Solution –Assignment-4

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

// Function to display an array
void displayArray(const vector<int>& arr) {
    for (int num : arr) {
        cout << num << " ";
    }
    cout << endl;
}

// Bubble Sort
void bubbleSort(vector<int> arr) {
    auto start = high_resolution_clock::now();
    int n = arr.size();
    for (int i = 0; i < n - 1; ++i) {
        for (int j = 0; j < n - i - 1; ++j) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
    auto end = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(end - start);
    cout << "Bubble Sort Time: " << duration.count() << " microseconds\n";
}

// Selection Sort
void selectionSort(vector<int> arr) {
    auto start = high_resolution_clock::now();
    int n = arr.size();
    for (int i = 0; i < n - 1; ++i) {
        int minIdx = i;
        for (int j = i + 1; j < n; ++j) {
            if (arr[j] < arr[minIdx]) {
                minIdx = j;
            }
        }
        swap(arr[minIdx], arr[i]);
    }
    auto end = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(end - start);
    cout << "Selection Sort Time: " << duration.count() << " microseconds\n";
}

// Insertion Sort
void insertionSort(vector<int> arr) {
    auto start = high_resolution_clock::now();
    int n = arr.size();
    for (int i = 1; i < n; ++i) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            --j;
        }
        arr[j + 1] = key;
    }
    auto end = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(end - start);
    cout << "Insertion Sort Time: " << duration.count() << " microseconds\n";
}

// Merge Sort
void merge(vector<int>& arr, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    vector<int> L(n1), R(n2);
    for (int i = 0; i < n1; ++i) L[i] = arr[left + i];
    for (int i = 0; i < n2; ++i) R[i] = arr[mid + 1 + i];

    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 mergeSortHelper(vector<int>& arr, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSortHelper(arr, left, mid);
        mergeSortHelper(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

void mergeSort(vector<int> arr) {
    auto start = high_resolution_clock::now();
    mergeSortHelper(arr, 0, arr.size() - 1);
    auto end = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(end - start);
    cout << "Merge Sort Time: " << duration.count() << " microseconds\n";
}

int main() {
    vector<int> data = {64, 34, 25, 12, 22, 11, 90};

    cout << "Original Array: ";
    displayArray(data);

    bubbleSort(data);
    selectionSort(data);
    insertionSort(data);
    mergeSort(data);

    return 0;
}
```