# Playbook Notes: Quick Sort

## Introduction:

The quick sort is a sorting algorithm that will take an array of integers (It could also be floats, doubles, or shorts) and sort the data so that after the call the quick sort function, all the data will be in numeric order.

## Let us start with Big Picture:

Select a Pivot (For this example we will pick the end of the array)

The point of the pivot:

After the first pass through the array, all numbers less than the pivot will be to the left of the pivot and all numbers that are greater than the pivot are to the right of the pivot. The pivot will be properly placed.

**Before the first pass:**

To start, all the data is unsorted.

| Array before calling | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| X | X | X | X | X | X | X | X | Z |

The elements here are not trivial, just know that all the values are random and are not in any kind of order.

The value at item 8, use I have used the letter Z. This is because the last item in the array will be our pivot. The number itself is still random.

**After the first pass**

| Array before calling | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| X | X | X | X | Z | Y | Y | Y | Y |

In the above picture, the pivot value is in element 4, but this is not a given, it could be in any position.

What is true is that all values to left of the pivot (X values) are less than Z and all values to the right of the Z (the Y values) are greater than the pivot.

After the first pass, we will call quicksort recursively with the first half of the array and then we will call quick sort recursively with the second half of the array.

Every call will receive a reduced number of elements to sort.

This will continue until the entire array is sorted.

## Looking at the Code:

**The prototype:**

void QuickSort(int array[], startIndex, endIndex);

Note: The endIndex would be the array length – 1.

**Writing the code:**

If (start < end) then do the following (This is testing to see if we do the recursive case):

Declare an integer called pivot set it equal to a function call of the Partition function. (This is defined below)

When this function returns, all values to left of the pivot will be less than the pivot and all values to the right of the pivot will be greater than the pivot.

Call QuickSort recursively passing in the start of the array and pivot -1. This will sort the left side of the pivot.

Call QuickSort recursively passing in pivot + 1 and end. This will sort the right side of the pivot.

**The Partition Function:**

**The Prototype**:

int Partition(int array[], int start, int end);

**Writing the code:**

Create an integer called pivot and set it to the end. (Remember we said earlier that we will always start with the pivot at the end)

Declare two more integer variables. I will call one the *lag* and the other *lead*. (Examples of the quick sort will often use names like i and j, but you know how feel about one letter variable names).

Set lag to start – 1

Set lead to start (This will be our LCV and may be part of a for-loop)

Loop until lead is equal to the index value of the end:

Check the to see if the value at lead is less than our pivot value.
If this is true:

- increase the lag by 1
- Swap the values at lead and lag.

Increase lead by 1.

After the loop is done executing, increase lag variable by 1.

Swap the value at the pivot with the value at the lag location.

Return the lag value (return the index, not the value at the lag location)