1. *Quick sort* is another 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 Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>All items &lt; to Pivot</td>
</tr>
</tbody>
</table>

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

a) What base case(s) would we have?

b) Because of the recursive nature of quick sort, what “extra” parameters would we need to specify the part of the list to sort?

c) Write the code for quick sort. Assume there is a partition function:

```c
int partition(int set[], int start, int end)
```

Partition: (1) selects a value in the middle of the array `set` as the pivot, rearranges the values from start to end such that all the values less than the pivot are on its left and all the values greater than or equal to the pivot are on its right. Partition returns the index of the pivot after this rearrangement.
int partition(int set[], int start, int end) {
    int pivotValue, pivotIndex, mid;

    mid = (start + end) / 2;
    swap(set[start], set[mid]);
    pivotIndex = start;
    pivotValue = set[start];
    for (int scan = start + 1; scan <= end; scan++) {
        if (set[scan] < pivotValue) {
            pivotIndex++;
            swap(set[pivotIndex], set[scan]);
        } // end if
    } // end for
    swap(set[start], set[pivotIndex]);
    return pivotIndex;
} // end partition

// swap simply exchanges the contents of value1 and value2.
void swap(int &value1, int &value2) {
    int temp = value1;
    value1 = value2;
    value2 = temp;
} // end swap

d) Given the above partition and swap functions, trace your quickSort function on the following array.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>35</td>
<td>10</td>
<td>40</td>
<td>45</td>
<td>20</td>
<td>25</td>
<td>50</td>
</tr>
</tbody>
</table>