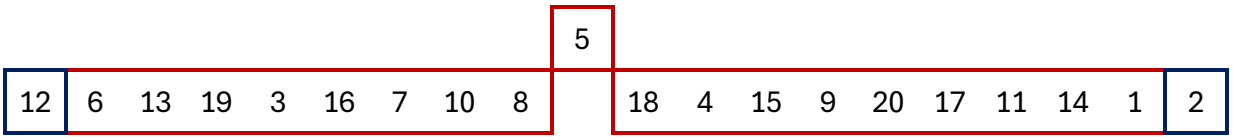
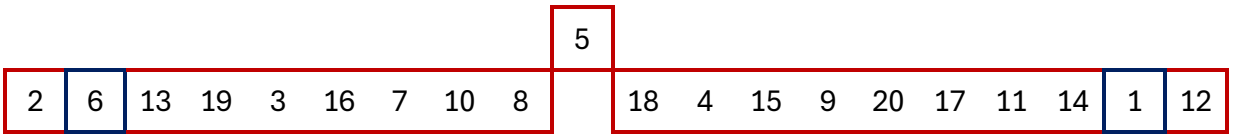
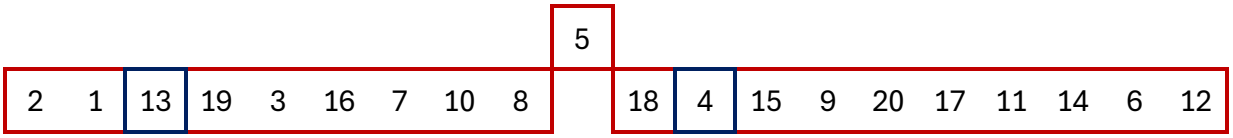
Step 1: By Hand



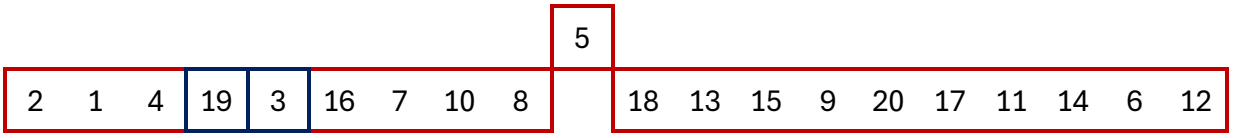

POINTER SCANNING LEFT DETECTS LARGER ITEM THAN PIVOT AND POINTER ON RIGHT DETECTS SMALLER NUMBER THAN PIVOT SO SWAP POINTER VALUES



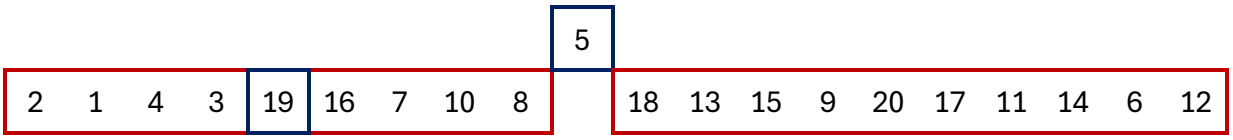

POINTER SCANNING LEFT DETECTS LARGER ITEM THAN PIVOT AND POINTER ON RIGHT DETECTS SMALLER NUMBER THAN PIVOT SO SWAP POINTER VALUES



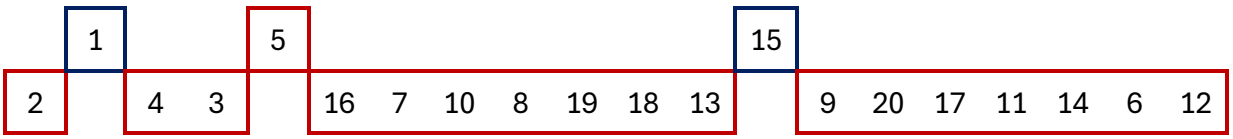

POINTER SCANNING LEFT DETECTS LARGER ITEM THAN PIVOT AND POINTER ON RIGHT DETECTS SMALLER NUMBER THAN PIVOT SO SWAP POINTER VALUES



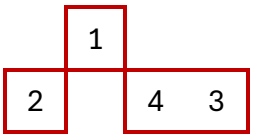

POINTER SCANNING LEFT DETECTS LARGER ITEM THAN PIVOT AND POINTER ON RIGHT DETECTS SMALLER NUMBER THAN PIVOT SO SWAP POINTER VALUES



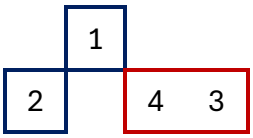

POINTERS MEET AND POINTER VALUE IS OUT OF ORDER WITH PIVOT SO SWAP PLACES



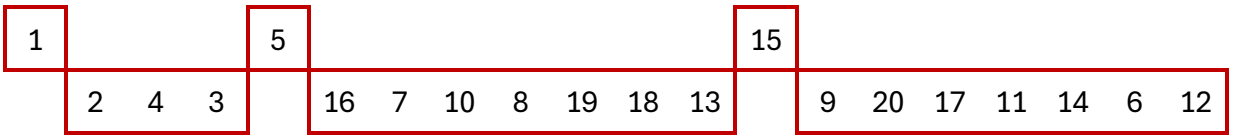

PIVOT POSITION HAS BEEN SORTED SO NEW PIVIOT POINT FOR SUB-LISTS



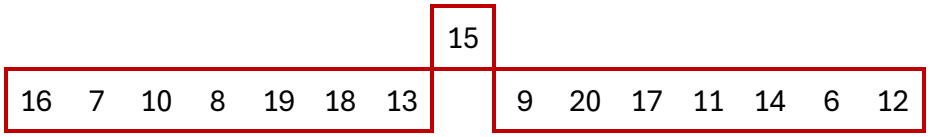

FOCUS ON THE FIRST SUB-LIST



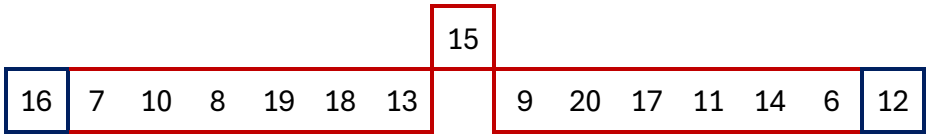

POINTERS MEET AND POINTER VALUE IS OUT OF ORDER WITH PIVOT SO SWAP PLACES



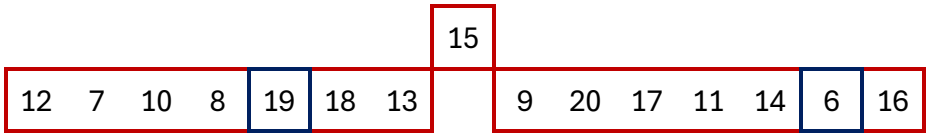

PIVOT POSITION HAS BEEN SORTED SO FIRST SUB-LIST IS DONE



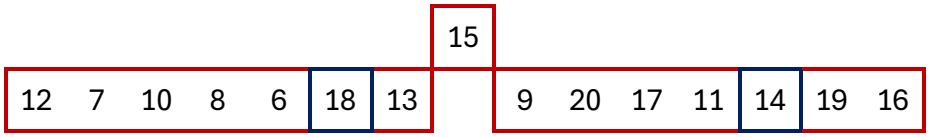

FOCUS ON THE SECOND SUB-LIST



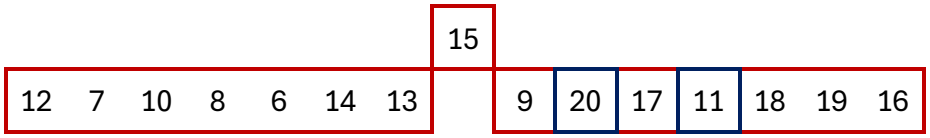

POINTER SCANNING LEFT DETECTS LARGER ITEM THAN PIVOT AND POINTER ON RIGHT DETECTS SMALLER NUMBER THAN PIVOT SO SWAP POINTER VALUES



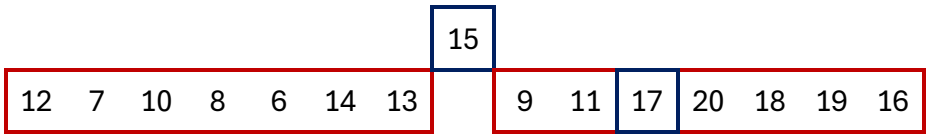

POINTER SCANNING LEFT DETECTS LARGER ITEM THAN PIVOT AND POINTER ON RIGHT DETECTS SMALLER NUMBER THAN PIVOT SO SWAP POINTER VALUES



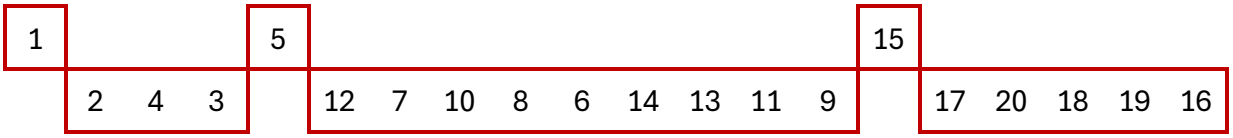

POINTER SCANNING LEFT DETECTS LARGER ITEM THAN PIVOT AND POINTER ON RIGHT DETECTS SMALLER NUMBER THAN PIVOT SO SWAP POINTER VALUES



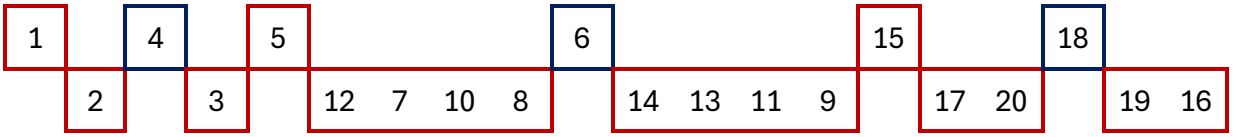

POINTER SCANNING LEFT DETECTS LARGER ITEM THAN PIVOT AND POINTER ON RIGHT DETECTS SMALLER NUMBER THAN PIVOT SO SWAP POINTER VALUES



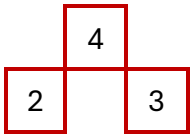

POINTERS MET UP AND THE POINTER VALUE IS IN ORDER WITH THE PIVOT VALUE SO SWAP THE PIVOT WITH THE VALUE TO THE LEFT OF THE POINTER



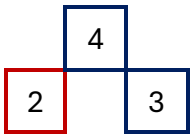

PIVOT POSITION HAS BEEN SORTED SO SECOND SUB-LIST IS DONE



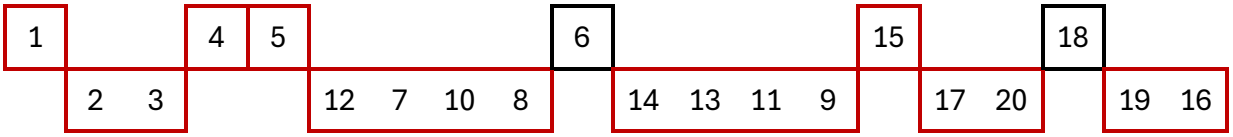

NEW PIVOT POINTS FOR EACH SUBLIST



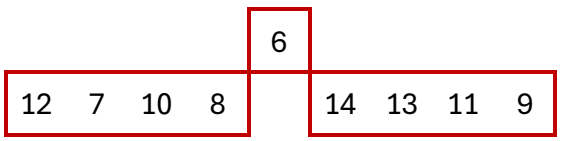

FOCUS ON THE FIRST SUB-LIST



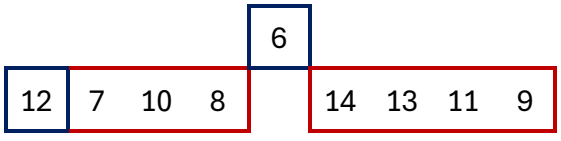

POINTERS MEET AND POINTER VALUE IS OUT OF ORDER WITH PIVOT SO SWAP PLACES



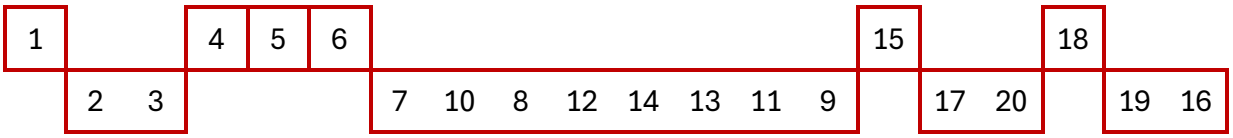

PIVOT POSITION HAS BEEN SORTED SO FIRST SUB-LIST IS DONE



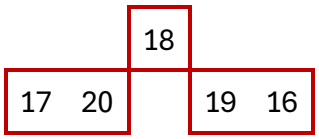

FOCUS ON THE SECOND SUB-LIST



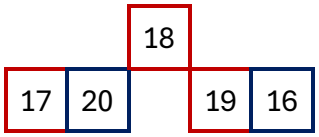

POINTERS MEET AND POINTER VALUE IS OUT OF ORDER WITH PIVOT SO SWAP PLACES



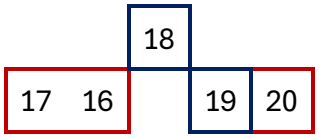

PIVOT POSITION HAS BEEN SORTED SO SECOND SUB-LIST IS DONE



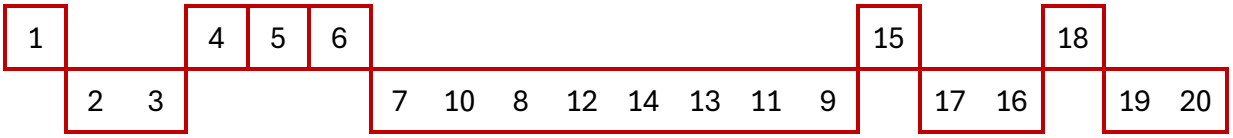

FOCUS ON THE THIRD SUB-LIST



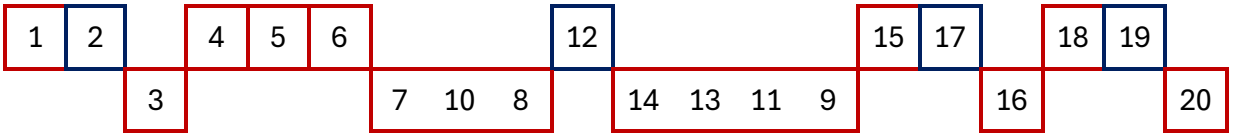

POINTER SCANNING LEFT DETECTS LARGER ITEM THAN PIVOT AND POINTER ON RIGHT DETECTS SMALLER NUMBER THAN PIVOT SO SWAP POINTER VALUES



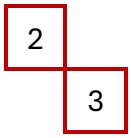

POINTERS MET UP AND THE POINTER VALUE IS IN ORDER WITH THE PIVOT VALUE SO SWAP THE PIVOT WITH THE VALUE TO THE LEFT OF THE POINTER WHICH IS ITSELF



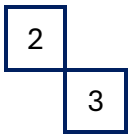

PIVOT POSITION HAS BEEN SORTED SO THIRD SUB-LIST IS DONE



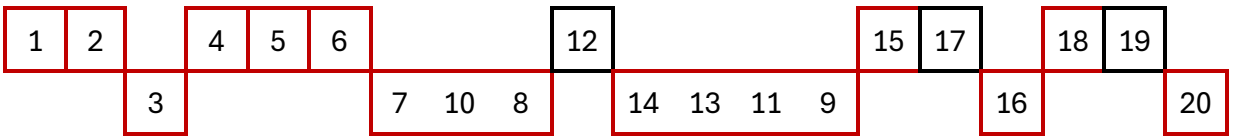

NEW PIVOT POINTS FOR EACH SUBLIST



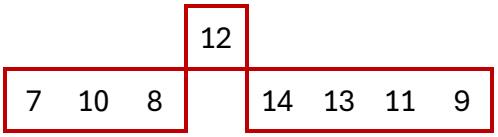

FOCUS ON THE FIRST SUB-LIST



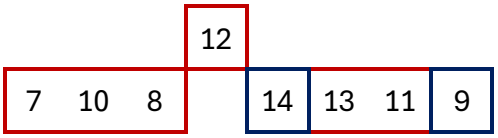

POINTERS MET UP AND THE POINTER VALUE IS IN ORDER WITH THE PIVOT VALUE SO SWAP THE PIVOT WITH THE VALUE TO THE LEFT OF THE POINTER WHICH IS ITSELF



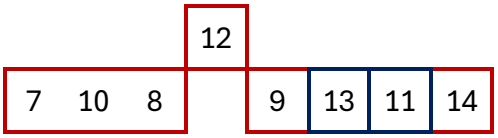

PIVOT POSITION HAS BEEN SORTED SO FIRST SUB-LIST IS DONE



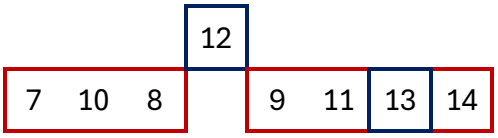

FOCUS ON THE SECOND SUB-LIST



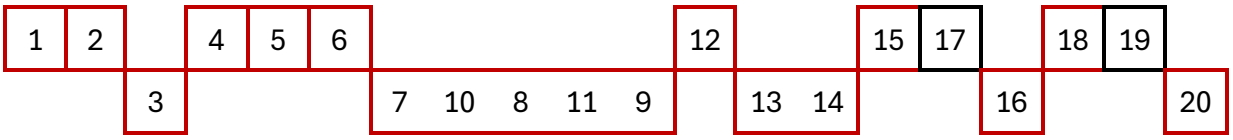

POINTER SCANNING LEFT DETECTS LARGER ITEM THAN PIVOT AND POINTER ON RIGHT DETECTS SMALLER NUMBER THAN PIVOT SO SWAP POINTER VALUES



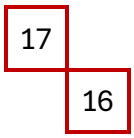

POINTER SCANNING LEFT DETECTS LARGER ITEM THAN PIVOT AND POINTER ON RIGHT DETECTS SMALLER NUMBER THAN PIVOT SO SWAP POINTER VALUES



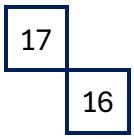

POINTERS MET UP AND THE POINTER VALUE IS IN ORDER WITH THE PIVOT VALUE SO SWAP THE PIVOT WITH THE VALUE TO THE LEFT OF THE POINTER



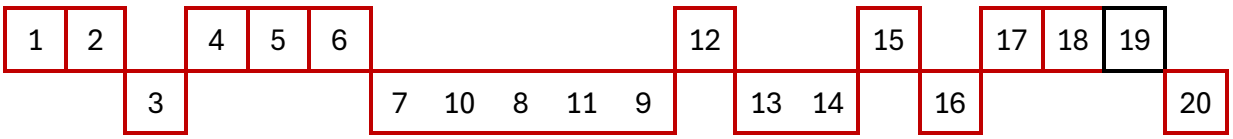

PIVOT POSITION HAS BEEN SORTED SO SECOND SUB-LIST IS DONE



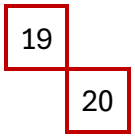

FOCUS ON THE THIRD SUB-LIST



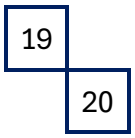

POINTERS MEET AND POINTER VALUE IS OUT OF ORDER WITH PIVOT SO SWAP PLACES



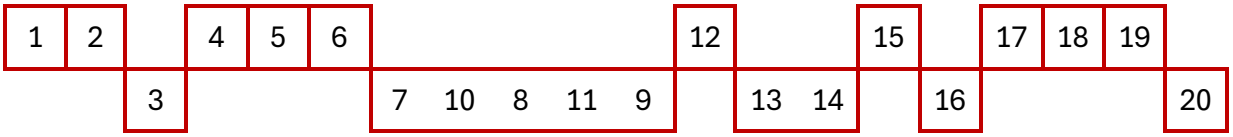

PIVOT POSITION HAS BEEN SORTED SO THIRD SUB-LIST IS DONE



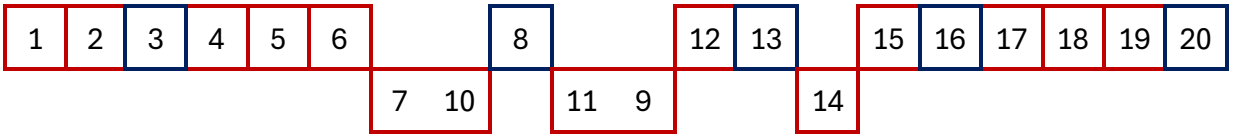

FOCUS ON THE FOURTH SUB-LIST




POINTERS MET UP AND THE POINTER VALUE IS IN ORDER WITH THE PIVOT VALUE SO SWAP THE PIVOT WITH THE VALUE TO THE LEFT OF THE POINTER WHICH IS ITSELF

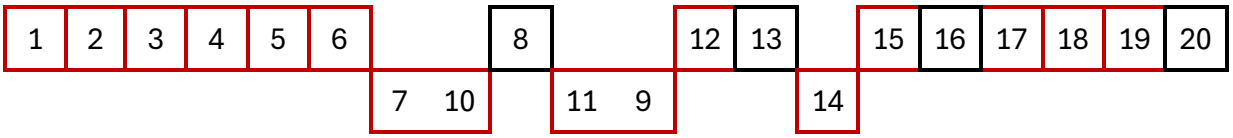



PIVOT POSITION HAS BEEN SORTED SO FOURTH SUB-LIST IS DONE

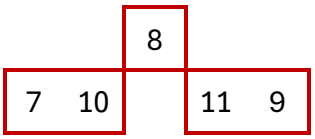



NEW PIVOT POINTS FOR EACH SUBLIST

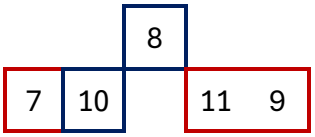

FOCUS ON THE FIRST SUB-LIST



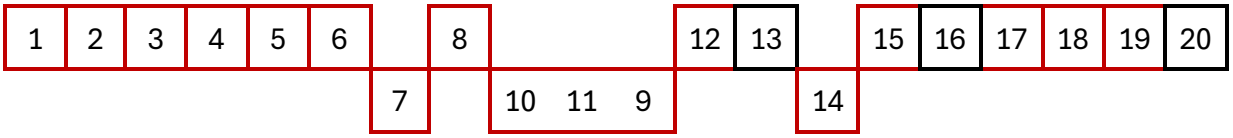

ONLY ITEM IN SUB-LIST EQUALS PIVOT IS IN PLACE SO FIRST SUB-LIST IS DONE



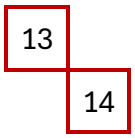

FOCUS ON THE SECOND SUB-LIST



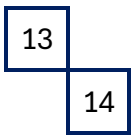

POINTERS MEET AND POINTER VALUE IS OUT OF ORDER WITH PIVOT SO SWAP PLACES



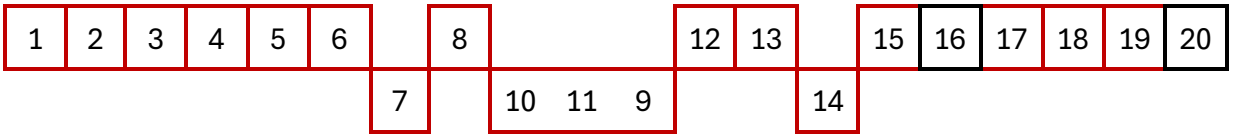

PIVOT POSITION HAS BEEN SORTED SO SECOND SUB-LIST IS DONE




FOCUS ON THE THIRD SUB-LIST

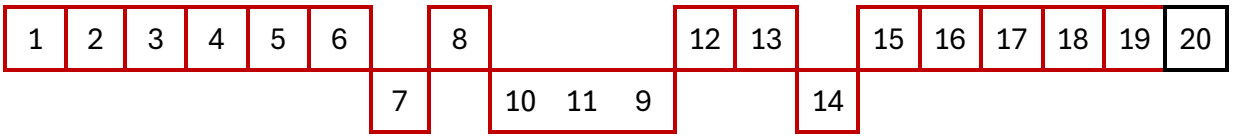



POINTERS MET UP AND THE POINTER VALUE IS IN ORDER WITH THE PIVOT VALUE SO SWAP THE PIVOT WITH THE VALUE TO THE LEFT OF THE POINTER WHICH IS ITSELF



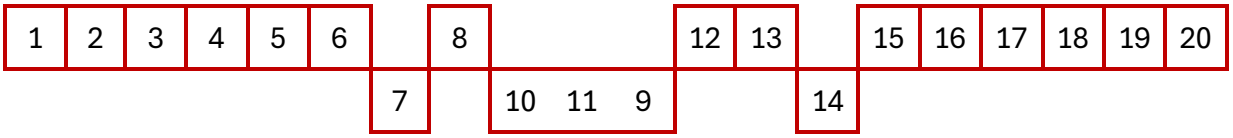

PIVOT POSITION HAS BEEN SORTED SO THIRD SUB-LIST IS DONE


FOCUS ON THE FOURTH SUB-LIST

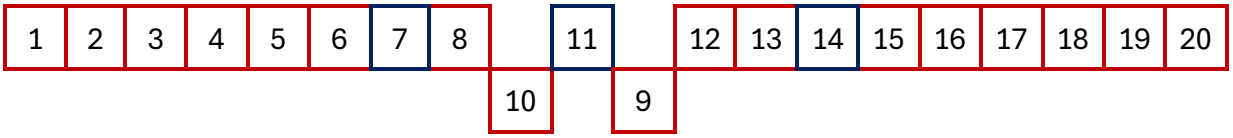



ONLY ITEM IN SUB-LIST EQUALS PIVOT IS IN PLACE SO FOURTH SUB-LIST IS DONE


FOCUS ON THE FIFTH SUB-LIST

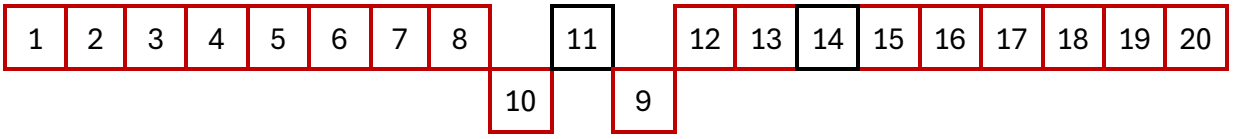



ONLY ITEM IN SUB-LIST EQUALS PIVOT IS IN PLACE SO FIFTH SUB-LIST IS DONE

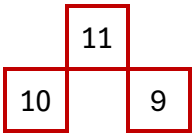



NEW PIVOT POINTS FOR EACH SUBLIST

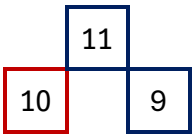

FOCUS ON THE FIRST SUB-LIST



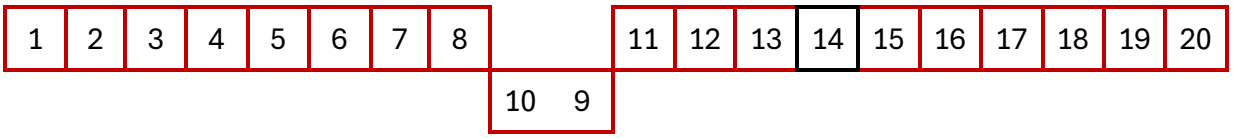

ONLY ITEM IN SUB-LIST EQUALS PIVOT IS IN PLACE SO FIRST SUB-LIST IS DONE




FOCUS ON THE SECOND SUB-LIST

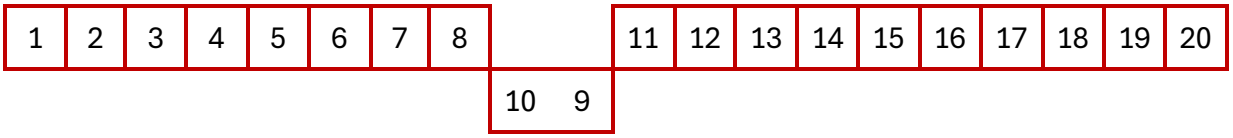



POINTERS MEET AND POINTER VALUE IS OUT OF ORDER WITH PIVOT SO SWAP PLACES

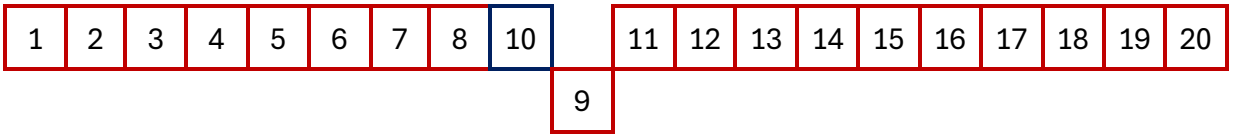



PIVOT POSITION HAS BEEN SORTED SO SECOND SUB-LIST IS DONE

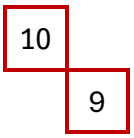

FOCUS ON THE THIRD SUB-LIST



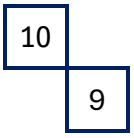

ONLY ITEM IN SUB-LIST EQUALS PIVOT IS IN PLACE SO THIRD SUB-LIST IS DONE



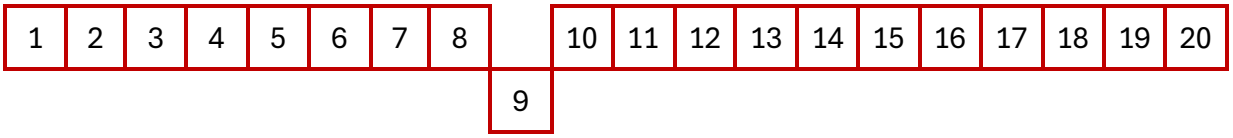

NEW PIVOT POINTS FOR EACH SUBLIST



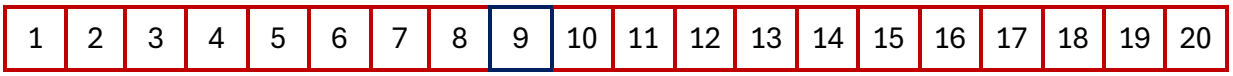

FOCUS ON THE SUB-LIST




POINTERS MEET AND POINTER VALUE IS OUT OF ORDER WITH PIVOT SO SWAP PLACES




PIVOT POSITION HAS BEEN SORTED SO FIRST SUB-LIST IS DONE




NEW PIVOT POINTS FOR EACH SUBLIST

| 9 |
|---|

FOCUS ON THE FIRST SUB-LIST

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

ONLY ITEM IN SUB-LIST EQUALS PIVOT IS IN PLACE SO FIRST SUB-LIST IS DONE

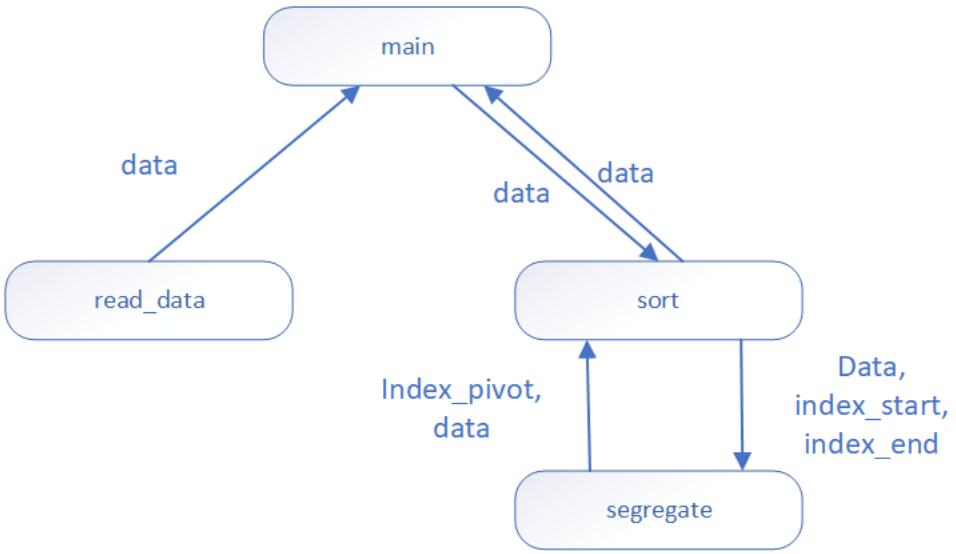
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

LIST HAS BEEN SORTED PROGRAM IS FINISHED

Step 2: Approach

Create a function called segregate() that takes in an array and two index points, named index_start and index_end, as parameters and works on organizing the array from smallest to largest from a pivot point. If there is more than one item in the array, then a pivot is created called index_pivot and it's picked from the middle of the two index points. Starting at index_start search for the first value in the array that is larger than the index_pivot value. After that search for the first value that is smaller than the index_pivot value starting at index_end. Once both values are found, swap the two values and continue searching while index_start is less than index_end. When index_start is equal to index_end, or vice versa, then the pivot needs to be relocated in the array. If the value of index_start is greater than the value of index_pivot, then swap the values of index_start minus one and the value of index_pivot and return index_pivot. If the size of the sub-list is zero or one, then return the list because the sub-list is already sorted. Next, sort the sub-list to the left then right of the pivot until the array is sorted smallest to largest value.

Step 3.1: Structure chart



Step 3.2: Pseudocode

```
Segregate(data, index_start, index_end)
      IF index_start = index_end
            RETUNR index_start

      index_pivot <- (index_start + index_end) / 2

      WHILE index_start < index_end
            WHILE data[index_start] <= data[index_pivot]
                  index_start += 1
            WHILE data[index_end] >= data[index_pivot]
                  index_end -= 1

            IF index_start < index_end
                  SWAP data[index_start], data[index_end]

      IF index_start > index_pivot AND data[index_start] > data[index_pivot]
            index_start -= 1

      SWAP data[index_start], data[index_pivot]

      RETURN index_pivot <- index_start, data
```

Step 4: Copilot

```
FUNCTION segregate(array, index_start, index_end):

    IF index_end - index_start <= 0:
        RETURN array  // Sub-list is already sorted

    SET index_pivot = (index_start + index_end) // 2
    SET pivot_value = array[index_pivot]

    WHILE index_start < index_end:
        // Find first value from the left greater than pivot
        WHILE array[index_start] <= pivot_value AND index_start <
index_end:
            INCREMENT index_start

        // Find first value from the right less than pivot
        WHILE array[index_end] >= pivot_value AND index_start <
index_end:
            DECREMENT index_end

        IF index_start < index_end:
            SWAP array[index_start] WITH array[index_end]

    // At this point, index_start == index_end
    IF array[index_start] > pivot_value:
        SWAP array[index_start - 1] WITH array[index_pivot]
        SET index_pivot = index_start - 1
    ELSE:
        SWAP array[index_start] WITH array[index_pivot]
        SET index_pivot = index_start

    // Recursively sort left and right sublists
    CALL segregate(array, original_index_start, index_pivot - 1)
    CALL segregate(array, index_pivot + 1, original_index_end)

    RETURN array
```

Step 5: Compare and Contrast

The CoPilot solution looks quite similar to mine but it incorporates a recursive section to handle the sub-lists created after the first pass of sorting. I was going to have the sort function handle this step but I see that I could potentially incorporate it into this step or another function entirely. It also handles the final pivot value swap slightly differently and I'm not sure if it's more effective or not than what I did in my code.

Step6: Update

I plan to incorporate the recursive portion into my own code and see if the results are better than what I initially intended to do.

Step 1 By Hand: 180 minutes
Step 2 Approach: 30 minutes
Step 3 Pseudocode and Structure Chart: 120 minutes
Step 4 Copilot: 10 minutes
Step 5 Compare and Contrast: 15 minutes
Step 6 Update: 5 minutes