Algorithms Chapter 8 Assignment

* Read **Chapter 8** in the McAllister text.
* Complete as many of the **Knowledge Exercises** as you can / want to.
* Write a **Short Essay** regarding the following topics:
  + Since we've already written a Merge Sort, I want you to focus on, and write about, **Quicksort**.
    - Merge Sort and QuickSort are the fastest general sorting algorithms known for unconstrained data sets.
  + Analyze both Merge Sort and QuickSort.
    - What are are the best case and worst case for each?
    - When do each of them do well, and when do each of them do poorly?
  + Also, as an afterthought, is it ever useful to use an [O(N^2)](https://en.wikipedia.org/wiki/Big_O_notation) sorting algorithm like Bubble Sort? When? Why?
* You may, if you wish, also code a QuickSort, to compare it to Merge Sort. But this is **optional**.

Like Merge Sort, Quick Sort also uses the divide and conquer approach. However, in Merge Sort most of the work isn’t being done in the divide step, but is done during the combine step. Quick Sort on the other hand performs all of the real work in the divide step, while no real functionality occurs in the combine step. **Quicksort’s Cases** 🡪 best: O(n log(n)), average: O(n log(n)), worst: O(n^2). **Merge Sort’s cases** 🡪 best: O(n log(n)), average: O(n log(n)), worst: O(n log(n)).

Even though Quick Sort has O(n^2) in worst case, this can usually be avoided. It’s cache performance is much higher than other sorting algorithms, and the operations of its inner loop are much simpler. Therefore, when implemented appropriately Quick Sort is generally a much faster sorting algorithm, and especially on an unsorted array. Merge Sort performs irrespectively the same whether data is sorted or not. Quick Sort also performs better when the data is stored in memory, whereas when the data is stored on an external device, Merge Sort would be the clear winner in terms of speed. I think overall Quick Sort is much more complicated than Merge Sort, and for this reason someone aiming for simplicity and maintainability might choose Merge Sort instead.

Divide:

* Choose any element in array, call this element pivot
* Rearrange array elements so that:
* all values less than the pivot are on its right
* all values greater than the pivot are on its left
* still have an unsorted list, but with elements that are generally on the correct side of the pivot

Conquer:

* recursively sort the two subarrays:
* elements to the left of the pivot (less than)
* elements to the right of the pivot (greater than)