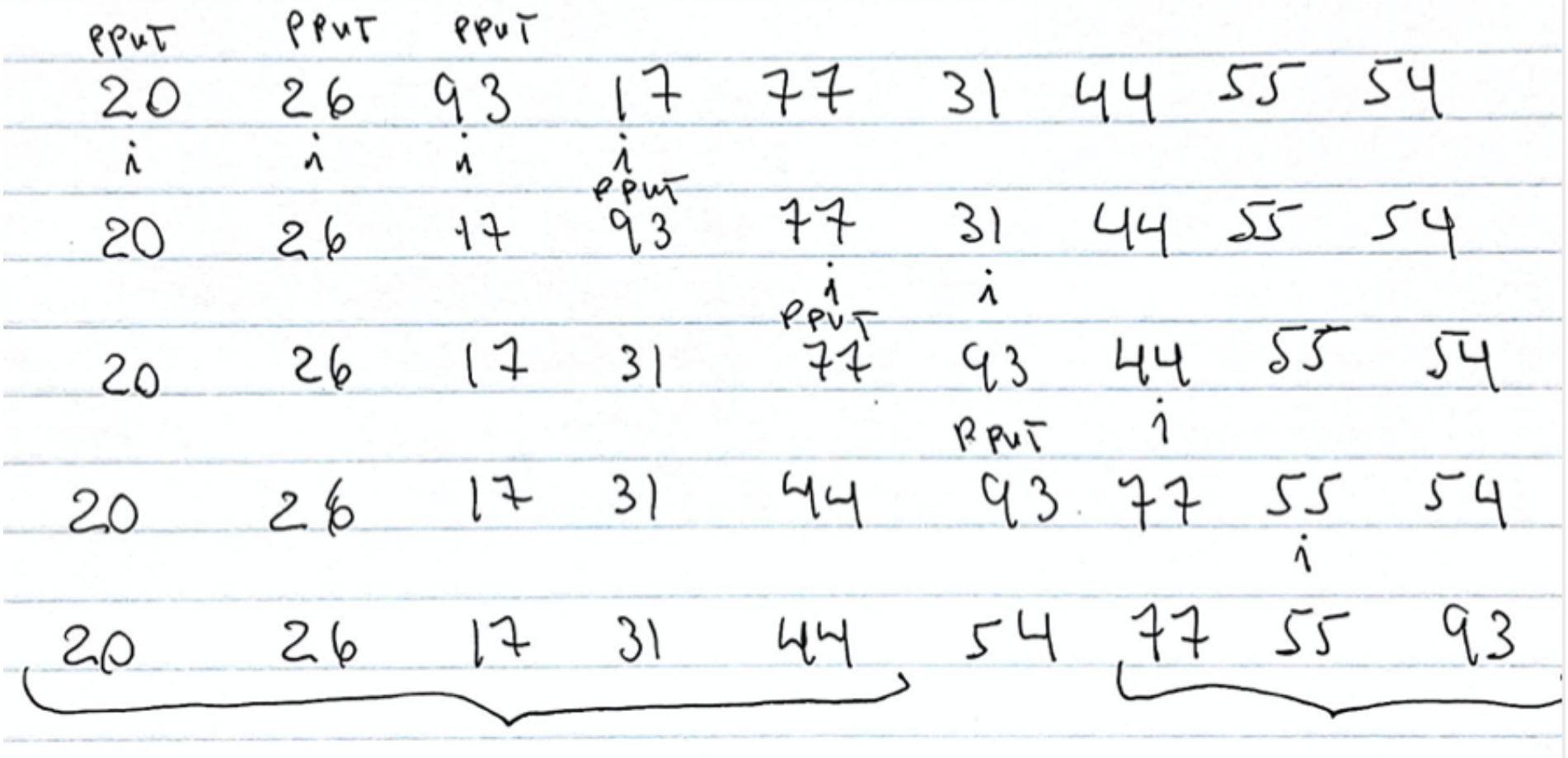
This question is about quicksort.

Given the following vector

#(	20	26	93	17	77	31	44	55	54)

what would be the elements of the two subarrays immediately after the first array split?

Solution


PPUT PPUT PPUT
20 26 93 17 77 31 44 55 54
i i i i
PPUT
20 26 17 93 77 31 44 55 54
i i
PPUT
20 26 17 31 77 93 44 55 54
PPUT i
20 26 17 31 44 93 77 55 54
i
20 26 17 31 44 54 77 55 93