Exercise 2-1  In-Place QuickSort

Implement an in-place QuickSort algorithm in Rust and test your program on a vector of random numbers, e.g. 3, 6, 7, 5, 2, 1, 4, 8.

In-place means that only a small amount of extra storage space is allowed for auxiliary variables. For example, you should not create a new array for holding the result.

You can use the algorithm below, or use your own variant of the in-place QuickSort algorithm.

Algorithm

function quicksort(array)
  if length(array) > 1
    pivot := select any element of array
    left := first index of array
    right := last index of array
    while left <= right
      while array[left] < pivot
        left := left + 1
      while array[right] > pivot
        right := right - 1
      if left <= right
        swap array[left] with array[right]
        left := left + 1
        right := right - 1
        quicksort(array from first index to right)
        quicksort(array from left to last index)

Test your solution

• The resulting vector should be sorted.

• Your solution is most likely in-place if you use the macro vec! only once.

Hints

• Generic types would require the use of type traits which have not been covered in the lecture yet. Suggestion: Use the concrete type i32 for the exercise.

• Side effects are essential for an in-place solution. Therefore, the vector to be sorted must be declared mutable.

• Use mutable references for passing the vector to a different function.

• For swapping values in a vector you can use the predefined method swap. For documentation and an example see