We mentioned quick sort several times in the past.

This lecture explains the quick sort algorithm.

Quick sort is quick because it uses the principle of transitivity. Transitivity says that if ei is greater than bee and bee is greater than cee, then ei is greater than cee. It is not necessary comparing ei and cee.

Quick sort finds a reference value, called the pivot. Usually, it is the first element in the array. You can also choose another element in the array as the pivot.

Quick sort divides the array into three parts: the part that is smaller than the pivot, the pivot, and the part greater than the pivot. Because of transitivity, the first and the third parts do not need to be compared.

These are the steps for quick sort. First, find a pivot. Usually we select the first element in the array as the pivot. Then, we keep two indexes called low and high. The low index starts from the second element in the array. The high index starts from the last element in the array.

The low index moves towards the high index if the array element is smaller than the pivot. The movement stops when the array element is greater than the pivot.

The high index moves towards the low index if the array element is greater than the pivot. The movement stops when the array element is smaller than the pivot.

If the low index and the high index have not crossed each other, swap the two array elements.

This process continues until the two indexes cross. Then, move the pivot.

Now, the array is divided into three parts. Sort the first part and the third part.

Let’s go through an example to see how this works.

Consider this array. We select the first element as the pivot. The value is 19. The low index is set to the index of the second element. The high index is the index of the last element.

The algorithm compares the element at the low index. The value is 7, smaller than the pivot 19. Thus, the low index increases by one.

The element at the low index is 12. It is also smaller than the pivot 19. The low index increases by one.

The element at the low index is 23. It is also greater than the pivot 19. The low index does not move.

Next, consider the high index. The element at the high index is 33. It is greater than the pivot 19. The high index decreases by one.

The element at the high index is 51. It is greater than the pivot 19. The high index decreases by one.

The element at the high index is 16. It is smaller than the pivot 19. The high index does not move.

Next, swap the values at the low and the high indexes. The value 16 is moved to the low index. The value 23 is moved to the high index.

Then, we can continue this process. The value 16 is smaller than the pivot and the low index increases by one.

The value at the low index is 8. It is smaller than the pivot 19. The low index increases by one.

The value at the low index is 31. It is greater than the pivot 19. The low index does not move.

Now, we consider the high index. The value is 23 and it is greater than the pivot 19. The high index decreases by one.

The value at the high index is 28. It is greater than the pivot. The high index decreases by one.

The value at the high index is 42. It is greater than the pivot. The high index decreases by one.

The value at the high index is 6. It is smaller than the pivot. The high index does not move.

The two values 31 and 6 are swapped.

Continue this process. The low index increases by one. The high index decreases by one.

The two indexes have crossed. Stop.

The next step swaps the pivot so that the pivot is between the two parts.

We have successfully divided the array into three parts. The first part is smaller than the pivot. The second part is the pivot. The third part is greater than the pivot.

We need to sort the first and the third parts. Because of transitivity, we do not need to compare the values in the first part with the values in the third part.

Let’s see how to write this in C language. The quick sort function takes two arguments: the starting address of the array and the length of the array.

The function calls a helper function that has three arguments.

The quick sort function has only two arguments: the array and the length. The helper function has three arguments: the array, and two additional arguments to keep track of the low and high indexes.

Also, the helper function is static. That means this helper function is visible within the same file. The helper function cannot be called by another function outside the file.

This helper function is a recursive function. At the top, the stop condition is when the array has only one or no element. If the low index is the same as the high index, the array has only one element. Sorting is not necessary. If the low index is greater than the high index, the array has no element and sorting is also unnecessary.

If the stop condition is not met, the function initializes three variables: The pivot is the first element of the array. The low index is the index of the second element. The high index is the index of the last element.

Before we continue reading the recursive function, this slide summarizes the concept of helper function.

Sometimes, we need to adjust the arguments in functions to keep track of the progress of recursion. Usually, we need more arguments. In the case of quick sort, the original function has only two arguments: the array and the size. The beginning index starts from zero and thus is unnecessary.

The helper function has three arguments: the array, the low index, and the high index.

The helper function is usually declared as a static function. A static function can be called from a function inside the same file. A static function can be recursive.

A static function cannot be called by any function in another file. This reduces the chance when this static function is accidentally called.

You can use static functions, not static variables.

Do not get confused between a static function and a static variable. A static variable keeps the value from the previous call. As a result, a static variable can be very confusing because a function’s behavior depends on the arguments from previous calls.

Don’t use static variables.

Let’s go back to the recursive function for quick sort.

There are three while loops. The outside while loop checks whether low and high indexes have crossed.

Inside, the first while loop increases the low index as long as the array elements are smaller than the pivot. This while loop stops when an array element is greater than the pivot.

The next while loop decreases the high index as long as the array elements are greater than the pivot.

If the two indexes have not crossed yet, swap the two array elements.

This process continues until the low index and the high index cross.

Next, move the pivot. We need to check whether it is necessary moving the pivot because it is possible that the pivot is already the smallest value in the array. In that case, we should not move the pivot.

After this process, the array should have one part that is smaller than the pivot and another part that is greater than the pivot.

The function calls itself with the new low index and high index to sort these two parts separately.

As we explained earlier, due to transitivity, there is no need to compare the values between these two parts.