Merge sort
Merge sort is a divide and conquer algorithm, we will pass an array into a sort function, this function will divide our array into two. We have a left and right array, these will be sub-arrays(copied information from the original array into the sub arrays).

Merge sort is a recursive function, we will call it again and repeat. We will stop when the size of the sub array is one.

Merge is a helper for merge sort, it will accept the total of 3 arguments.
Left sub array,Right sub array and the original sub array these came from. Merge will place them back into the sub array they came from in correct order. This will repeat til we connect with the original array.

You usually start with one branch at the time , where you take the left side all the way down to end. Then start the sorting, then you go for the next branch to the right.
So when both the Left[4] and Right[4] is sorted it will place it back into the original array sorted

Original array[8]

Left[4] right[4]
Left[2] Left[2] Right[2] Right[2]
Left[1] left[1] Left[1] Left[1] Right[1] Right[1] Right[1] Right[1]

QuickSort
We need an array or another type of collection that we can pass into the quicksort function, after passing our argument into the quick sort function we need to pick a pivot(there are different types of way to choose,either start,mid,end) but the standard is to place the pivot at the end.
We try to find the final location for the pivot, we will declare and use two indices (i and j)

J starts at the beginning of the array.
I will be once less than the beginning.

You will also create a temp that we can use to swap values. The sorting starts with checking if the value at J is less than pivot, if it's greater or equal to pivot value we ignore it . if the current j value is greater we increment j with one (j++). Repeat the check on J, if J is less than pivot value increment i with one(i++). Then swap the values on I and J, here is where you get the use of the temp.

1. Assign the value at i on temp
2.assign the value at j on I
3.assign temp value to j.

Then we can move to the next in iteration, increment j and repeat the checking. Repeat til J reaches the pivot

When j reaches the pivot, we know where the final resting place for the pivot is going to be.It's going to be I + 1. Swap the value at i with the value at pivot.

To check this, all the elements on the left side should be less than our pivot.Alll elements on the right should be greater/equal than our pivot(They are still not in order)

Now we need to create two partitions, first partition is the left side of the array(up to the pivot, not including the pivot) and the second partition will be the right side(everything after the pivot,not including the pivot).

Quicksort is a recursive algorithm(divide and conquer). We need to pass these partitions as arguments into the quicksort function. Unlike with merge sort where we create subarrays, in quicksort we will be sorting these arrays in place. We need to keeptrack of the beginning and the ending indices of these partitions. Then you just repeat the algorithm again.

Heap sort
Heap: ordered binary tree
Max heap : parent > child

There is usually some other functions that are needed to use heap sort

- Build-max-heap
    - Creates max heap from unsorted array
- Heapify
    - Similar to build-max-heap,but assumes part of the array is already sorted.

1.Create max heap
2.Remove largest item
3.Place item in sorted partition.

We assume the array given is unsorted so we use the function build-max-heap. Now they are sorted with the highest value at the root. We swap the highest and lowest value then remove the highest from the tree. Now we are back to having a tree and not a heap, now we call heapify since the lowest value is on the wrong spot. Now the lowest value goes to the bottom again and the highest goes to the top. Now we repeat.