**Quick Sort Algorithm – Version 2**

**Original Unsorted Array:**  
[12] [22] [14] [16] [54] [32] [60] [92] [88]  
  
**Call 1: Quicksort (0, 8, List);  
low = 0, high = 8**  
**Use median algorithm**: middle = (0 + 8) / 2 = 4  
  
Order values at low, middle and high (0, 4, 8):  
  
[12] [22] [14] [16] [54] [32] [60] [92] [88]  
  
Swap values at low and middle to place median in first cell:  
  
[54] [22] [14] [16] [12] [32] [60] [92] [88]  
  
Pivot = 54, Low = 22, High = 88  
  
Left is less than or equal to right, so enter loop.  
Move left to the right until it points to a value greater than pivot's (or hits high).  
Move right to the left until it points to a value less than pivot's (or passes left).  
  
[54] [22] [14] [16] [12] [32] [60] [92] [88]  
 P L R  
  
Left is still less than or equal to right, so enter loop again.  
Move left to the right until it points to a value greater than pivot's (or hits high).  
Move right to the left until it points to a value less than pivot's (or passes left).  
  
[54] [22] [14] [16] [12] [32] [60] [92] [88]  
 P L R  
  
Left is still less than or equal to right, so enter loop again.  
Move left to the right until it points to a value greater than pivot's (or hits high).  
Move right to the left until it points to a value less than pivot's (or passes left).  
  
[54] [22] [14] [16] [12] [32] [60] [92] [88]  
 P L R  
  
Left is still less than or equal to right, so enter loop again.  
Move left to the right until it points to a value greater than pivot's (or hits high).  
Move right to the left until it points to a value less than pivot's (or passes left).  
  
[54] [22] [14] [16] [12] [32] [60] [92] [88]  
 P R L  
  
Since right is no longer greater than left, exit loop. All smaller values are on the left and all bigger values are on the right.  
  
Now move the pivot point between the two partitions, by swapping the values pointed to by pivot and right.

[12] [22] [14] [16] [54] [32] [60] [92] [88]  
 P R L

Call 2a to sort low to right – 1 (0 to 3):  
Call 2b to sort right + 1 to high (5 to 8):  
  
Call 2a: Quicksort (0, 3, List);

[12] [22] [14] [16]

Since high is greater than low, compute the listSize.

listSize = 4 is NOT greater than 4, so use Insertion Sort

**Pass 1:**  
No swap  
[12] [22] [14] [16]

**Pass 2:**  
InsertValue = 14  
  
[12] [22] [22] [16]

[12] [14] [22] [16]  
  
**Pass 3:**  
InsertValue = 16  
[12] [14] [22] [22]

[12] [14] [16] [22]

List is sorted. Return to Call 1.

Call 2b to sort right + 1 to high (5 to 8):

[32] [60] [92] [88]  
  
Since high is greater than low, compute the listSize.

listSize = 4 is NOT greater than 4, so use Insertion Sort

**Pass 1:**  
No swap  
[32] [60] [92] [88]

**Pass 2:**  
No swap  
[32] [60] [92] [88]  
  
**Pass 3:**  
InsertValue = 88  
[32] [60] [92] [92]

[32] [60] [88] [92]

List is sorted.

Call 3a: Quicksort (4, 5, List);  
  
Since high is greater than low, compute the listSize.

listSize = 2 is NOT greater than 4, so use Insertion Sort

**Pass 1:**  
InsertValue = 32

[54] [54]

[32] [54]

List is sorted.  
  
Call 3b: Quicksort (6, 7, List);  
  
Since high is greater than low, compute the listSize.  
  
listSize = 2 is NOT greater than 4, so use Insertion Sort

No swaps occur. List is sorted.

Call 3c: Quicksort (7, 8, List);  
  
Since high is greater than low, compute the listSize.  
  
listSize = 2 is NOT greater than 4, so use Insertion Sort

No swaps occur. List is sorted.

Return to Call 2b.

All recursive calls have now returned.

Sorted List:  
[12] [14] [16] [22] [32] [54] [60] [88] [92]