```cpp
#include <iostream>
#include <vector>
#include <chrono>
using namespace std;
using namespace std::chrono;

void insertionSort(vector<int> &arr, int left, int right) {
    for (int i = left + 1; i <= right; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= left && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

void merge(vector<int> &arr, int left, int mid, int right) {
    int n1 = mid - left + 1, n2 = right - mid;
    vector<int> L(n1), R(n2);
    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(vector<int> &arr, int left, int right, int threshold) {
    if (right - left + 1 <= threshold) {
        insertionSort(arr, left, right);
        return;
    }
    if (left < right) {
        int mid = left + (right - left) / 2;
```

```
        mergeSort(arr, left, mid, threshold);
        mergeSort(arr, mid + 1, right, threshold);
        merge(arr, left, mid, right);
    }
}

int partition(vector<int> &arr, int low, int high) {
    int pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}

void quickSort(vector<int> &arr, int low, int high, int threshold) {
    if (high - low + 1 <= threshold) {
        insertionSort(arr, low, high);
        return;
    }
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1, threshold);
        quickSort(arr, pi + 1, high, threshold);
    }
}

void testModifiedSortFunction(void (*sortFunc)(vector<int> &, int, int, int), vect
    auto start = high_resolution_clock::now();
    sortFunc(arr, 0, arr.size() - 1, threshold);
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<nanoseconds>(stop - start);
    cout << "Time taken by function with threshold " << threshold << ": " << durat
}

int main() {
    vector<int> sizes = {1000, 10000, 100000, 1000000};
    vector<int> thresholds = {2, 5, 10, 20, 50};
    for (int size : sizes) {
        cout << "Testing with input size: " << size << endl;
        vector<int> arr(size);
        for (int i = 0; i < size; i++) {
            arr[i] = rand() % 10000;
        }
        for (int threshold : thresholds) {
            vector<int> arrCopy = arr;
            cout << "Threshold: " << threshold << " | Modified Recursive Merge Sor
            testModifiedSortFunction(mergeSort, arrCopy, threshold);

            arrCopy = arr;
            cout << "Threshold: " << threshold << " | Modified Recursive Quick Sor
            testModifiedSortFunction(quickSort, arrCopy, threshold);
        }
        cout << "---------------------------------" << endl;
```

```
    }
    return 0;
}
```

1. Terminal Condition(s) Chosen:

The terminal condition for recursion in both mergeSort and quickSort is based on the size of the subarray being sorted. Specifically, if the number of elements in the subarray (i.e., right - left + 1) is less than or equal to a given threshold, the sorting switches to Insertion Sort instead of continuing with the divide-and-conquer method.
This threshold can be adjusted to find the optimal performance by changing the recursion depth.

- Merge Sort: The subarray is directly sorted using Insertion Sort if its size is less than or equal to the threshold.
- Quick Sort: Similarly, Quick Sort switches to Insertion Sort when the subarray size is below the threshold.
  The threshold values tested in the code are: 2, 5, 10, 20, 50.

2. Non-Recursive Method(s) Chosen:

The non-recursive method chosen in the code is Insertion Sort. Insertion Sort is an efficient algorithm for small arrays because it has a low constant overhead and performs well when the array is already nearly sorted or of small size.

- Insertion Sort is triggered when the size of the subarray becomes less than or equal to a specified threshold value. It essentially finishes the sorting for very small subarrays, which avoids the overhead of recursive calls for small inputs.

3. Different Input Sizes Chosen to Test the Program:

The code tests the sorting algorithms for a range of input sizes:

- 1000
- 10000
- 100000
- 1000000

These sizes are chosen to test how the performance of the algorithm scales as the input size increases. The larger the input, the more significant the impact of the chosen threshold and the method used (Insertion Sort vs. Merge/Quick Sort).

4. Experiment Results of the Different Combinations of the Above 3 Things:

```
Testing with input size: 1000
Threshold: 2 | Modified Recursive Merge Sort: Time taken by function with threshold 2: 394910 nanoseconds
Threshold: 2 | Modified Recursive Quick Sort: Time taken by function with threshold 2: 167985 nanoseconds
Threshold: 5 | Modified Recursive Merge Sort: Time taken by function with threshold 5: 261895 nanoseconds
Threshold: 5 | Modified Recursive Quick Sort: Time taken by function with threshold 5: 153948 nanoseconds
Threshold: 10 | Modified Recursive Merge Sort: Time taken by function with threshold 10: 238382 nanoseconds
Threshold: 10 | Modified Recursive Quick Sort: Time taken by function with threshold 10: 136974 nanoseconds
Threshold: 20 | Modified Recursive Merge Sort: Time taken by function with threshold 20: 204477 nanoseconds
Threshold: 20 | Modified Recursive Quick Sort: Time taken by function with threshold 20: 150974 nanoseconds
Threshold: 50 | Modified Recursive Merge Sort: Time taken by function with threshold 50: 189544 nanoseconds
Threshold: 50 | Modified Recursive Quick Sort: Time taken by function with threshold 50: 168253 nanoseconds
--------------------------------
Testing with input size: 10000
Threshold: 2 | Modified Recursive Merge Sort: Time taken by function with threshold 2: 4757266 nanoseconds
Threshold: 2 | Modified Recursive Quick Sort: Time taken by function with threshold 2: 2313468 nanoseconds
Threshold: 5 | Modified Recursive Merge Sort: Time taken by function with threshold 5: 3779851 nanoseconds
Threshold: 5 | Modified Recursive Quick Sort: Time taken by function with threshold 5: 2151176 nanoseconds
Threshold: 10 | Modified Recursive Merge Sort: Time taken by function with threshold 10: 3016304 nanoseconds
Threshold: 10 | Modified Recursive Quick Sort: Time taken by function with threshold 10: 2174286 nanoseconds
Threshold: 20 | Modified Recursive Merge Sort: Time taken by function with threshold 20: 2698264 nanoseconds
Threshold: 20 | Modified Recursive Quick Sort: Time taken by function with threshold 20: 2277611 nanoseconds
Threshold: 50 | Modified Recursive Merge Sort: Time taken by function with threshold 50: 2877779 nanoseconds
Threshold: 50 | Modified Recursive Quick Sort: Time taken by function with threshold 50: 2209476 nanoseconds
--------------------------------
Testing with input size: 100000
Threshold: 2 | Modified Recursive Merge Sort: Time taken by function with threshold 2: 52893416 nanoseconds
Threshold: 2 | Modified Recursive Quick Sort: Time taken by function with threshold 2: 29936840 nanoseconds
Threshold: 5 | Modified Recursive Merge Sort: Time taken by function with threshold 5: 44376812 nanoseconds
Threshold: 5 | Modified Recursive Quick Sort: Time taken by function with threshold 5: 27868197 nanoseconds
Threshold: 10 | Modified Recursive Merge Sort: Time taken by function with threshold 10: 36152319 nanoseconds
Threshold: 10 | Modified Recursive Quick Sort: Time taken by function with threshold 10: 24447370 nanoseconds
Threshold: 20 | Modified Recursive Merge Sort: Time taken by function with threshold 20: 33728401 nanoseconds
Threshold: 20 | Modified Recursive Quick Sort: Time taken by function with threshold 20: 23370353 nanoseconds
Threshold: 50 | Modified Recursive Merge Sort: Time taken by function with threshold 50: 33445366 nanoseconds
Threshold: 50 | Modified Recursive Quick Sort: Time taken by function with threshold 50: 24243573 nanoseconds
--------------------------------
Testing with input size: 1000000
Threshold: 2 | Modified Recursive Merge Sort: Time taken by function with threshold 2: 550379345 nanoseconds
Threshold: 2 | Modified Recursive Quick Sort: Time taken by function with threshold 2: 891392398 nanoseconds
Threshold: 5 | Modified Recursive Merge Sort: Time taken by function with threshold 5: 490855236 nanoseconds
Threshold: 5 | Modified Recursive Quick Sort: Time taken by function with threshold 5: 929175497 nanoseconds
Threshold: 10 | Modified Recursive Merge Sort: Time taken by function with threshold 10: 401192732 nanoseconds
Threshold: 10 | Modified Recursive Quick Sort: Time taken by function with threshold 10: 886825354 nanoseconds
Threshold: 20 | Modified Recursive Merge Sort: Time taken by function with threshold 20: 380323326 nanoseconds
Threshold: 20 | Modified Recursive Quick Sort: Time taken by function with threshold 20: 873807363 nanoseconds
Threshold: 50 | Modified Recursive Merge Sort: Time taken by function with threshold 50: 377962871 nanoseconds
Threshold: 50 | Modified Recursive Quick Sort: Time taken by function with threshold 50: 720350944 nanoseconds
--------------------------------
```