quicksort

- can also be a very efficient approach for sorting large collections of data stored in an array
- guess a value that (hopefully) winds up near middle of the sorted data, called the ***pivot*** value
- rearrange contents so all values are on correct side of pivot (smaller on one side, large on other), called the ***partition*** step

  need a good way to pick pivot and perform partitioning
- repeat recursively on two sides:

  call quicksort on the side of the array with smaller values

  call again on side with larger values

picking the pivot

- given a portion of an array to sort, e.g. sort array contents from positions low through high (inclusive)
- ideally pick a pivot that will wind up close to the middle of the sorted results
- this difficult to do efficiently in practice
- pick a random element from the section we're sorting:
  - gives decent behaviour on average, but ...
  - if we're consistently unlucky it can be as bad as bubblesort
- pick element in “low” position
  - easy, but gives poor efficiency if array segment already sorted

partitioning algorithm

```
int partition(float arr[ ], int low, int high)
   if (low >= high)
        return // no sorting needed
   pivotPos = low
   pivotVal = arr[low]
   low++  // since we have the pivotVal in position low
   repeat
        repeatedly increment low until we reach high OR we reach a value > pivotVal
        repeatedly decrement row until we pass low OR we reach a value < pivotVal
        if low/high haven't passed each other
               then swap what's in position low with what's in position high
               (we've found two things that are each on the wrong side of the pivot value)
    swap the pivot value, in position pivotPos, with what's in position high
        (since high has crossed low it is now sitting on a value which is < pivotVal,
        so this puts pivotVal into its correct final position in the array and has all
        the other values correctly positioned on one side or the other)
    return the position the pivot value wound up in
```

partitioning example

- partition (7, 3, 9, 8, 4, 6)
- pick 7 as pivot, mark low and high positions with *
  - move low up until value > 7, i.e. when reaches 9
  - move high down until value < 7, i.e. the 6 right away
  - swap the 9 and 6
- resume with (7, 3, *6, 8, 4, *9)
  - low goes up to the 8, high comes down to the 4, swap again
- resume with (7, 3, 6, *4, *8, 9)
  - low and high cross right away, leaves loop
- swap the 7 and the 4, giving (4, 3, 6, *7, 8, 9)
  - puts 7 in right spot, everything else on correct sides

quicksort algorithm

```
quicksort(int arr[], int lower, int upper)

  if lower >= upper
    return // at most one element, no sorting needed

  // let partition pick a pivot and rearrange, tell us where pivot wound up
  pos = partition(arr, lower, upper)

  // sort the side below the pivot
  quicksort(arr, lower, pos-1)

  // sort the side above the pivot
  quicksort(arr, pos+1, upper)
```