```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <omp.h>

using namespace std;

// Parallel Bubble Sort function
void parallelBubbleSort(int *array, int n) {
    int i, j;

    #pragma omp parallel for private(i, j) shared(array)
    for (i = 0; i < n-1; i++) {
        for (j = 0; j < n-i-1; j++) {
            if (array[j] > array[j+1]) {
                // Swap elements
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
            }
        }
    }
}

// Parallel Merge Sort function
void merge(int *array, int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;

    // Create temp arrays
    int *L = new int[n1];
    int *R = new int[n2];

    // Copy data to temp arrays L[] and R[]
    for (i = 0; i < n1; i++)
        L[i] = array[l + i];
    for (j = 0; j < n2; j++)
        R[j] = array[m + 1+ j];

    // Merge the temp arrays back into array[l..r]
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            array[k] = L[i];
            i++;
        } else {
            array[k] = R[j];
            j++;
        }
        k++;
    }
```

```
    // Copy the remaining elements of L[], if there are any
    while (i < n1) {
        array[k] = L[i];
        i++;
        k++;
    }

    // Copy the remaining elements of R[], if there are any
    while (j < n2) {
        array[k] = R[j];
        j++;
        k++;
    }

    delete [] L;
    delete [] R;
}

void parallelMergeSort(int *array, int l, int r) {
    if (l < r) {
        int m = l+(r-l)/2;

        #pragma omp parallel sections
        {
            #pragma omp section
            parallelMergeSort(array, l, m);

            #pragma omp section
            parallelMergeSort(array, m+1, r);
        }

        merge(array, l, m, r);
    }
}

int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;

int *array = new int[n];
srand(time(0));
for (int i = 0; i < n; i++) {
    array[i] = rand() % 100;
}

cout << "Original Array: ";
for (int i = 0; i < n; i++) {
    cout << array[i] << " ";
}
cout << endl;

int choice;
```

```
cout << "Enter 1 for Parallel Bubble Sort or 2 for Parallel Merge Sort:
";
cin >> choice;

if (choice == 1) {
    parallelBubbleSort(array, n);
} else if (choice == 2) {
    parallelMergeSort(array, 0, n-1);
} else {
    cout << "Invalid choice. Exiting program." << endl;
    return 0;
}

cout << "Sorted Array: ";
for (int i = 0; i < n; i++) {
    cout << array[i] << " ";
}
cout << endl;

delete [] array;

return 0;
}
```

//

Test Case 1:
Input:
Enter the size of the array: 5
Enter 1 for Parallel Bubble Sort or 2 for Parallel Merge Sort: 1

Output:
Original Array: 68 67 69 73 29
Sorted Array: 29 67 68 69 73

Test Case 2:
Input:
Enter the size of the array: 10
Enter 1 for Parallel Bubble Sort or 2 for Parallel Merge Sort: 2

Output:
Original Array: 97 73 76 77 35 77 29 44 50 77
Sorted Array: 29 35 44 50 73 76 77 77 77 97

Test Case 3:
Input:
Enter the size of the array: 7
Enter 1 for Parallel Bubble Sort or 2 for Parallel Merge Sort: 3

Output:

```
Invalid choice. Exiting program.
```