Quick Sort Algorithm: Iterative vs Recursive

Quick Sort: Full C Code and Explanation

```
// Utility function to swap two integers
void swap(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = t;
}

// Partition function used in both versions
// It chooses the last element as the pivot and arranges
// elements smaller than the pivot to its left, and larger to its right.
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;  // Index of smaller element

    for (int j = low; j <= high - 1; j++) {
        // If current element is smaller than the pivot
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }

    swap(&arr[i + 1], &arr[high]);
    return (i + 1); // Return the partitioning index
}

// Iterative Quick Sort
// Instead of recursion, this uses a manual stack to simulate recursive calls.
void quickSortIterative(int arr[], int l, int h) {
    int* stack = (int*)malloc(sizeof(int) * (h - l + 1));
    int top = -1;

    stack[++top] = l;
    stack[++top] = h;

    while (top >= 0) {
        h = stack[top--];
        l = stack[top--];

        int p = partition(arr, l, h);

        // If there are elements on the left side of the pivot
        if (p - 1 > l) {
            stack[++top] = l;
            stack[++top] = p - 1;
        }
```

Quick Sort Algorithm: Iterative vs Recursive

```
        // If there are elements on the right side of the pivot
        if (p + 1 < h) {
            stack[++top] = p + 1;
            stack[++top] = h;
        }
    }

    free(stack); // Free memory used by the stack
}

// Recursive Quick Sort
// This is the traditional version that uses function call stack
void quickSortRecursive(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);

        // Sort elements before and after partition
        quickSortRecursive(arr, low, pi - 1);
        quickSortRecursive(arr, pi + 1, high);
    }
}

// Function to print the array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Main function to demonstrate both versions
int main() {
    int arr1[10] = {34, 7, 23, 32, 5, 62, 78, 1, 9, 12};
    int arr2[10];

    // Copy original array to another for fair comparison
    for (int i = 0; i < 10; i++) arr2[i] = arr1[i];

    printf("Original array:\n");
    printArray(arr1, 10);

    quickSortIterative(arr1, 0, 9);
    printf("Sorted using Iterative Quick Sort:\n");
    printArray(arr1, 10);

    quickSortRecursive(arr2, 0, 9);
    printf("Sorted using Recursive Quick Sort:\n");
    printArray(arr2, 10);

    return 0;
```

Quick Sort Algorithm: Iterative vs Recursive

```
}
```

Comparison Summary

```
Comparison between Recursive and Iterative Quick Sort

1. Time Complexity:
   - Both: O(n log n) on average, O(n^2) worst-case.
   - Best Case: O(n log n)

2. Space Complexity:
   - Recursive: O(log n) due to call stack.
   - Iterative: O(log n) to O(n) due to manual stack.

3. Ease of Implementation:
   - Recursive: Easier to understand and implement.
   - Iterative: More complex but avoids stack overflow.

4. Use Cases:
   - Recursive is preferred for simplicity and readability.
     - Iterative is useful in low-memory environments or where deep recursion isn't allowed.
```