REMINDER: Programs need to be:

* Correct: produce required results for valid inputs
* Reliable: behave sensibly for invalid inputs/ errors
* Efficient: give results quickly (even for large inputs)
* Maintainable: code is clear and well-structured

3 sorting functions for arrays:

* Random sort: choose random pairs of items, swap it out of order (RUBBISH)
* Bubble sort: repeatedly scan array, swapping out of order items (NOT GREAT)
* Quick sort: partition array into big / small, recursively sort partitions. (A LITTLE BETTER)
* You can compare performance of functions by time measurements.

IMPORTANT - Analysing and determining performance results:

* **Here is a set of results 🡪 Why did these results occur (in my opinion) 🡪 Based on that, what kind of algorithm might have produced those results.**
* You can mix things up
* **I’ve hypothesised that I’ve got this kind of algorithm 🡪 This algorithm should exhibit these kind of performance characteristics 🡪 It is exactly the same set of performance characteristics that I have observed when I test my program**

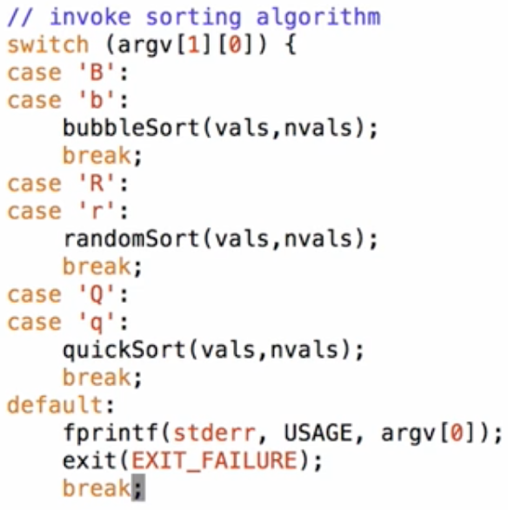
**Sort(a, n)** - What does SORTING mean?

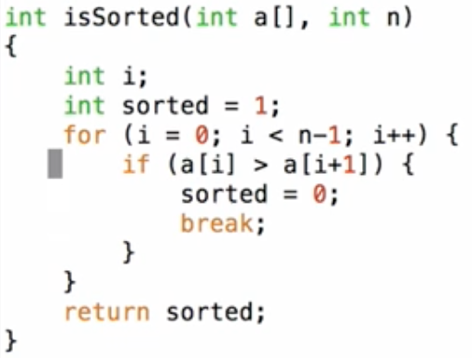
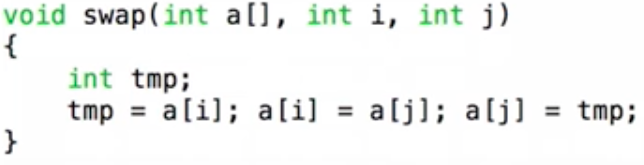
* PRECONDITION: Sorting is a function, where we sort an array a of n elements.
* POSTCONDITION: Forall i [0 … n – 1], a[i] <= a[i + 1]
  + Elements must be in ascending order
  + Duplicates are allowed.

./sorter X 5

./sorter X 6 < numbers

./sorter X ‘wc –l < numbers’ < numbers





Implementing a SWAP function, which is usually needed to do sorting.

Determine if array is already sorted.  
Return 1 for yes, 0 for no.

randomSort Function

