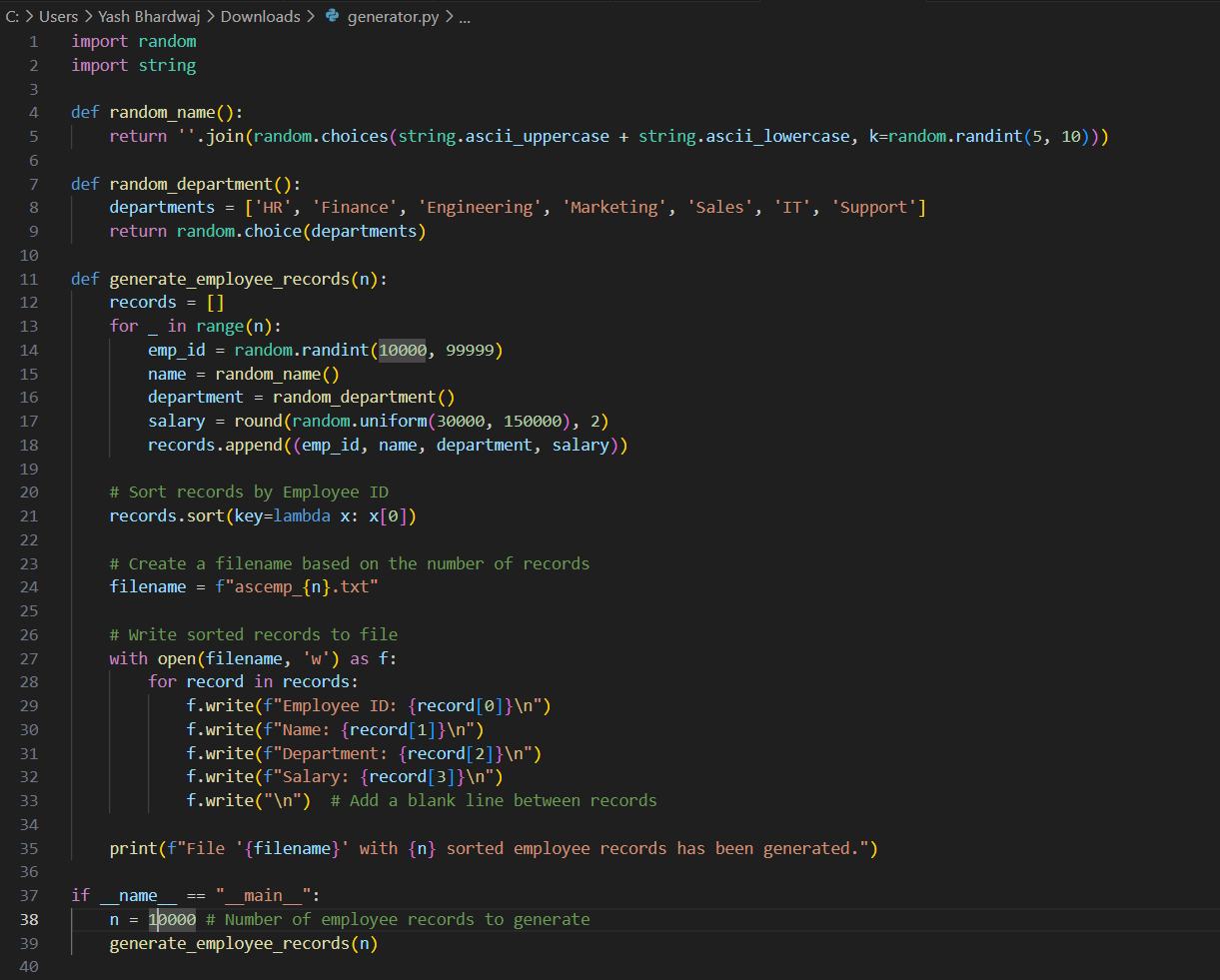
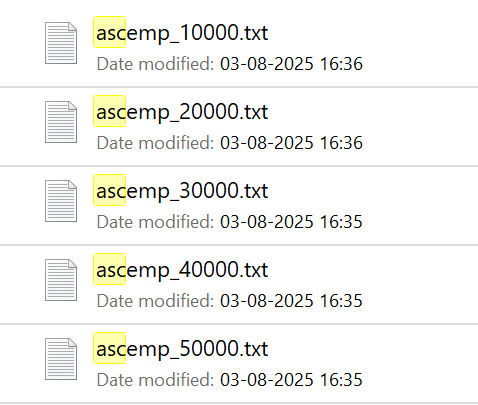
**EMPLOYEE RECORDS**

YASH BHARDWAJ 23BRS1236

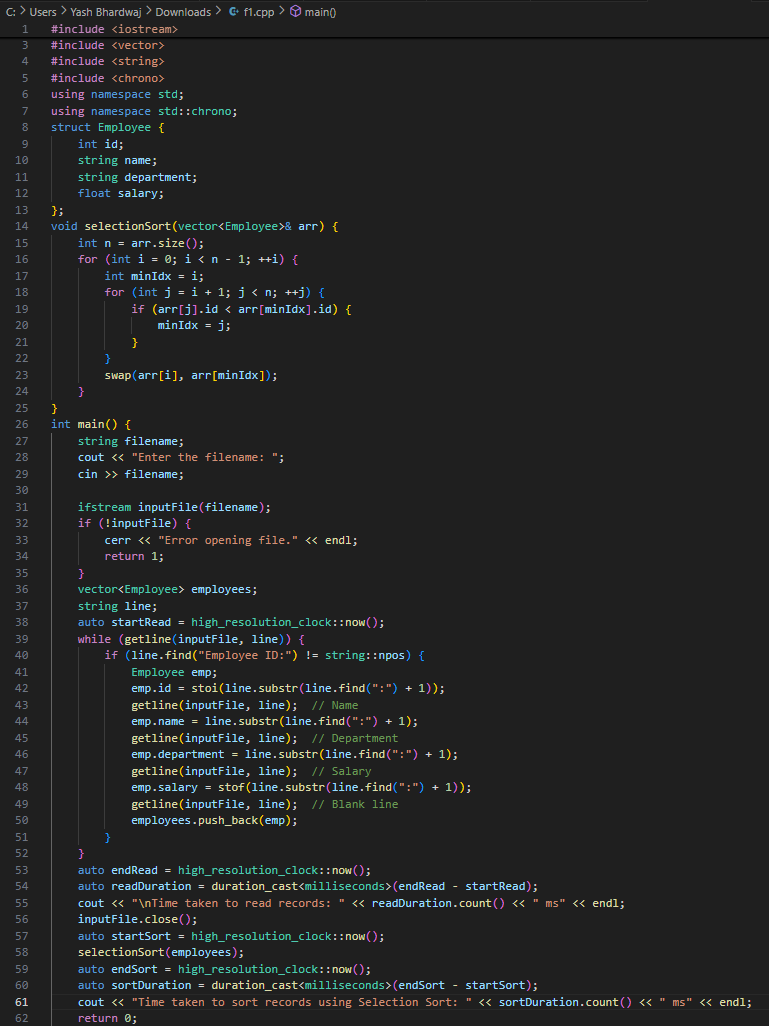
**FILE GENERATION**

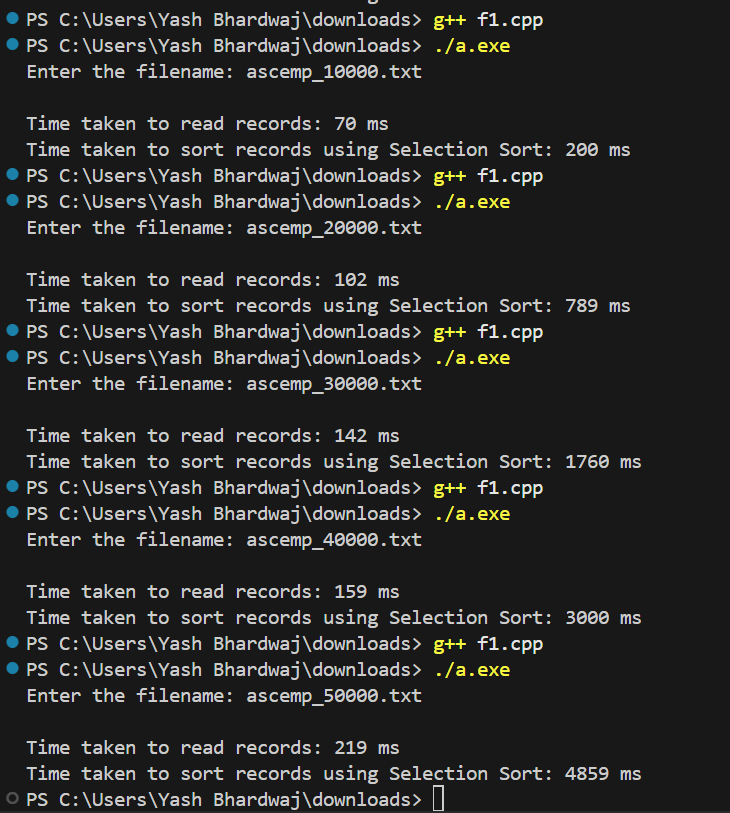




**ASCENDING ORDER**

SELECTION SORT





INSERTION SORT

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <chrono>

using namespace std;

using namespace std::chrono;

struct Employee {

    int id;

    string name;

    string department;

    float salary;

};

// Insertion Sort on Employee ID

void insertionSort(vector<Employee>& arr) {

    int n = arr.size();

    for (int i = 1; i < n; ++i) {

        Employee key = arr[i];

        int j = i - 1;

        while (j >= 0 && arr[j].id > key.id) {

            arr[j + 1] = arr[j];

            --j;

        }

        arr[j + 1] = key;

    }

}

int main() {

    string filename;

    cout << "Enter the filename: ";

    cin >> filename;

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Error opening file." << endl;

        return 1;

    }

    vector<Employee> employees;

    string line;

    // Time reading

    auto startRead = high\_resolution\_clock::now();

    while (getline(inputFile, line)) {

        if (line.find("Employee ID:") != string::npos) {

            Employee emp;

            emp.id = stoi(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Name

            emp.name = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Department

            emp.department = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Salary

            emp.salary = stof(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Blank line

            employees.push\_back(emp);

        }

    }

    auto endRead = high\_resolution\_clock::now();

    auto readDuration = duration\_cast<milliseconds>(endRead - startRead);

    cout << "\nTime taken to read records: " << readDuration.count() << " ms" << endl;

    inputFile.close();

    auto startSort = high\_resolution\_clock::now();

    insertionSort(employees);

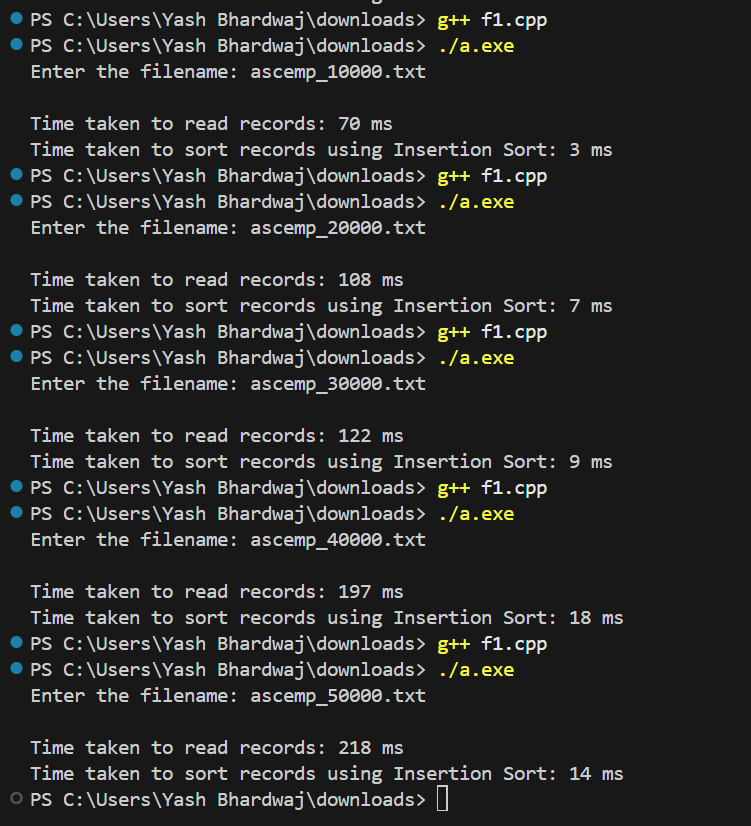
    auto endSort = high\_resolution\_clock::now();

    auto sortDuration = duration\_cast<milliseconds>(endSort - startSort);

    cout << "Time taken to sort records using Insertion Sort: " << sortDuration.count() << " ms" << endl;

    return 0;

}



MERGE SORT

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <chrono>

using namespace std;

using namespace std::chrono;

struct Employee {

    int id;

    string name;

    string department;

    float salary;

};

// Merge function

void merge(vector<Employee>& arr, int left, int mid, int right) {

    int n1 = mid - left + 1;

    int n2 = right - mid;

    vector<Employee> L(n1);

    vector<Employee> 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].id <= R[j].id) {

            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;

    }

}

// Merge Sort

void mergeSort(vector<Employee>& arr, int left, int right) {

    if (left < right) {

        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);

        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);

    }

}

int main() {

    string filename;

    cout << "Enter the filename: ";

    cin >> filename;

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Error opening file." << endl;

        return 1;

    }

    vector<Employee> employees;

    string line;

    // Time reading

    auto startRead = high\_resolution\_clock::now();

    while (getline(inputFile, line)) {

        if (line.find("Employee ID:") != string::npos) {

            Employee emp;

            emp.id = stoi(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Name

            emp.name = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Department

            emp.department = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Salary

            emp.salary = stof(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Blank line

            employees.push\_back(emp);

        }

    }

    auto endRead = high\_resolution\_clock::now();

    auto readDuration = duration\_cast<milliseconds>(endRead - startRead);

    cout << "\nTime taken to read records: " << readDuration.count() << " ms" << endl;

    inputFile.close();

    auto startSort = high\_resolution\_clock::now();

    mergeSort(employees, 0, employees.size() - 1);

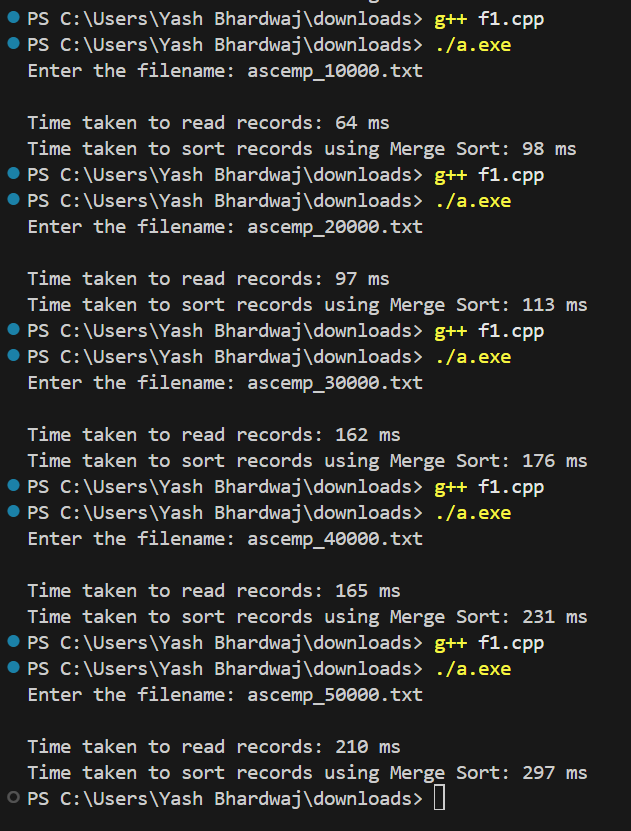
    auto endSort = high\_resolution\_clock::now();

    auto sortDuration = duration\_cast<milliseconds>(endSort - startSort);

    cout << "Time taken to sort records using Merge Sort: " << sortDuration.count() << " ms" << endl;

    return 0;

}



QUICK SORT

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <chrono>

using namespace std;

using namespace std::chrono;

struct Employee {

    int id;

    string name;

    string department;

    float salary;

};

// Quick Sort partition function

int partition(vector<Employee>& arr, int low, int high) {

    int pivot = arr[high].id;

    int i = low - 1;

    for (int j = low; j < high; ++j) {

        if (arr[j].id <= pivot) {

            ++i;

            swap(arr[i], arr[j]);

        }

    }

    swap(arr[i + 1], arr[high]);

    return i + 1;

}

// Quick Sort

void quickSort(vector<Employee>& arr, int low, int high) {

    if (low < high) {

        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);

        quickSort(arr, pi + 1, high);

    }

}

int main() {

    string filename;

    cout << "Enter the filename: ";

    cin >> filename;

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Error opening file." << endl;

        return 1;

    }

    vector<Employee> employees;

    string line;

    // Time reading

    auto startRead = high\_resolution\_clock::now();

    while (getline(inputFile, line)) {

        if (line.find("Employee ID:") != string::npos) {

            Employee emp;

            emp.id = stoi(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Name

            emp.name = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Department

            emp.department = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Salary

            emp.salary = stof(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Blank line

            employees.push\_back(emp);

        }

    }

    auto endRead = high\_resolution\_clock::now();

    auto readDuration = duration\_cast<milliseconds>(endRead - startRead);

    cout << "\nTime taken to read records: " << readDuration.count() << " ms" << endl;

    inputFile.close();

    auto startSort = high\_resolution\_clock::now();

    quickSort(employees, 0, employees.size() - 1);

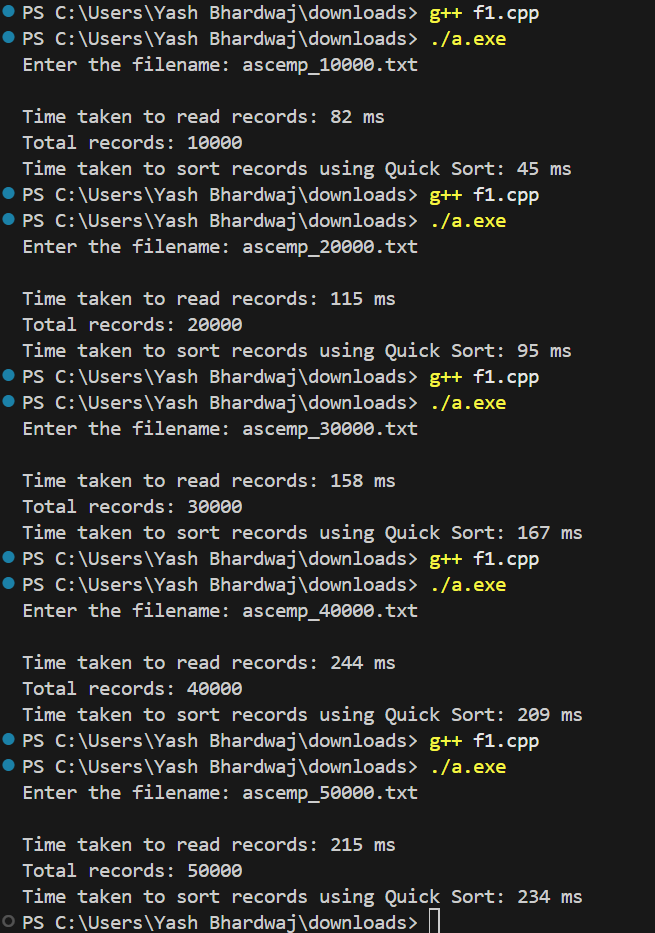
    auto endSort = high\_resolution\_clock::now();

    auto sortDuration = duration\_cast<milliseconds>(endSort - startSort);

    cout << "Time taken to sort records using Quick Sort: " << sortDuration.count() << " ms" << endl;

    return 0;

}



**DESCENDING ORDER**

import random

import string

def random\_name():

    return ''.join(random.choices(string.ascii\_uppercase + string.ascii\_lowercase, k=random.randint(5, 10)))

def random\_department():

    departments = ['HR', 'Finance', 'Engineering', 'Marketing', 'Sales', 'IT', 'Support']

    return random.choice(departments)

def generate\_descending\_employee\_records(n):

    records = []

    emp\_ids = list(range(10000, 10000 + n))  # Generate sequential Employee IDs

    random.shuffle(emp\_ids)  # Shuffle to randomize order

    # Create records with descending Employee IDs

    emp\_ids.sort(reverse=True)  # Sort in descending order

    for emp\_id in emp\_ids:

        name = random\_name()

        department = random\_department()

        salary = round(random.uniform(30000, 150000), 2)

        records.append((emp\_id, name, department, salary))

    # Create a filename based on the number of records

    filename = f"descemp\_{n}.txt"

    # Write records to file

    with open(filename, 'w') as f:

        for record in records:

            f.write(f"Employee ID: {record[0]}\n")

            f.write(f"Name: {record[1]}\n")

            f.write(f"Department: {record[2]}\n")

            f.write(f"Salary: {record[3]}\n")

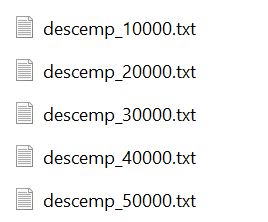
            f.write("\n")  # Add a blank line between records

    print(f"File '{filename}' with {n} employee records in descending order of Employee IDs has been generated.")

if \_\_name\_\_ == "\_\_main\_\_":

    n = 50000  # Number of employee records to generate

    generate\_descending\_employee\_records(n)

****

SELECTION SORT

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <chrono>

using namespace std;

using namespace std::chrono;

struct Employee {

    int id;

    string name;

    string department;

    float salary;

};

// Selection Sort

void selectionSort(vector<Employee>& arr) {

    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].id < arr[minIdx].id) {

                minIdx = j;

            }

        }

        swap(arr[i], arr[minIdx]);

    }

}

int main() {

    string filename;

    cout << "Enter the filename: ";

    cin >> filename;

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Error opening file." << endl;

        return 1;

    }

    vector<Employee> employees;

    string line;

    // Time reading

    auto startRead = high\_resolution\_clock::now();

    while (getline(inputFile, line)) {

        if (line.find("Employee ID:") != string::npos) {

            Employee emp;

            emp.id = stoi(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Name

            emp.name = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Department

            emp.department = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Salary

            emp.salary = stof(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Blank line

            employees.push\_back(emp);

        }

    }

    auto endRead = high\_resolution\_clock::now();

    auto readDuration = duration\_cast<milliseconds>(endRead - startRead);

    cout << "\nTime taken to read records: " << readDuration.count() << " ms" << endl;

    inputFile.close();

    auto startSort = high\_resolution\_clock::now();

    selectionSort(employees);

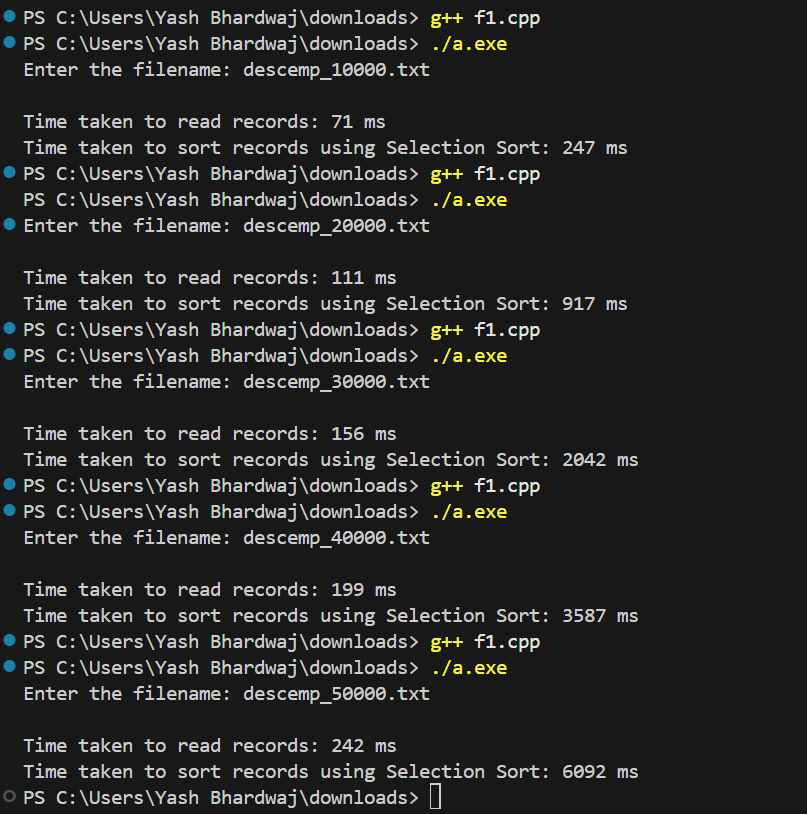
    auto endSort = high\_resolution\_clock::now();

    auto sortDuration = duration\_cast<milliseconds>(endSort - startSort);

    cout << "Time taken to sort records using Selection Sort: " << sortDuration.count() << " ms" << endl;

    return 0;

}



INSERTION SORT

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <chrono>

using namespace std;

using namespace std::chrono;

struct Employee {

    int id;

    string name;

    string department;

    float salary;

};

// Insertion Sort

void insertionSort(vector<Employee>& arr) {

    int n = arr.size();

    for (int i = 1; i < n; ++i) {

        Employee key = arr[i];

        int j = i - 1;

        while (j >= 0 && arr[j].id > key.id) {

            arr[j + 1] = arr[j];

            --j;

        }

        arr[j + 1] = key;

    }

}

int main() {

    string filename;

    cout << "Enter the filename: ";

    cin >> filename;

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Error opening file." << endl;

        return 1;

    }

    vector<Employee> employees;

    string line;

    // Time reading

    auto startRead = high\_resolution\_clock::now();

    while (getline(inputFile, line)) {

        if (line.find("Employee ID:") != string::npos) {

            Employee emp;

            emp.id = stoi(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Name

            emp.name = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Department

            emp.department = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Salary

            emp.salary = stof(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Blank line

            employees.push\_back(emp);

        }

    }

    auto endRead = high\_resolution\_clock::now();

    auto readDuration = duration\_cast<milliseconds>(endRead - startRead);

    cout << "\nTime taken to read records: " << readDuration.count() << " ms" << endl;

    inputFile.close();

    auto startSort = high\_resolution\_clock::now();

    insertionSort(employees);

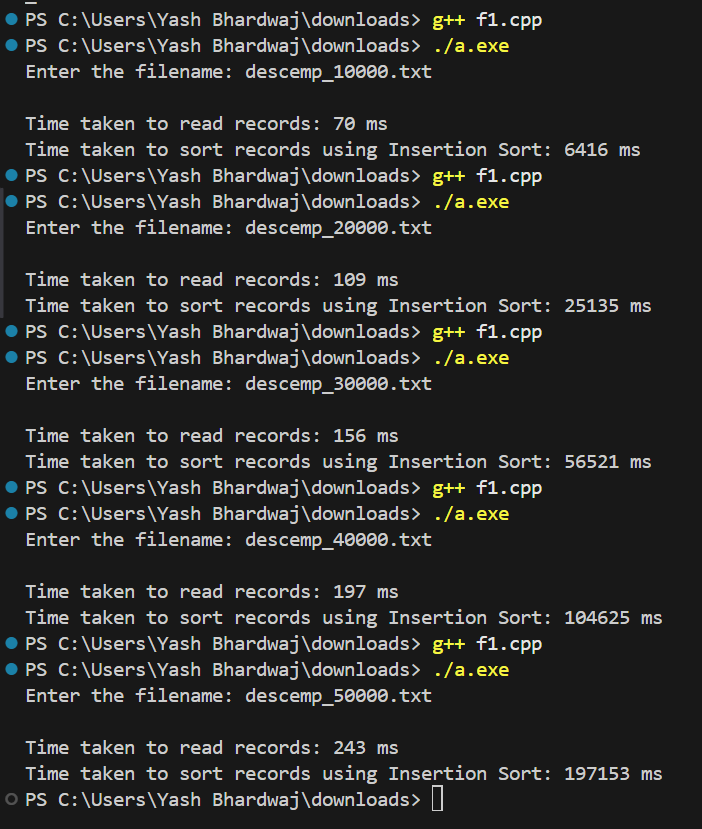
    auto endSort = high\_resolution\_clock::now();

    auto sortDuration = duration\_cast<milliseconds>(endSort - startSort);

    cout << "Time taken to sort records using Insertion Sort: " << sortDuration.count() << " ms" << endl;

    return 0;

}



MERGE SORT

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <chrono>

using namespace std;

using namespace std::chrono;

struct Employee {

    int id;

    string name;

    string department;

    float salary;

};

// Merge Sort

void merge(vector<Employee>& arr, int left, int mid, int right) {

    int n1 = mid - left + 1;

    int n2 = right - mid;

    vector<Employee> L(n1);

    vector<Employee> 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].id <= R[j].id) {

            arr[k++] = L[i++];

        } else {

            arr[k++] = R[j++];

        }

    }

    while (i < n1)

        arr[k++] = L[i++];

    while (j < n2)

        arr[k++] = R[j++];

}

void mergeSort(vector<Employee>& arr, int left, int right) {

    if (left < right) {

        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);

        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);

    }

}

int main() {

    string filename;

    cout << "Enter the filename: ";

    cin >> filename;

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Error opening file." << endl;

        return 1;

    }

    vector<Employee> employees;

    string line;

    // Time reading

    auto startRead = high\_resolution\_clock::now();

    while (getline(inputFile, line)) {

        if (line.find("Employee ID:") != string::npos) {

            Employee emp;

            emp.id = stoi(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Name

            emp.name = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Department

            emp.department = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Salary

            emp.salary = stof(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Blank line

            employees.push\_back(emp);

        }

    }

    auto endRead = high\_resolution\_clock::now();

    auto readDuration = duration\_cast<milliseconds>(endRead - startRead);

    cout << "\nTime taken to read records: " << readDuration.count() << " ms" << endl;

    inputFile.close();

    auto startSort = high\_resolution\_clock::now();

    mergeSort(employees, 0, employees.size() - 1);

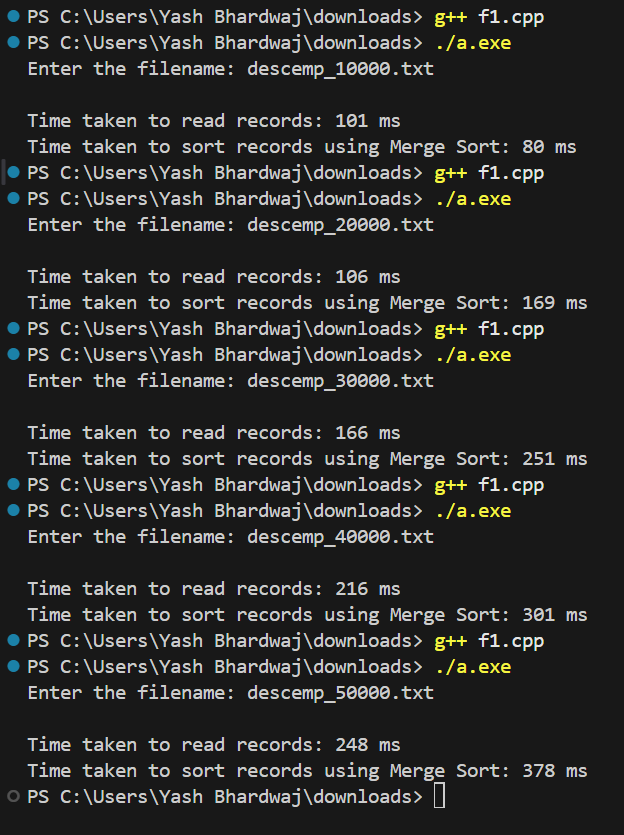
    auto endSort = high\_resolution\_clock::now();

    auto sortDuration = duration\_cast<milliseconds>(endSort - startSort);

    cout << "Time taken to sort records using Merge Sort: " << sortDuration.count() << " ms" << endl;

    return 0;

}



QUICK SORT

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <chrono>

using namespace std;

using namespace std::chrono;

struct Employee {

    int id;

    string name;

    string department;

    float salary;

};

int medianOfThree(vector<Employee>& arr, int low, int high) {

    int mid = low + (high - low) / 2;

    int a = arr[low].id;

    int b = arr[mid].id;

    int c = arr[high].id;

    if ((a < b && b < c) || (c < b && b < a)) return mid;

    else if ((b < a && a < c) || (c < a && a < b)) return low;

    else return high;

}

// Optimized Quick Sort with Median-of-Three

int partition(vector<Employee>& arr, int low, int high) {

    int pivotIndex = medianOfThree(arr, low, high);

    swap(arr[pivotIndex], arr[high]);

    int pivot = arr[high].id;

    int i = low - 1;

    for (int j = low; j < high; ++j) {

        if (arr[j].id <= pivot) {

            ++i;

            swap(arr[i], arr[j]);

        }

    }

    swap(arr[i + 1], arr[high]);

    return i + 1;

}

void quickSort(vector<Employee>& arr, int low, int high) {

    if (low < high) {

        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);

        quickSort(arr, pi + 1, high);

    }

}

int main() {

    string filename;

    cout << "Enter the filename: ";

    cin >> filename;

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Error opening file." << endl;

        return 1;

    }

    vector<Employee> employees;

    string line;

    // Time reading

    auto startRead = high\_resolution\_clock::now();

    while (getline(inputFile, line)) {

        if (line.find("Employee ID:") != string::npos) {

            Employee emp;

            emp.id = stoi(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Name

            emp.name = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Department

            emp.department = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Salary

            emp.salary = stof(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Blank line

            employees.push\_back(emp);

        }

    }

    auto endRead = high\_resolution\_clock::now();

    auto readDuration = duration\_cast<milliseconds>(endRead - startRead);

    cout << "\nTime taken to read records: " << readDuration.count() << " ms" << endl;

    inputFile.close();

    cout << "Total records: " << employees.size() << endl;

    auto startSort = high\_resolution\_clock::now();

    quickSort(employees, 0, employees.size() - 1);

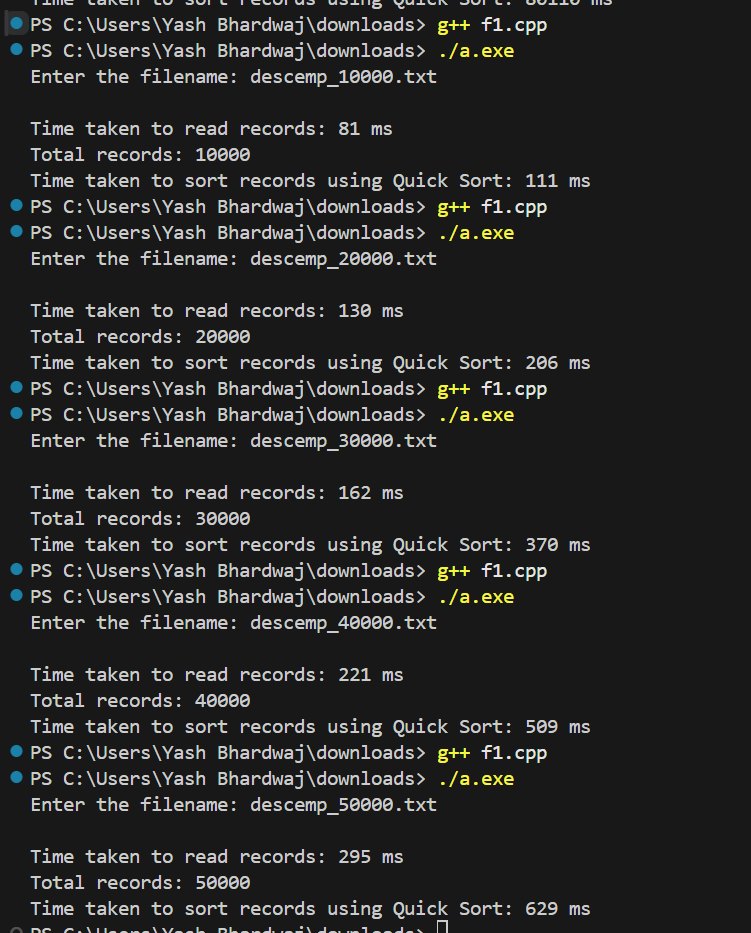
    auto endSort = high\_resolution\_clock::now();

    auto sortDuration = duration\_cast<milliseconds>(endSort - startSort);

    cout << "Time taken to sort records using Quick Sort: " << sortDuration.count() << " ms" << endl;

    return 0;

}



**RANDOM ORDER**

import random

import string

def random\_name():

    return ''.join(random.choices(string.ascii\_uppercase + string.ascii\_lowercase, k=random.randint(5, 10)))

def random\_department():

    departments = ['HR', 'Finance', 'Engineering', 'Marketing', 'Sales', 'IT', 'Support']

    return random.choice(departments)

def generate\_random\_employee\_records(n):

    records = []

    emp\_ids = random.sample(range(10000, 100000), n)  # Generate unique random Employee IDs

    for emp\_id in emp\_ids:

        name = random\_name()

        department = random\_department()

        salary = round(random.uniform(30000, 150000), 2)

        records.append((emp\_id, name, department, salary))

    # Create a filename based on the number of records

    filename = f"randemp\_{n}.txt"

    # Write records to file

    with open(filename, 'w') as f:

        for record in records:

            f.write(f"Employee ID: {record[0]}\n")

            f.write(f"Name: {record[1]}\n")

            f.write(f"Department: {record[2]}\n")

            f.write(f"Salary: {record[3]}\n")

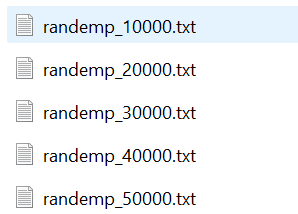
            f.write("\n")  # Add a blank line between records

    print(f"File '{filename}' with {n} randomly arranged employee records has been generated.")

if \_\_name\_\_ == "\_\_main\_\_":

    n = 10000  # Number of employee records to generate

    generate\_random\_employee\_records(n)



SELECTION SORT

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <chrono>

using namespace std;

using namespace std::chrono;

struct Employee {

    int id;

    string name;

    string department;

    float salary;

};

// Selection Sort

void selectionSort(vector<Employee>& arr) {

    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].id < arr[minIdx].id) {

                minIdx = j;

            }

        }

        swap(arr[i], arr[minIdx]);

    }

}

int main() {

    string filename;

    cout << "Enter the filename: ";

    cin >> filename;

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Error opening file." << endl;

        return 1;

    }

    vector<Employee> employees;

    string line;

    // Time reading

    auto startRead = high\_resolution\_clock::now();

    while (getline(inputFile, line)) {

        if (line.find("Employee ID:") != string::npos) {

            Employee emp;

            emp.id = stoi(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Name

            emp.name = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Department

            emp.department = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Salary

            emp.salary = stof(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Blank line

            employees.push\_back(emp);

        }

    }

    auto endRead = high\_resolution\_clock::now();

    auto readDuration = duration\_cast<milliseconds>(endRead - startRead);

    cout << "\nTime taken to read records: " << readDuration.count() << " ms" << endl;

    inputFile.close();

    auto startSort = high\_resolution\_clock::now();

    selectionSort(employees);

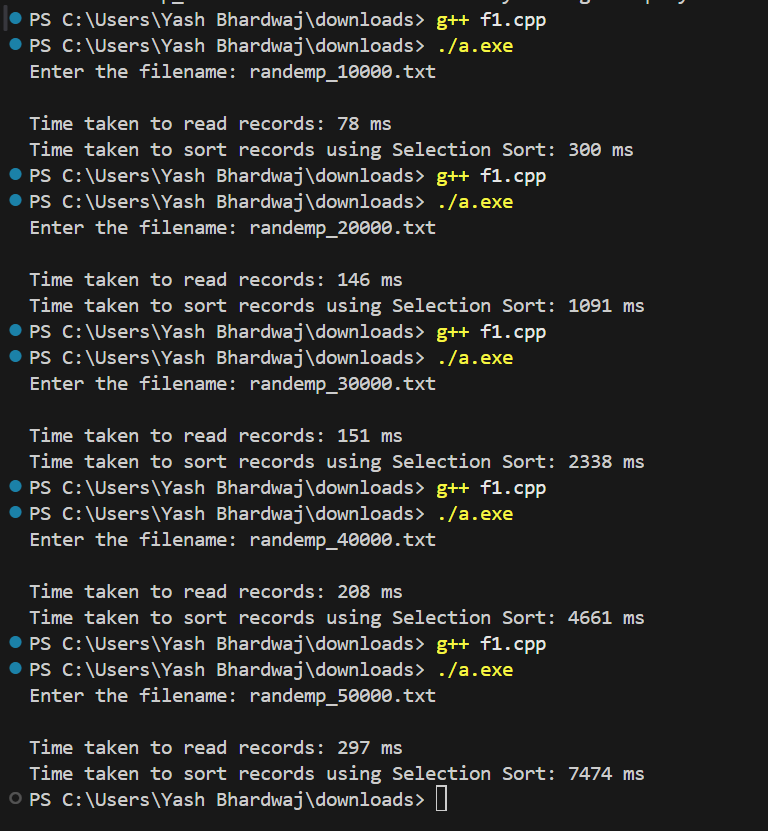
    auto endSort = high\_resolution\_clock::now();

    auto sortDuration = duration\_cast<milliseconds>(endSort - startSort);

    cout << "Time taken to sort records using Selection Sort: " << sortDuration.count() << " ms" << endl;

    return 0;

}



INSERTION SORT

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <chrono>

using namespace std;

using namespace std::chrono;

struct Employee {

    int id;

    string name;

    string department;

    float salary;

};

// Insertion Sort

void insertionSort(vector<Employee>& arr) {

    int n = arr.size();

    for (int i = 1; i < n; ++i) {

        Employee key = arr[i];

        int j = i - 1;

        while (j >= 0 && arr[j].id > key.id) {

            arr[j + 1] = arr[j];

            --j;

        }

        arr[j + 1] = key;

    }

}

int main() {

    string filename;

    cout << "Enter the filename: ";

    cin >> filename;

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Error opening file." << endl;

        return 1;

    }

    vector<Employee> employees;

    string line;

    // Time reading

    auto startRead = high\_resolution\_clock::now();

    while (getline(inputFile, line)) {

        if (line.find("Employee ID:") != string::npos) {

            Employee emp;

            emp.id = stoi(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Name

            emp.name = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Department

            emp.department = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Salary

            emp.salary = stof(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Blank line

            employees.push\_back(emp);

        }

    }

    auto endRead = high\_resolution\_clock::now();

    auto readDuration = duration\_cast<milliseconds>(endRead - startRead);

    cout << "\nTime taken to read records: " << readDuration.count() << " ms" << endl;

    inputFile.close();

    auto startSort = high\_resolution\_clock::now();

    insertionSort(employees);

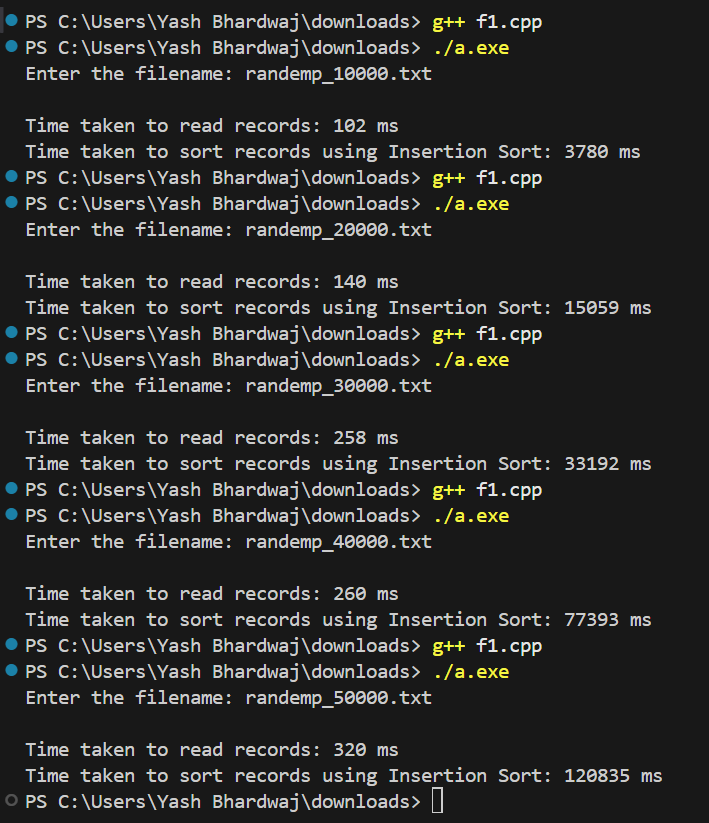
    auto endSort = high\_resolution\_clock::now();

    auto sortDuration = duration\_cast<milliseconds>(endSort - startSort);

    cout << "Time taken to sort records using Insertion Sort: " << sortDuration.count() << " ms" << endl;

    return 0;

}



MERGE SORT

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <chrono>

using namespace std;

using namespace std::chrono;

struct Employee {

    int id;

    string name;

    string department;

    float salary;

};

// Merge Sort

void merge(vector<Employee>& arr, int left, int mid, int right) {

    int n1 = mid - left + 1;

    int n2 = right - mid;

    vector<Employee> L(n1);

    vector<Employee> 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].id <= R[j].id) {

            arr[k++] = L[i++];

        } else {

            arr[k++] = R[j++];

        }

    }

    while (i < n1)

        arr[k++] = L[i++];

    while (j < n2)

        arr[k++] = R[j++];

}

void mergeSort(vector<Employee>& arr, int left, int right) {

    if (left < right) {

        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);

        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);

    }

}

int main() {

    string filename;

    cout << "Enter the filename: ";

    cin >> filename;

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Error opening file." << endl;

        return 1;

    }

    vector<Employee> employees;

    string line;

    // Time reading

    auto startRead = high\_resolution\_clock::now();

    while (getline(inputFile, line)) {

        if (line.find("Employee ID:") != string::npos) {

            Employee emp;

            emp.id = stoi(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Name

            emp.name = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Department

            emp.department = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Salary

            emp.salary = stof(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Blank line

            employees.push\_back(emp);

        }

    }

    auto endRead = high\_resolution\_clock::now();

    auto readDuration = duration\_cast<milliseconds>(endRead - startRead);

    cout << "\nTime taken to read records: " << readDuration.count() << " ms" << endl;

    inputFile.close();

    auto startSort = high\_resolution\_clock::now();

    mergeSort(employees, 0, employees.size() - 1);

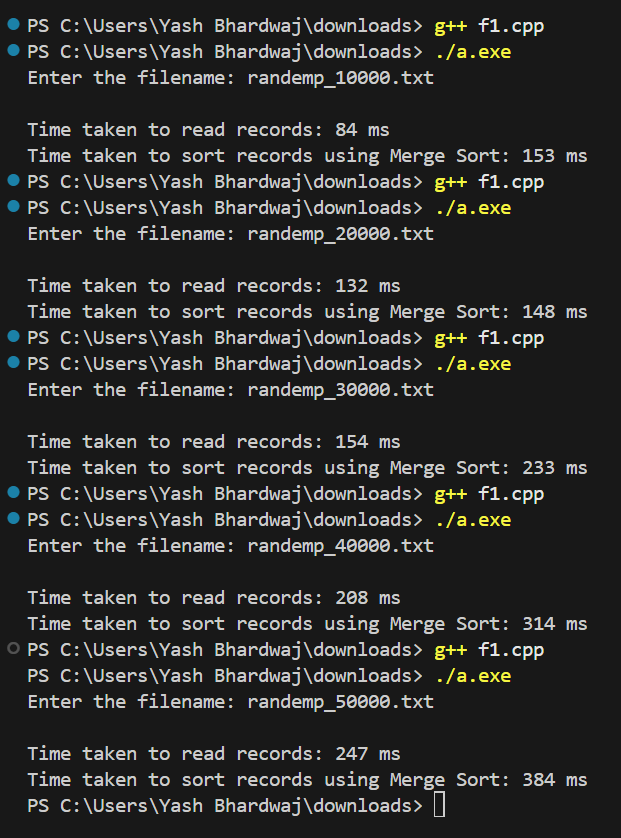
    auto endSort = high\_resolution\_clock::now();

    auto sortDuration = duration\_cast<milliseconds>(endSort - startSort);

    cout << "Time taken to sort records using Merge Sort: " << sortDuration.count() << " ms" << endl;

    return 0;

}



QUICK SORT

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <chrono>

using namespace std;

using namespace std::chrono;

struct Employee {

    int id;

    string name;

    string department;

    float salary;

};

int medianOfThree(vector<Employee>& arr, int low, int high) {

    int mid = low + (high - low) / 2;

    int a = arr[low].id;

    int b = arr[mid].id;

    int c = arr[high].id;

    if ((a < b && b < c) || (c < b && b < a)) return mid;

    else if ((b < a && a < c) || (c < a && a < b)) return low;

    else return high;

}

// Optimized Quick Sort with Median-of-Three

int partition(vector<Employee>& arr, int low, int high) {

    int pivotIndex = medianOfThree(arr, low, high);

    swap(arr[pivotIndex], arr[high]);

    int pivot = arr[high].id;

    int i = low - 1;

    for (int j = low; j < high; ++j) {

        if (arr[j].id <= pivot) {

            ++i;

            swap(arr[i], arr[j]);

        }

    }

    swap(arr[i + 1], arr[high]);

    return i + 1;

}

void quickSort(vector<Employee>& arr, int low, int high) {

    if (low < high) {

        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);

        quickSort(arr, pi + 1, high);

    }

}

int main() {

    string filename;

    cout << "Enter the filename: ";

    cin >> filename;

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Error opening file." << endl;

        return 1;

    }

    vector<Employee> employees;

    string line;

    // Time reading

    auto startRead = high\_resolution\_clock::now();

    while (getline(inputFile, line)) {

        if (line.find("Employee ID:") != string::npos) {

            Employee emp;

            emp.id = stoi(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Name

            emp.name = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Department

            emp.department = line.substr(line.find(":") + 1);

            getline(inputFile, line);  // Salary

            emp.salary = stof(line.substr(line.find(":") + 1));

            getline(inputFile, line);  // Blank line

            employees.push\_back(emp);

        }

    }

    auto endRead = high\_resolution\_clock::now();

    auto readDuration = duration\_cast<milliseconds>(endRead - startRead);

    cout << "\nTime taken to read records: " << readDuration.count() << " ms" << endl;

    inputFile.close();

    cout << "Total records: " << employees.size() << endl;

    auto startSort = high\_resolution\_clock::now();

    quickSort(employees, 0, employees.size() - 1);

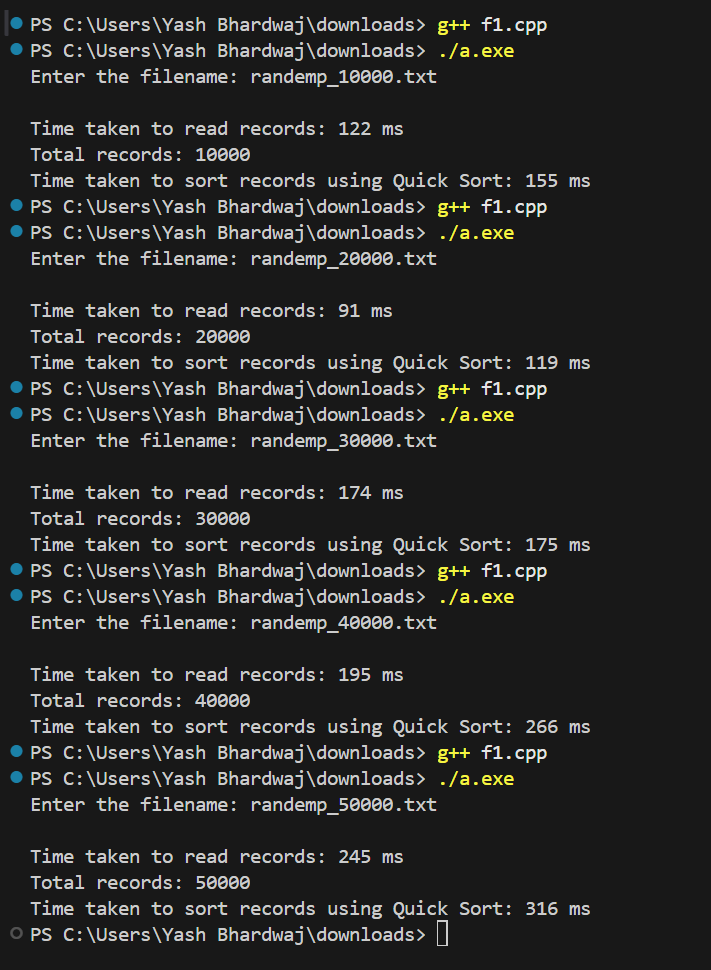
    auto endSort = high\_resolution\_clock::now();

    auto sortDuration = duration\_cast<milliseconds>(endSort - startSort);

    cout << "Time taken to sort records using Quick Sort: " << sortDuration.count() << " ms" << endl;

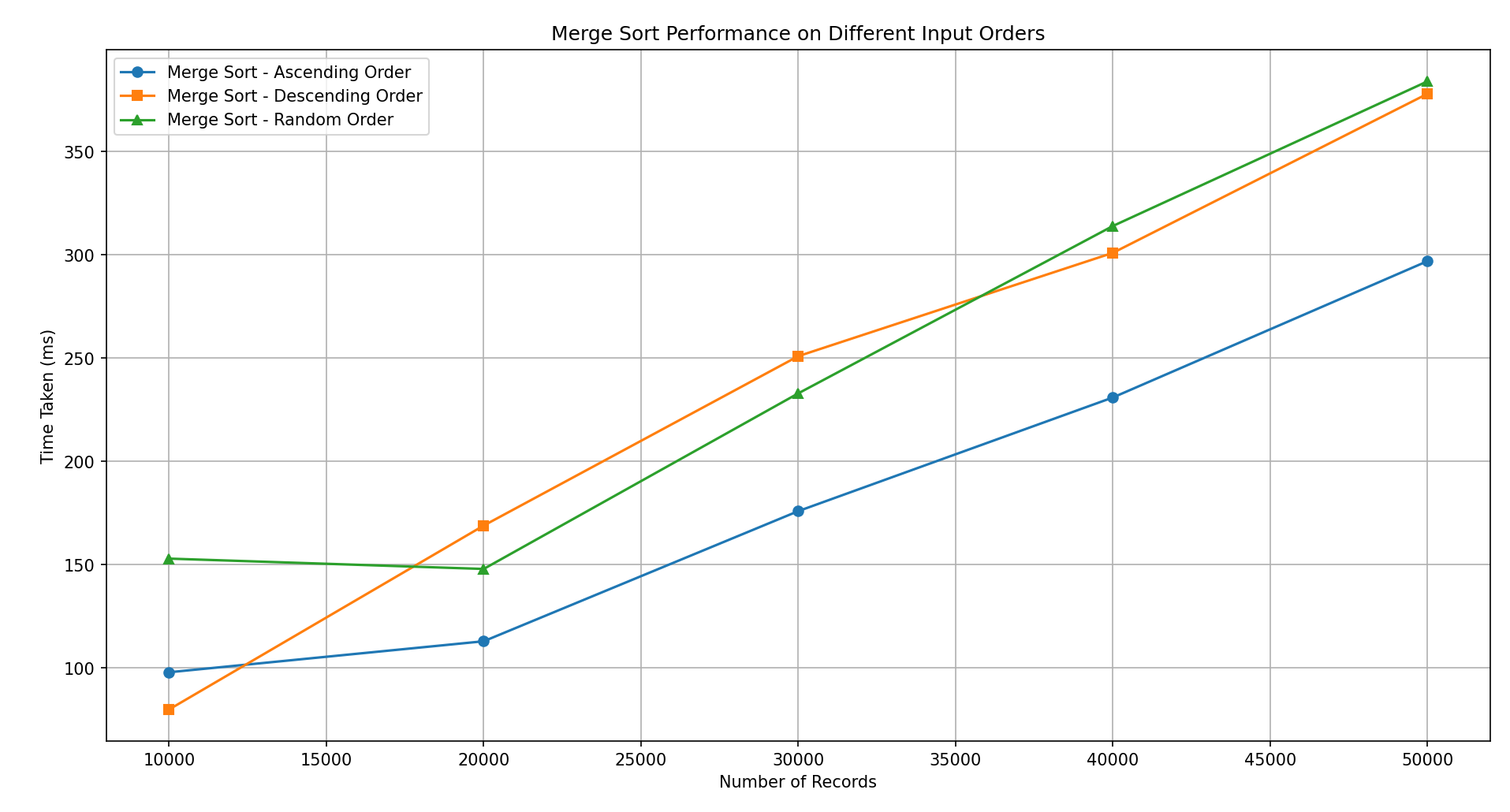
    return 0;

}



**GRAPHS**

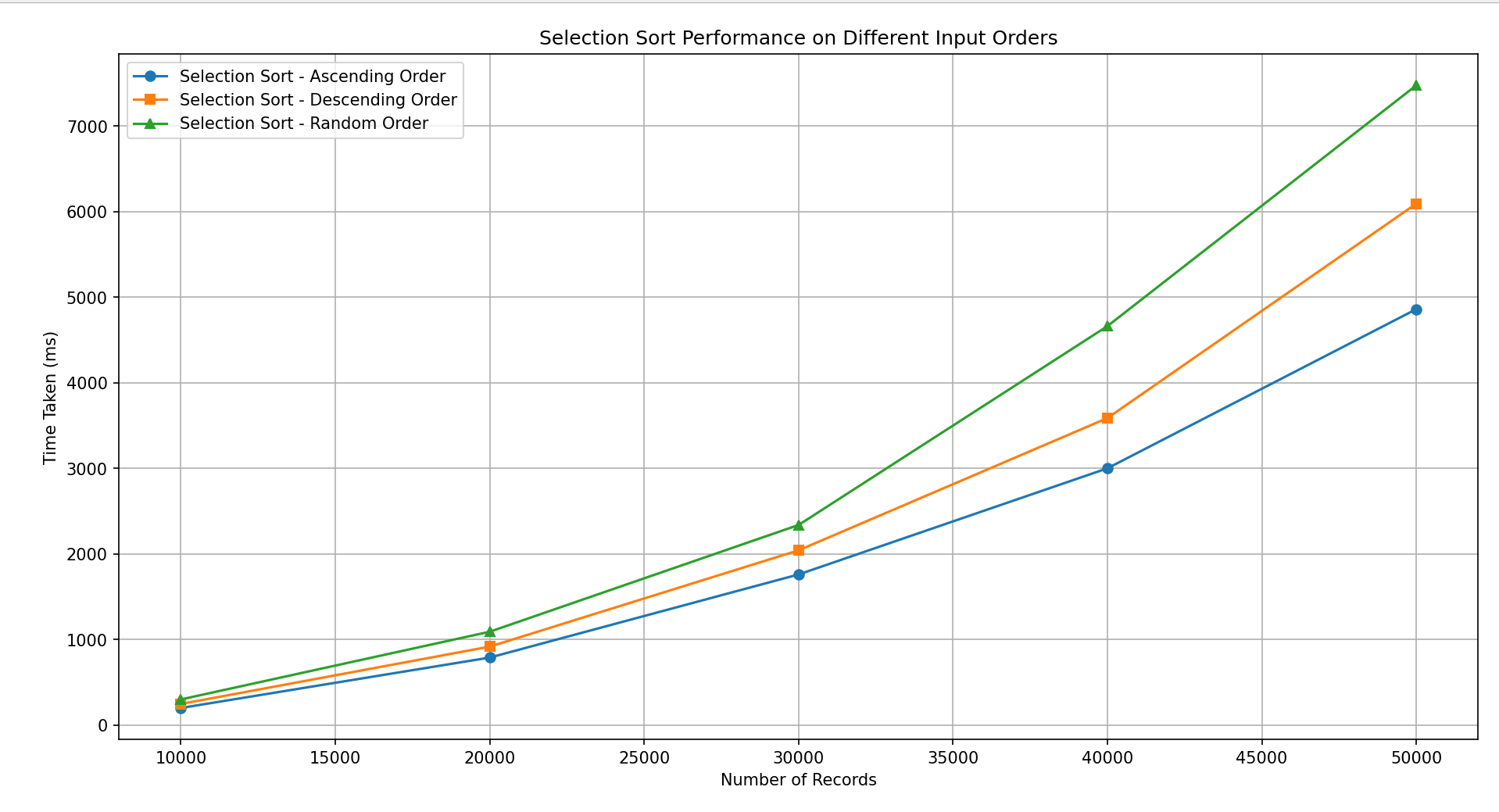
MERGE SORT



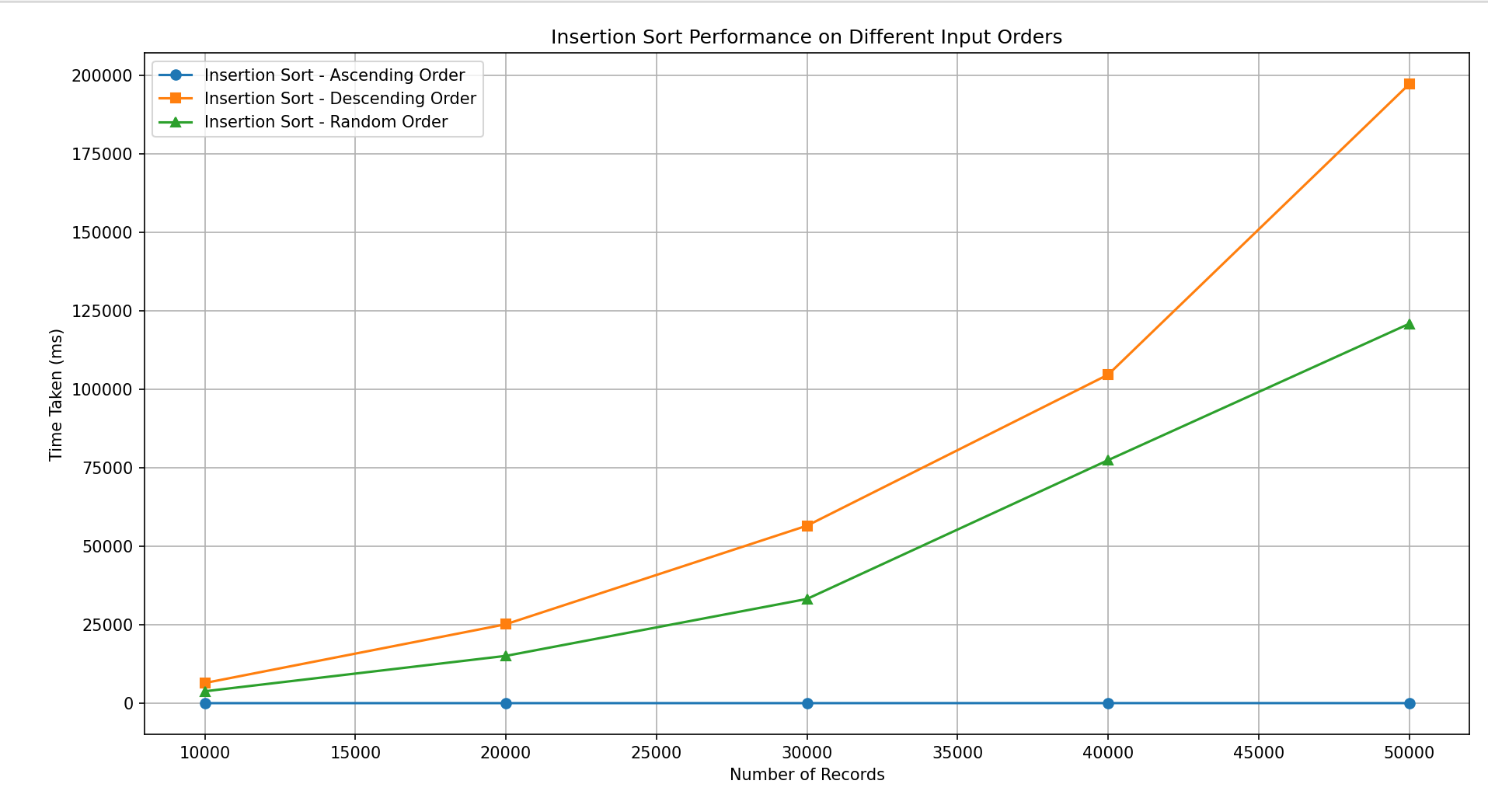
QUICK SORT



SELECTION SORT



INSERTION SORT



**TABLES**

**ASCENDING ORDER**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Records | Insertion | Selection | Merge | Quick |
| 10000 | 3 | 200 | 98 | 45 |
| 20000 | 7 | 789 | 113 | 95 |
| 30000 | 9 | 1760 | 176 | 167 |
| 40000 | 18 | 3000 | 231 | 209 |
| 50000 | 14 | 4859 | 297 | 234 |

**DESCENDING ORDER**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Records | Insertion | Selection | Merge | Quick |
| 10000 | 6416 | 247 | 80 | 111 |
| 20000 | 25135 | 917 | 169 | 206 |
| 30000 | 56521 | 2042 | 251 | 370 |
| 40000 | 104625 | 3587 | 301 | 509 |
| 50000 | 197153 | 6092 | 378 | 629 |

**RANDOM ORDER**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Records | Insertion | Selection | Merge | Quick |
| 10000 | 3780 | 300 | 153 | 155 |
| 20000 | 15059 | 1091 | 148 | 119 |
| 30000 | 33192 | 2338 | 233 | 175 |
| 40000 | 77393 | 4661 | 314 | 266 |
| 50000 | 120835 | 7474 | 384 | 316 |

**ANALYSIS**

Insertion Sort:

* Ascending input: Fastest (O(n)), nearly no shifting.
* Descending input: Slowest (O(n²)), every element must be shifted.
* Random input: Still O(n²), between best and worst case.

Selection Sort:

* Always O(n²) regardless of input order—same number of comparisons/swaps.
* Input order has almost no impact on performance.

Merge Sort:

* Always O(n log n), input order doesn’t matter.
* Consistently efficient across all cases.

Quick Sort:

* Random input: Fastest (O(n log n)).
* Descending input: Approaches worst case (O(n²)) if pivot choice is bad.
* Input order significantly affects performance.