Symbiosis Institute of Technology, Pune

Department of Computer Science and Engineering

Academic Year 2025-26

Design and Analysis of Algorithms– Lab

Batch 2023-27 - Sem V

Lab Assignment No:- 2	
Name of Student	Deepti Pal
PRN No.	23070122081
Batch	TY CSE (2023-27)
Class	A3
Academic Year & Semester	2025-26, 3rd year, 5th Semester
Date of Submission	3rd August 2025
Title of Assignment:	WAP to perform Quick sort, Merge sort and display the partial pass-wise sorting done. **Problem Given: Sorting Transaction Data** **Scenario:** A financial institution is analysing its customers' spending behaviour. To identify high-value transactions and spending trends, they need to **sort transaction records by transaction amount** in ascending order. You are required to implement and compare **Quick Sort** and **Merge Sort** algorithms to perform this task efficiently. Additionally, track how sorting progresses over multiple passes and measure the performance of each algorithm. **Input Specifications:** • Generate a dataset of at least **50 transaction records**.

	• Each transaction record must include:
	  ○ **Transaction ID** (e.g., TXN5238, unique alphanumeric ID)
	  ○ **Customer Name** (e.g., randomly generated)
	  ○ **Transaction Amount** (a float value between ₹100.00 and ₹1,00,000.00)
	  ○ **Timestamp** (in YYYY-MM-DD HH:MM:SS format)
	
	Sorting Requirement:
	• **Primary Key:** Sort based on **Transaction Amount** (lowest to highest).
	• **Stability Check (for Merge Sort):** If two transactions have the same amount, maintain their original input order.
	
	Implementation Tasks:
	1. **Implement Quick Sort and Merge Sort** on the transaction list.
	2. **Track the number of key comparisons** made during sorting.
	3. **Display partial pass-wise outputs**: For each algorithm, show the list after **at least the first 5 key passes/recursive stages**.
	
	Performance Metrics (Display):
	  ○ **Number of comparisons** made by each algorithm.
	  ○ **Time taken** to sort the data (in milliseconds or microseconds).
	  ○ **Sorted list** by transaction amount.
	
	Constraints & Validation Rules:
	• All Transaction IDs must be unique (prefix with TXN + random digits).
	• Transaction amounts should be valid floating-point numbers (₹100.00 – ₹1,00,000.00).
	• Timestamps should be properly formatted and represent valid dates within the last 30 days.

	Sample Output Format (for Pass-Wise Display): --- Quick Sort Pass 1 --- [500.25, 300.10, 700.00, 950.45, 120.00, 850.35] --- Quick Sort Pass 2 --- [120.00, 300.10, 500.25, 700.00, 850.35, 950.45] ... Total Comparisons: 72 Time Taken: 0.0023 seconds
Theory: (Handwritten)	1. Apply Quick Sort, Merge on 10 input items and display the partial pass-wise sorting done. 2. Explain time complexities of both algorithms.

Source code (Implementation Screenshot)	

```
#include<iostream>
#include<fstream>
#include<sstream>
#include<string>
#include<ctime>
#include<vector>
#include<chrono>
#include<iomanip>
using namespace std;

time_t parseTimestamp(const string &ts) {
	struct tm tmStruct = {};
	strptime(ts.c_str(), "%Y-%m-%d %H:%M:%S", &tmStruct);
	return mktime(&tmStruct);
}

string formatTimeStamp(time_t t) {
	char time_r[20];
	strftime(time_r, sizeof(time_r), "%Y-%m-%d %H:%M:%S",
localtime(&t));
	return string(time_r);
}


class Customer {
protected:
	int customerID;
	string  customerName;
	double balance;

public:
	Customer() {}
	Customer (int id, string name, double balance) {
		customerID = id;
		customerName = name;
		this->balance = balance;
	}
};

class Transaction : public Customer {
private:
	int transactionID;
	double transactionAMt;
	time_t timeStamp;

public:
	Transaction() {}
	Transaction(int tid, int id, string name, double bal, double amt,
time_t ts ) {
```

```
		transactionID = tid;
		customerID = id;
		customerName = name;
		balance = bal;
		transactionAMt = amt;
		timeStamp = ts;
	}

	int getTransactionID() const {
		return transactionID;
	}
	int getCustomerID() const {
		return customerID;
	}
	string getCustomerName() const {
		return customerName;
	}
	double getBalance() const {
		return balance;
	}
	double getTransactionAmt() const {
		return transactionAMt;
	}
	time_t getTimeStamp() const {
		return timeStamp;
	}
};

void swap(Transaction &a, Transaction &b) {
	Transaction temp = a;
	a = b;
	b = temp;
}

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

void quickSort(vector<Transaction> &arr, int low, int high, int &pass) {
	if(low < high) {
		int p = partitions(arr, low, high);
```

```
		cout << "Quick Sort Pass " << pass++ << ": ";
		for (auto &tx : arr) cout << tx.getTransactionID() << " ";
		cout << "\n";

		quickSort(arr, low, p-1, pass);
		quickSort(arr, p+1, high, pass);
	}
}
void merge(vector<Transaction> &arr, int left, int mid, int right, int &pass)
{
	int n1 = mid-left+1;
	int n2 = right-mid;

	vector<Transaction> 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].getTransactionAmt() <= R[j].getTransactionAmt())
arr[k++] = L[i++];
		else arr[k++] = R[j++];
	}
	while(i<n1) arr[k++] = L[i++];
	while(j<n2) arr[k++] = R[j++];

	cout << "Merge Sort Pass " << pass++ << ": ";
	for (auto &tx : arr) cout << tx.getTransactionID() << " ";
	cout << "\n";
}

void mergeSort(vector<Transaction> &arr, int left, int right, int &pass) {
	if(left < right) {
		int mid = (left+right)/2;
		mergeSort(arr, left, mid, pass);
		mergeSort(arr, mid+1, right, pass);
		merge(arr, left, mid, right, pass);
	}
}


void writeFile(const string &f, const vector<Transaction> &arr) {
	ofstream file(f);
	if(!file.is_open()) {
		cerr << "File is not opening!" << endl;
		return ;
	}
```

```
	file <<
"TransactionID,CustomerID,CustomerName,Balance,TransactionAmount,T
imeStamp\n";
	for(auto &tx : arr) {
		file << tx.getTransactionID() << ","
		    << tx.getCustomerID() << ","
		    << tx.getCustomerName() << ","
		    << tx.getBalance() << ","
		    << tx.getTransactionAmt() << ","
		    << formatTimeStamp(tx.getTimeStamp()) << "\n";
	}
	file.close();
}


int main()
{
	string inputFile;
	cout << "Enter the inputFile: ";
	cin >> inputFile;
	ifstream file(inputFile);
	if(!file.is_open()) {
		cerr << "File is not opening!" << endl;
		return 1;
	}

	vector<Transaction> arr;

	string line;
	getline(file, line); // skip header

	while(getline(file, line)) {
		if(line.empty()) continue;
		stringstream ss(line);
		string tID, cID, cName, balstr, amtstr, tsstr;

		getline(ss, tID, ',');
		getline(ss, cID, ',');
		getline(ss, cName, ',');
		getline(ss, balstr, ',');
		getline(ss, amtstr, ',');
		getline(ss, tsstr);

		if (balstr.empty() || amtstr.empty()) continue;

		Transaction tx(stoi(tID), stoi(cID), cName, stod(balstr),
stod(amtstr), parseTimestamp(tsstr));
		arr.emplace_back(tx);
	}
```

```
        vector<Transaction> quickArr = arr;
        vector<Transaction> mergeArr = arr;

        int qPass = 1, mPass = 1;

        // Quick Sort timing
        auto start = chrono::high_resolution_clock::now();
        quickSort(quickArr, 0, quickArr.size()-1, qPass);
        auto end = chrono::high_resolution_clock::now();
        double quickTime = chrono::duration<double>(end-start).count();

        // Merge Sort timing
        start = chrono::high_resolution_clock::now();
        mergeSort(mergeArr, 0, mergeArr.size()-1, mPass);
        end = chrono::high_resolution_clock::now();
        double mergeTime = chrono::duration<double>(end-start).count();

        string quickfile, mergefile;
        cout << "Enter output file for quick sort: ";
        cin >> quickfile;
        cout << "Enter output file for mergesort: ";
        cin >> mergefile;

        writeFile(quickfile, quickArr);
        writeFile(mergefile, mergeArr);

        cout << "\nQuick Sort Time: " << quickTime << " secs\n";
        cout << "Merge Sort Time: " << mergeTime << " secs\n";

        file.close();
        return 0;
}
```

Input file:

Draft version

Run | Debug | Stop | Save | { } Beautify | submit | Languag

main.cpp | ipSample_a2.txt | mergeFile.txt | quickFile.txt

```
TransactionID,CustomerID,CustomerName,Balance,TransactionAmt,Timestamp
1,101,Alice,5000,200,2025-08-01 10:23:45
2,102,Bob,3000,150,2025-07-31 18:45:10
3,103,Charlie,4500,500,2025-08-01 08:30:00
4,104,Diana,6000,250,2025-07-30 14:05:50
5,105,Edward,7000,100,2025-08-02 09:10:30
```

Output Screenshots

Draft version

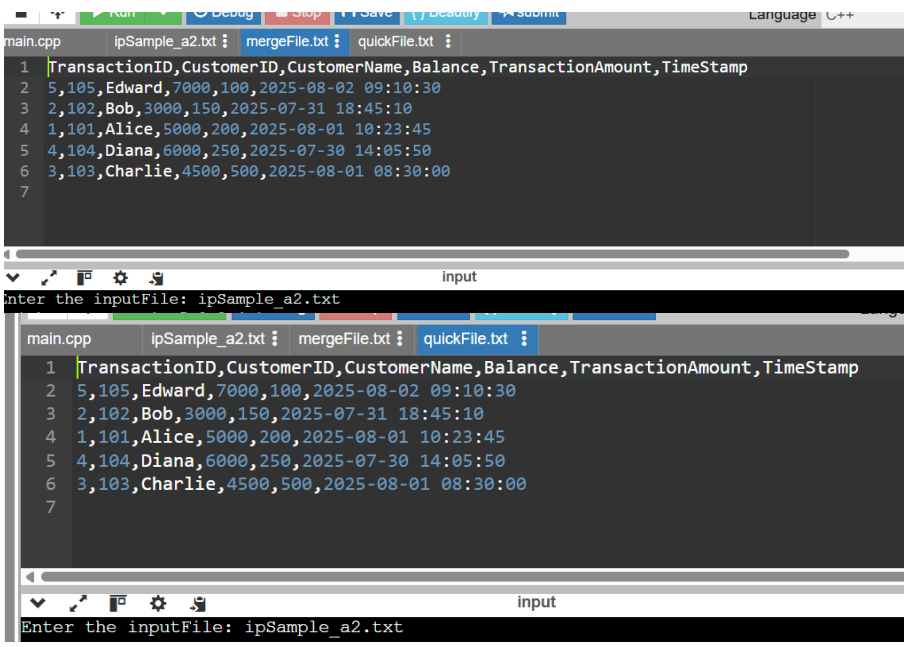
Run | Debug | Stop | Save | { } Beautify | submit | Langua

main.cpp | ipSample_a2.txt | mergeFile.txt | quickFile.txt

```
void quickSort(vector<Transaction> &arr, int low, int high, int &pass) {
    if(low < high) {
        int p = partitions(arr, low, high);

        cout << "Quick Sort Pass " << pass++ << ": ";
        for (auto &tx : arr) cout << tx.getTransactionID() << " ";
        cout << "\n";
```

input

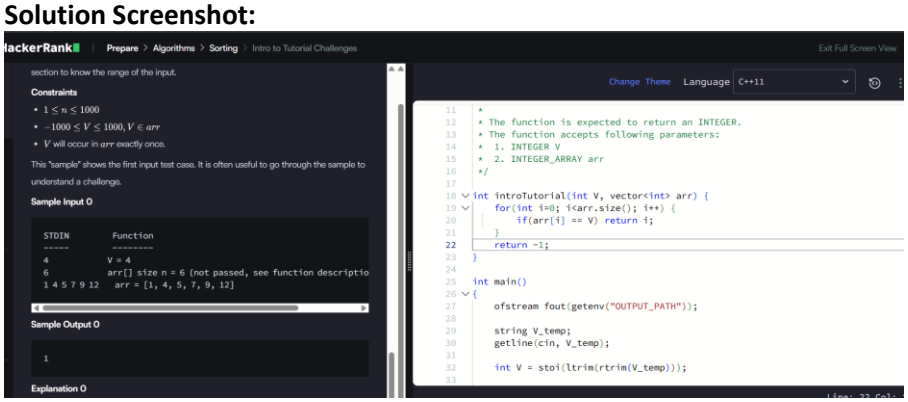
```
Enter the inputFile: ipSample_a2.txt
Quick Sort Pass 1: 5 2 3 4 1
Quick Sort Pass 2: 5 2 1 4 3
Quick Sort Pass 3: 5 2 1 4 3
Merge Sort Pass 1: 2 1 3 4 5
Merge Sort Pass 2: 2 1 3 4 5
Merge Sort Pass 3: 2 1 3 5 4
Merge Sort Pass 4: 5 2 1 4 3
Enter output file for quick sort: quickFile.txt
Enter output file for mergesort: mergeFile.txt

Quick Sort Time: 3.17e-05 secs
Merge Sort Time: 3.596e-05 secs


...Program finished with exit code 0
Press ENTER to exit console.
```

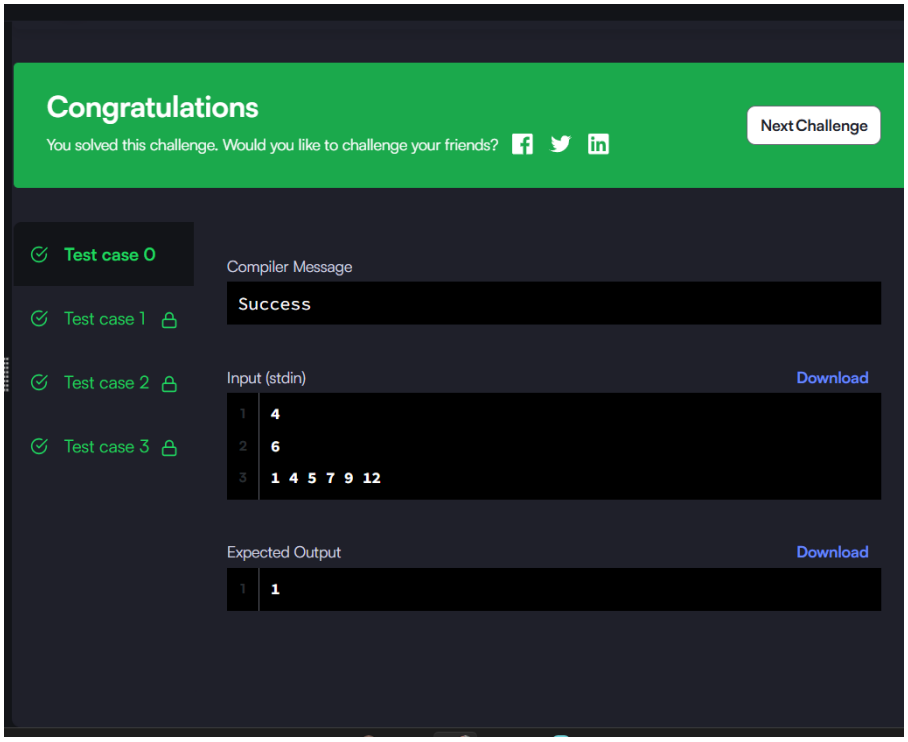


Problems Solved from Hacker Rank (Minimum 4)

1. https://www.hackerrank.com/challenges/tutorial-intro/problem?isFullScreen=true

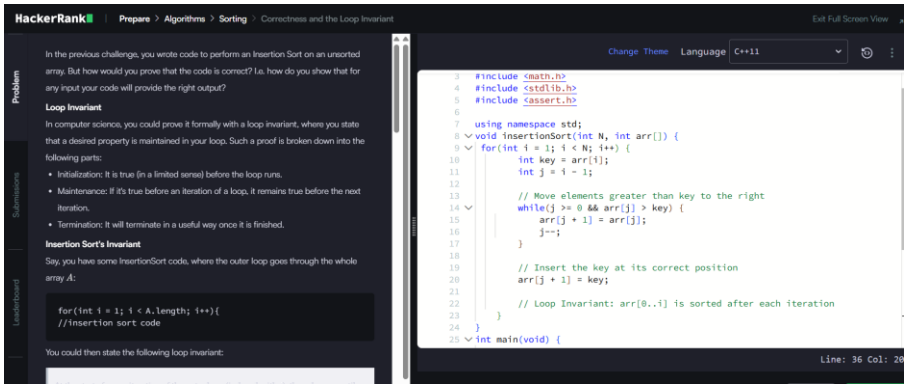
Solution Screenshot:

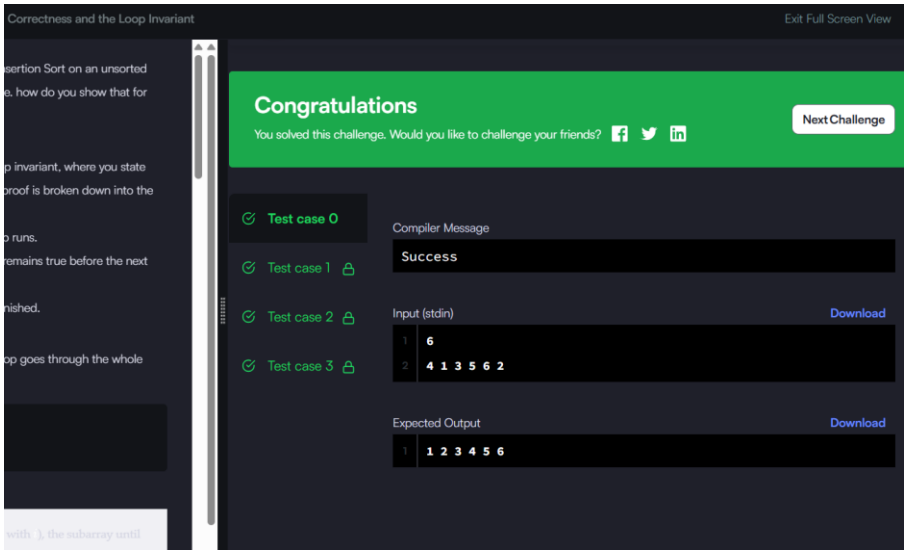




2. https://www.hackerrank.com/challenges/correctness-invariant/problem?isFullScreen=true

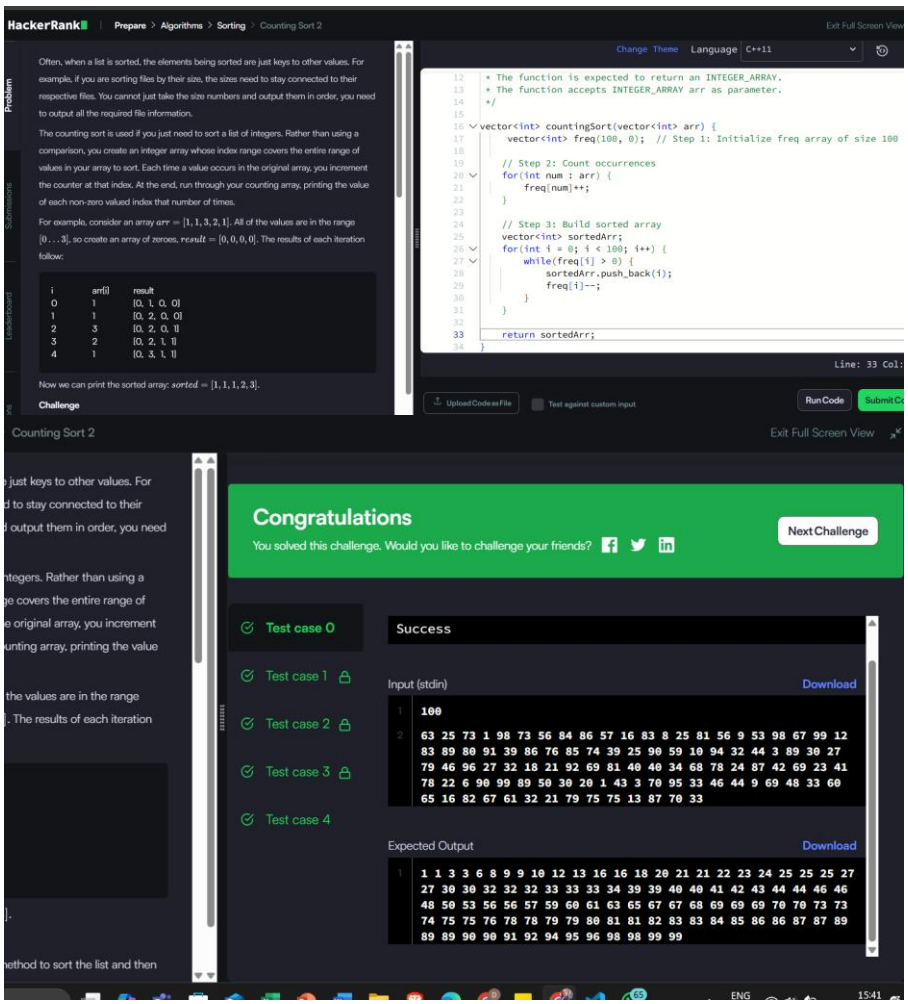
Solution Screenshot:





3. https://www.hackerrank.com/challenges/countingsort2/problem?isFullScreen=true

Solution Screenshot:



4. https://www.hackerrank.com/challenges/runningtime/problem?isFullScreen=true

Solution Screenshot: