1. The Quick Sort code from the text is given below:

```python
def quickSort(alist):
    quickSortHelper(alist, 0, len(alist) - 1)

def quickSortHelper(alist, first, last):
    if first < last:
        splitpoint = partition(alist, first, last)
        quickSortHelper(alist, first, splitpoint - 1)
        quickSortHelper(alist, splitpoint + 1, last)

def partition(alist, first, last):
    pivotvalue = alist[first]

    leftmark = first + 1
    rightmark = last

    done = False
    while not done:
        while leftmark <= rightmark and alist[leftmark] < pivotvalue:
            leftmark = leftmark + 1

        while alist[rightmark] > pivotvalue and rightmark >= leftmark:
            rightmark = rightmark - 1

        if rightmark < leftmark:
            done = True
        else:
            alist[leftmark], alist[rightmark] = alist[rightmark], alist[leftmark]
            alist[first], alist[rightmark] = alist[rightmark], alist[first]

    return rightmark
```

a) For the list below, trace the first call to partition and determine the resulting list.

```
alist: 54 26 93 17 77 31 44 55 20
```

b) What would be the parameters to the recursive calls from within the initial quickSortHelper?
c) What initial arrangement of the list would cause the Quick Sort code to run the longest?

d) What would be the big-oh for Quick Sort in the worst case?

e) Can you suggest a modification that might help in the worst case scenario above?

f) What would be the big-oh for Quick Sort in the best case?

g) What would be the big-oh for Quick Sort in the best case?