1. *Quick sort* is another divide-and-conquer, advanced sort that often is quicker than merge sort (hence its name). The general idea is as follows. Assume “n” items to sort.
   - Select a “random” item in the unsorted part as the *pivot*
   - Rearrange (called *partitioning*) the unsorted items such that:

<table>
<thead>
<tr>
<th>Pivot Point</th>
<th>All items &lt; Pivot</th>
<th>Pivot Item</th>
<th>All items &gt;= Pivot</th>
</tr>
</thead>
</table>

   - Quick sort the unsorted part to the left of the pivot
   - Quick sort the unsorted part to the right of the pivot

```c
int partition(int myArray[], int low, int high) {
    int pivotItem, pivotPoint, scan;
    pivotPoint = low;
    pivotItem = myArray[low];
    for (scan = low + 1; scan <= high; scan++) {
        if (myArray[scan] < pivotItem) {
            pivotPoint++;
            swap(myArray[pivotPoint], myArray[scan]);
        } // end if
    } // end for
    swap(myArray[low], myArray[pivotPoint]);
    return pivotPoint;
} // end partition
```

b) For the array below, trace the first call to partition and determine the resulting array, and value returned.

```
myArray:  0  1  2  3  4  5  6  7  8
          54 26 93 17 77 31 44 55 20
```

a) Complete the recursive quickSort code:

c) What initial arrangement of the array would cause partition to perform the most amount of work?

d) Let “n” be the number of items between low and high. What is the worst-case $\Theta(\ )$ for partition?
e) In the worst case, the pivot point is on the right side. What would be the overall, worst-case $\Theta(\,)$ for Quick Sort?

- $\text{total # partition loops}$
- $\text{pivotPoint}$
- $\text{pivotPoint}$
- $\text{pivotPoint}$
- $\text{pivotPoint}$

$\text{+ ________}$

f) Ideally (best-case), the pivot item always splits the array into two equal size problems. What would be the best-case $\Theta(\,)$ for Quick Sort?

- $\text{total # partition loops}$
- $\text{pivotPoint}$
- $\text{pivotPoint}$
- $\text{pivotPoint}$
- $\text{pivotPoint}$
- $\text{pivotPoint}$
- $\text{pivotPoint}$
- $\text{pivotPoint}$

$\text{+ ________}$

g) Why do some versions of the partition code select the middle item of the array as the pivot?

h) For small problem instances (e.g., $n = 100$), a simple sort might be faster than an advanced sort. How might you determine this optimal threshold?