Performance Report

SUBJECT: ADA

SUBMITTED TO:
MS. SHIVANGI RAJPUT

Algorithm Analysis

Analysis of efficiency of an algorithm can be performed at two different stages, before implementation and after implementation, as

A priori analysis – This is defined as theoretical analysis of an algorithm. Efficiency of algorithm is measured by assuming that all other factors e.g., speed of processor, are constant and have no effect on implementation.

A posterior analysis – This is defined as empirical analysis of an algorithm. The chosen algorithm is implemented using programming language. Next the chosen algorithm is executed on target computer machine. In this analysis, actual statistics like running time and space needed are collected.

Algorithm analysis is dealt with the execution or running time of various operations involved. Running time of an operation can be defined as number of computer instructions executed per operation.

Algorithm Complexity

The complexity of an algorithm f(N) provides the running time and / or storage space needed by the algorithm with respect of N as the size of input data.

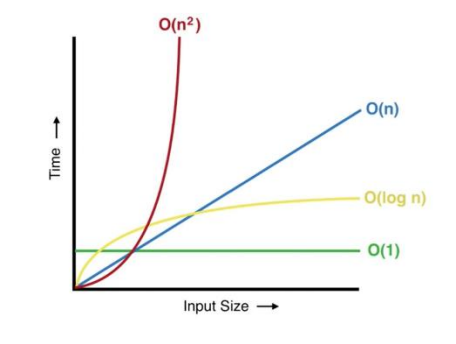
Suppose X is treated as an algorithm and N is treated as the size of input data, the time and space implemented by the Algorithm X are the two main factors which determine the efficiency of X.

Time Factor – The time is calculated or measured by counting the number of key operations such as comparisons in sorting algorithm.

Space Factor – The space is calculated or measured by counting the maximum memory space required by the algorithm.

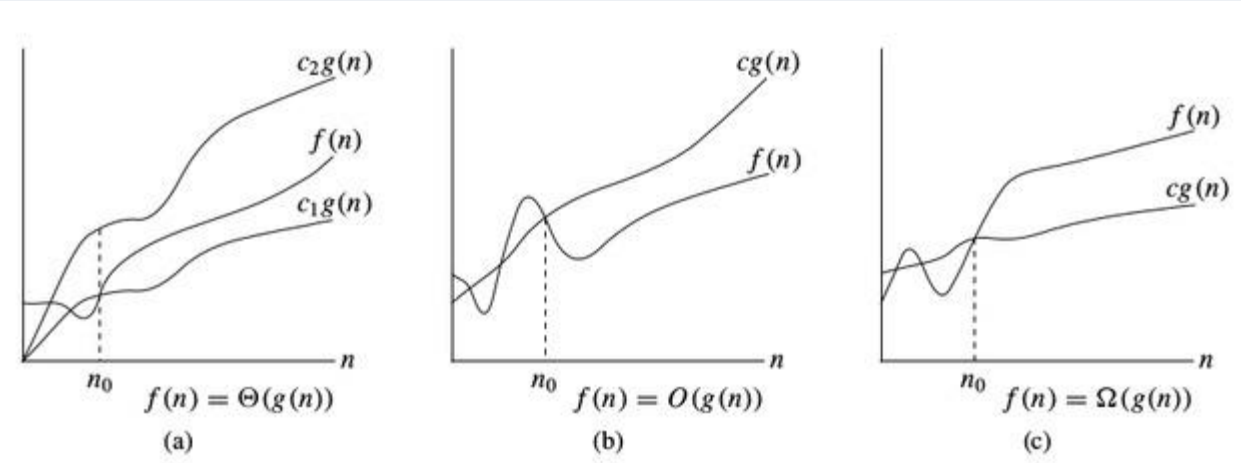
Time Complexity

Time complexity is a concept in computer science that deals with the quantification of the amount of time taken by a set of code or algorithm to process or run as a function of the amount of input. In the other words, time complexity is essentially efficiency, or how long a program function takes to process a given input.



Space Complexity

The space complexity of an algorithm or a computer program is the amount of memory space required to solve an instance of the computational problem as a function of characteristics of the input. It is the memory required by an algorithm until it executes completely.



Project Complexities

Time Complexities

Product Details

Time Complexity: O(m*n)

Where,

m = no. of records in the ProductRecords table
n = parameters called from ProductRecords table

(Here 3 fields are: SU, Product_Name, and Cost)

SQL Query: -

pst = con.prepareStatement("SELECT * FROM ProductRecords ORDER BY SU ASC");

```
pst = con.prepareStatement( sql: "SELECT * FROM ProductRecords ORDER BY SU ASC");
rs = pst.executeQuery();
```

Customer Details

Time Complexity: O(m*n)

Where,

m = no. of records in the CustomerRecords table
n = parameters called from ProductRecords table

(Here 8 fields are: Customer_Name, Customer_Phno, Customer_Email, Customer_Address, Seller, Total_Price, Purchase_Time, Bill_No)

SQL Query: -

String sql = "SELECT * FROM CustomerRecords ORDER BY Bill_No DESC";

```
String sql = "SELECT * FROM CustomerRecords ORDER BY Bill_No DESC";
ps = con.prepareStatement(sql);
rs = ps.executeQuery();
```

Login

Time Complexity: O(m*n)

Where,

m = no. of records/rows in Registration table
n = parameters called from Registration table

(Here 2 fields are: E-mail and password)

SQL Query: -

String sql = "SELECT * FROM Registration WHERE Email = ? AND Password = ?";

```
String sql = "SELECT * FROM Registration WHERE Email = ? AND Password = ?";
ps = con.prepareStatement(sql);
ps.setString( parameterIndex: 1, EnteredEmail);
ps.setString( parameterIndex: 2, EnteredPassword);
rs = ps.executeQuery();
```

Edit Profile

Time Complexity: O(m*n*o)

Where,

m = no. of records in Registration table
n = no. of columns in Registration table

(Here 7 fields are: r.Name, r.DOB, r.Email, r.Mobile_Number, r.Gender, r.Address, r.Password)

o = no. of records in Login table

(Here 7 fields are: r.Name, r.DOB, r.Email, r.Mobile_Number, r.Gender, r.Address, r.Password)

SQL Query: -

query1 = "SELECT r.Name, r.DOB, r.Email, r.Mobile_Number, r.Gender, r.Address, r.Password FROM Registration r, Login l WHERE r.Email = l.UID";

```
query="SELECT r.Name, r.DOB, r.Email, r.Mobile_Number, r.Gender, r.Address, r.Password FROM Registration r, Login l WHERE  r.Email = l.UID;";
rs = stmt.executeQuery(query);
```

Bill Generation

Time Complexity: O(klog(k)*m*n)

Where,

m = no. Of records in ProductRecords table
n = no. Of columns in ProductRecords table

(Here 2 fields are: Cost, ProductName)

k = no of records in Customer_Details table

(Sorting is performed according to Bill_No)

SQL Query: -

query1 = "select c.Bill_No , p.Product_Name,p.Cost from CustomerRecords c , ProductRecords p where p.SU='"+psq+"'order by c.Bill_No desc limit 1";

```
query1 = "select c.Bill_No , p.Product_Name,p.Cost, p.SU from CustomerRecords c , ProductRecords p where p.SU='"+psq+"'order by c.Bill_No desc limit 1";
pst = con.prepareStatement(query1);
rs = pst.executeQuery();
```

Registration

Time Complexity: O(n)

Where,

n = no. of columns in Registration table

SQL Query: -

query="Insert into Registration values ('"+uname+" ',' "+udob+"', '"+uemail+"', '"+uphno+"', '"+ugender+"', '"+uaddress+"', '"+upassword+"')";

```
query = "Insert into Registration values ('"+uname+" ',' "+udob+"','"+uemail+"','"+uphno+"','"+ugender+"','"+uaddress+"','"+upassword+"'); ";
stmt = con.createStatement();
new_register=stmt.executeUpdate(query);
```

Final Bill

Time Complexity: O(p*q*r*s)

Where,

p = no. of records in ProductRecords table
q = parameters called from ProductRecords table

(Here 2 fields are: Product_Name and Cost)

r = no. of records in Bill table
s = parameters called from ProductRecords table

(Here 2 fields are: SU and QTY)

SQL Query: -

billQuery = "select p.Product_Name , p.Cost , b.SU, b.QTY from ProductRecords p , Bill b where b.Bill_No=? AND b.SU=p.SU";

```
billQuery = "select p.Product_Name, p.Cost, b.SU, b.QTY from ProductRecords p , Bill b where b.Bill_No=? AND b.SU=p.SU";
pst=con.prepareStatement(billQuery);
pst.setString( parameterIndex: 1,billno);
rs=pst.executeQuery();
```

Customer Bill Info

Time Complexity: O(p*q*r*s)

Where,

p = no. of records in ProductRecords table
q = parameters called from ProductRecords table

(Here 2 fields are: Product_Name and Cost)

r = no. of records in Bill table
s = parameters called from ProductRecords table

(Here 2 fields are: SU and QTY)

SQL Query: -

billQuery = "select p.Product_Name , p.Cost , b.SU, b.QTY from ProductRecords p , Bill b where b.Bill_No=? AND b.SU=p.SU";

```
billQuery = "select p.Product_Name , p.Cost , b.SU, b.QTY from ProductRecords p , Bill b where b.Bill_No=? AND b.SU=p.SU";
pst=con.prepareStatement(billQuery);
pst.setInt( parameterIndex: 1,bill_no);
rs=pst.executeQuery();
```

Dashboard

Time Complexity: O(m*n)

Where,

m = number of rows in Registration table
n = parameters called from Registration table

(Here 3 fields are: Name, Email, Mobile_Number)

SQL Query: -

QUERY = "select r.Name, r.Email, r.Mobile_Number, l.UID from Registration r, Login l where r.Email = l.UID ORDER BY l.SNO DESC LIMIT 1";

```
QUERY = "select r.Name, r.Email, r.Mobile_Number, l.UID from Registration r, Login l where r.Email = l.UID ORDER BY l.SNO DESC LIMIT 1";
Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
PreparedStatement stmt = conn.prepareStatement(QUERY);
ResultSet rs = stmt.executeQuery();
```

Shop Details

Time Complexity: O(n)

Where,

n: number of parameters called from ShopDetails table

(Here 6 fields are: ID, Shop_Name, Owner_Name, Phone_Number, Email, Shop_Address)

SQL Query: -

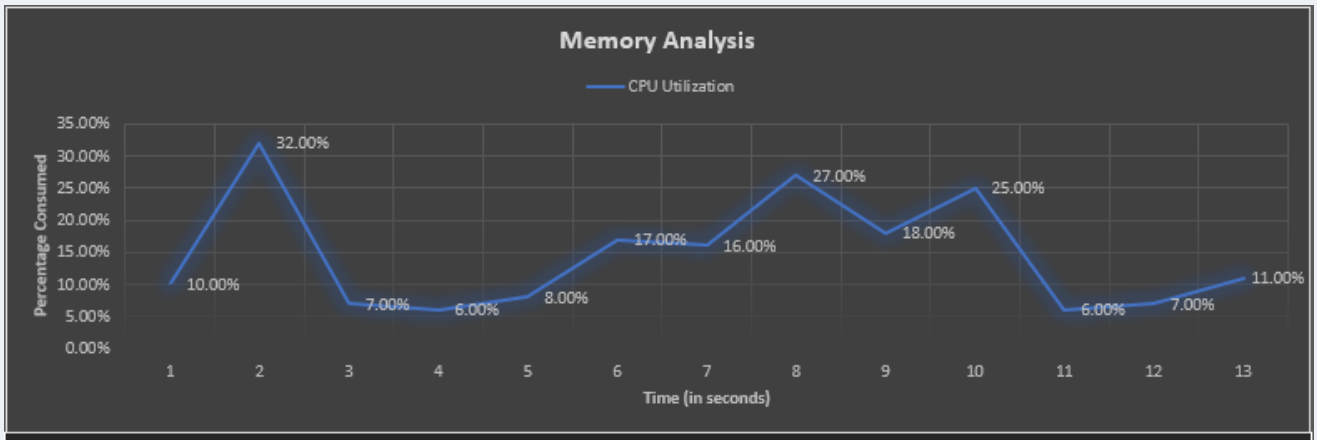
shopQuery = "SELECT * FROM ShopDetails"

```
shopQuery="SELECT * FROM ShopDetails";
rs = stmt.executeQuery(shopQuery);
```

Space Complexities

CPU UTILIZATION

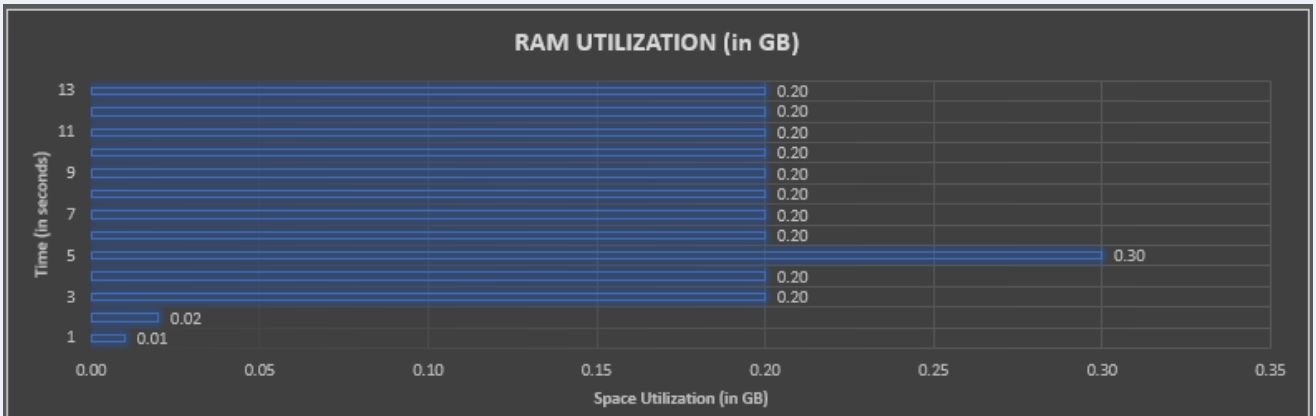
This data and graph depict the CPU usage on the startup of the APP.



CPU UTILIZATION	
SNO	UTILIZATION (%)
1	10.00%
2	32.00%
3	7.00%
4	6.00%
5	8.00%
6	17.00%
7	16.00%
8	27.00%
9	18.00%
10	25.00%
11	6.00%
12	7.00%
13	11.00%
Analysis	
Max	32.00%
Min	6.00%
Average	14.62%
Initial (without starting the app)	10.00%

RAM UTILIZATION

This data and graph depict the RAM usage on the startup of the APP.

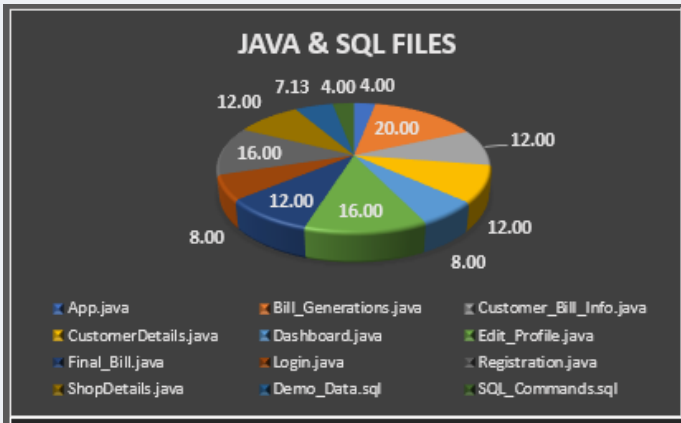


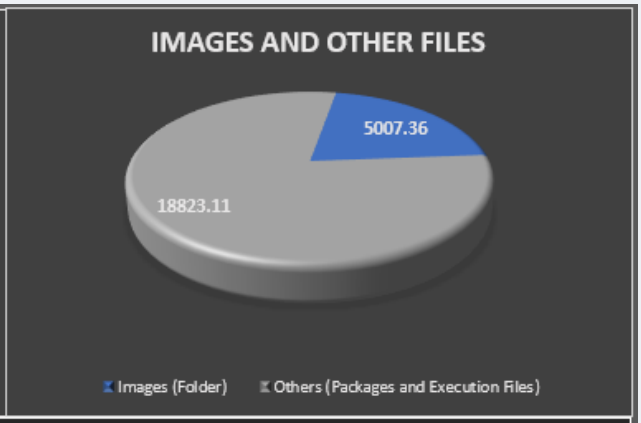
Ram Utilization		
SNO	TOTAL SYSTEM UTILIZATION (GB)	APP UTILIZATION (GB)
1	7.81	0.01
2	7.82	0.02
3	8.00	0.20
4	8.00	0.20
5	8.10	0.30
6	8.00	0.20
7	8.00	0.20
8	8.00	0.20
9	8.00	0.20
10	8.00	0.20
11	8.00	0.20
12	8.00	0.20
13	8.00	0.20

Analysis	
Max	8.10
Min	7.81
Average	7.98
Initial (without starting the app)	7.00
Max (APP)	0.30
Min (APP)	0.01
Average (APP)	0.18

STORAGE UTILIZATION

This data and graph depict the storage utilized on the disk.





STORAGE (HDD)		
SNO	FILE NAME	SIZE (in kb)
1	App.java	4.00
2	Bill_Generations.java	20.00
3	Customer_Bill_Info.java	12.00
4	CustomerDetails.java	12.00
5	Dashboard.java	8.00
6	Edit_Profile.java	16.00
7	Final_Bill.java	12.00
8	Login.java	8.00
9	Registration.java	16.00
10	ShopDetails.java	12.00
11	Demo_Data.sql	7.13
12	SQL_Commands.sql	4.00
13	Images (Folder)	5007.36
14	Others (Packages and Execution Files)	18823.11
Analysis		
Minimum Storage Required (Kb)		23961.60

DEVELOPED BY:

AYUSH LUTHRA
MANAS SHEKHAR
ALANKAR SHARMA
SAURABH WADHWA
KESHAV GARG
OM GUPTA