CSCI203 ASSIGNMENT 2 (REPORT)

1. A high-level description (in pseudo-code) of the overall solution strategy

```
Header:

DEFINE Customer(Struct):

        arrivalTime;

        serviceTime;

        priority;

DEFINE Teller(Struct):

        endTime = 0;

        workingTime = 0;

        customersServed = 0

DEFINE CustomerWaitedQueue (which uses a Array to implement a Queue):

    Customer* data;

    int head;

    int tail;

    int capacity;

Method:

        enqueue(Customer Object)
        dequeue(Delete elemtent[0])
        Condition (isEmpty, is Full)
        Peek(Element[0])
    -   All methods in definition (in main cpp file)
Main file:
 MAIN:
    DEFINE queue AS CUSOMTER LINE of size 100 (MAX)
    DEFINE waitingLine AS WaitingList of size 100 (MAX)
    DEFINE tellers[] FROM USER INPUT
    DEFINE
        currentTime
        totalWaitingTime
         maxQueueLength
         totalQueueingTime
         totalServiceTime
        currentOfCustomer

    READ from input file and add DATA in CUSTOMER
    WHILE queue is not empty OR teller is busy:
       WHILE arriving customers at currentTime(earlier/ontime):
        CHECK for a free teller using freeTeller ()
        IF found free tellers, teller will serve the customer
            UPDATE
            Working Time of TELLER
            END TIME of service
            Total Services
            Current of CUSTOMER
        ELSE add customer to WaitingList

    FOR each teller:
        IF teller finished serving with Customer AND WaitingList is not empty:
            teller serves next customer from WaitingList
            CALCULATE totalWaitingTime for served customer

    CHECK and UPDATE maxQueueLength
    UPDATE TIMESTEP
    INCREMENT currentTime = STARTING from ENDTIME

PRINT statistics:
    - Number of customers served by each teller
    - Total time of the simulation
    - Average service time per customer (excluded in queue)
    - Average waiting time per customer (included in service)
    - Maximum Length of Waiting Queue
    - Average Queue Length
    - Idle rate of each teller (ratio)

END MAIN
```

2. A complexity analysis of your solution with big-O notation and sufficient justification

- **Best Case**: The precise situation of your discrete event simulation and the beginning state of my data structures.
        - Array: O(1) (for access and insertion)
        - Priority Queue: O(log n)
        - Insertion Sort: O(n) (
        - Discrete Event Simulation: O(n)
        - File Reading: O(n)
        - Printing Output: O(n)
- **Average Case**:
    - This is a bit more complex to analyze precisely, as it depends on the distribution of data and events in your simulation. But assuming typical scenarios:
        - Array: O(1)
        - Priority Queue: O(log n)
        - Insertion Sort: O(n^2)
        - Discrete Event Simulation: (depending on processes)
        - File Reading: O(n)
        - Printing Output: O(n)
- **Worst Case**:
    - The worst-case complexities can be:
        - Array: O(n)
        - Priority Queue: O(log n)
        - Insertion Sort: O(n^2)
        - Discrete Event SimulationO(n)
        - File Reading: O(n)
        - Printing Output: O(n)

Total Complexity = O(n) (Worst Case)

Total Complexity = O(log n) (Average Case)

Total Complexity = O(n^2) (Worst Case)

3. A list of all of the data structures are used, and the reasons for use them.

- There are the list of DSA are used in my program (C++)

Data Structures:
a. Array: This structure is used to create tellers, including free tellers to assign to customers during the simulation process. This can solve the problem when one or more tellers need to handle orders.

b. Priority Queue: This structure is implemented by queue theories concept to store customers who arrive earlier than the time tellers complete their work. It will handle priority conditions (priority and arrival time) to arrange elements in the appropriate position when enqueuing the data from outside. The structure is built based on the basic properties of the queue, including enqueue (get data), dequeue (push the first element out of the queue), and check conditions in the queue (full, empty).

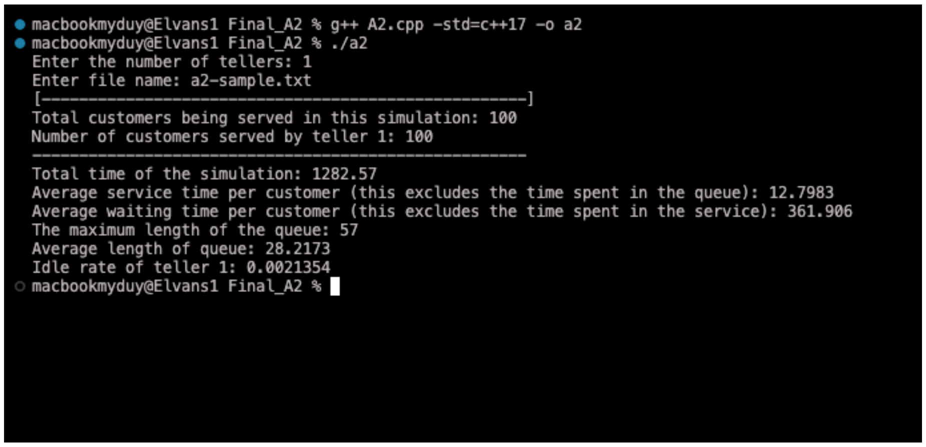
Algorithms:

c. (Priority Queue Handling - Enqueue) Insertion Sort When adding data to the queue, this technique will be utilised to process requests directly in the queue. They will be able to identify the qualities of the data that must be applied in descending and ascending order, respectively, to compare it with the remaining elements (priority and arrival).

d. Discrete Event Simulation: This technique simulates the many stages of the process by which clients and tellers interact. As well as figuring out how long it takes tellers to finish serving customers, how many people are served, and how much time each teller has free while no one is waiting to be served. This simulation so demonstrates the ideal time of one or more tellers.

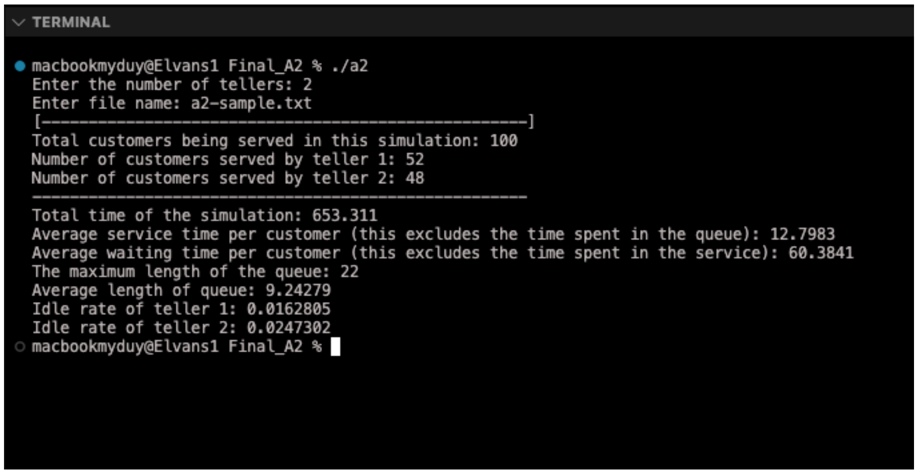
4. A snapshot of the compilation and the execution of your program on the provided "a2-sample.txt" file.

4.1 One Teller



```
● macbookmyduy@Elvans1 Final_A2 % g++ A2.cpp -std=c++17 -o a2
● macbookmyduy@Elvans1 Final_A2 % ./a2
  Enter the number of tellers: 1
  Enter file name: a2-sample.txt
  [--------------------------------------------------]
  Total customers being served in this simulation: 100
  Number of customers served by teller 1: 100
  ----------------------------------------------------
  Total time of the simulation: 1282.57
  Average service time per customer (this excludes the time spent in the queue): 12.7983
  Average waiting time per customer (this excludes the time spent in the service): 361.906
  The maximum length of the queue: 57
  Average length of queue: 28.2173
  Idle rate of teller 1: 0.0021354
○ macbookmyduy@Elvans1 Final_A2 %
```

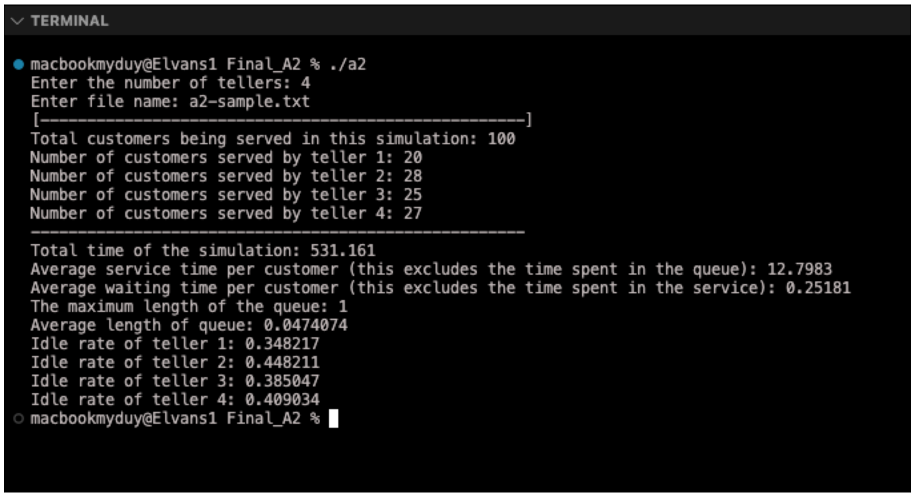
4.2 Two Tellers



```
TERMINAL

● macbookmyduy@Elvans1 Final_A2 % ./a2
  Enter the number of tellers: 2
  Enter file name: a2-sample.txt
  [--------------------------------------------------]
  Total customers being served in this simulation: 100
  Number of customers served by teller 1: 52
  Number of customers served by teller 2: 48
  ----------------------------------------------------
  Total time of the simulation: 653.311
  Average service time per customer (this excludes the time spent in the queue): 12.7983
  Average waiting time per customer (this excludes the time spent in the service): 60.3841
  The maximum length of the queue: 22
  Average length of queue: 9.24279
  Idle rate of teller 1: 0.0162805
  Idle rate of teller 2: 0.0247302
○ macbookmyduy@Elvans1 Final_A2 %
```

4.3 Four Tellers



```
TERMINAL

macbookmyduy@Elvans1 Final_A2 % ./a2
Enter the number of tellers: 4
Enter file name: a2-sample.txt
[--------------------------------------------------]
Total customers being served in this simulation: 100
Number of customers served by teller 1: 20
Number of customers served by teller 2: 28
Number of customers served by teller 3: 25
Number of customers served by teller 4: 27
-----------------------------------------------------
Total time of the simulation: 531.161
Average service time per customer (this excludes the time spent in the queue): 12.7983
Average waiting time per customer (this excludes the time spent in the service): 0.25181
The maximum length of the queue: 1
Average length of queue: 0.0474074
Idle rate of teller 1: 0.348217
Idle rate of teller 2: 0.448211
Idle rate of teller 3: 0.385047
Idle rate of teller 4: 0.409034
macbookmyduy@Elvans1 Final_A2 %
```

5. The outputs (three runs in total) are produced by your program on the provided “a2-sample.txt” file

5.1 One Teller

```
macbookmyduy@Elvans1 Final_A2 % ./a2
Enter the number of tellers: 1
Enter file name: a2-sample.txt
[--------------------------------------------------]
Total customers being served in this simulation: 100
Number of customers served by teller 1: 100
--------------------------------------------------
Total time of the simulation: 1282.57
Average service time per customer (this excludes the time spent in the queue): 12.7983
Average waiting time per customer (this excludes the time spent in the service): 361.906
The maximum length of the queue: 57
Average length of queue: 28.2173
Idle rate of teller 1: 0.0021354
```

5.2 Two Tellers

```
macbookmyduy@Elvans1 Final_A2 % ./a2
Enter the number of tellers: 2
Enter file name: a2-sample.txt
[--------------------------------------------------]
Total customers being served in this simulation: 100
Number of customers served by teller 1: 52
Number of customers served by teller 2: 48
--------------------------------------------------
Total time of the simulation: 653.311
Average service time per customer (this excludes the time spent in the queue): 12.7983
Average waiting time per customer (this excludes the time spent in the service): 60.3841
The maximum length of the queue: 22
Average length of queue: 9.24279
```

```
Idle rate of teller 1: 0.0162805
Idle rate of teller 2: 0.0247302
```

5.3 Four Tellers

```
macbookmyduy@Elvans1 Final_A2 % ./a2
Enter the number of tellers: 4
Enter file name: a2-sample.txt
[--------------------------------------------------]
Total customers being served in this simulation: 100
Number of customers served by teller 1: 20
Number of customers served by teller 2: 28
Number of customers served by teller 3: 25
Number of customers served by teller 4: 27
----------------------------------------------------
Total time of the simulation: 531.161
Average service time per customer (this excludes the time spent in the
queue): 12.7983
Average waiting time per customer (this excludes the time spent in the
service): 0.25181
The maximum length of the queue: 1
Average length of queue: 0.0474074
Idle rate of teller 1: 0.348217
Idle rate of teller 2: 0.448211
Idle rate of teller 3: 0.385047
Idle rate of teller 4: 0.409034
```

6. A discussion based on the statistics that how the teller's number affect the efficiency of the services.

Maintaining a high degree of efficiency is a major concern in any service system. The quantity of cashiers is one of the most crucial variables influencing performance. When there is only one cashier working, it is clear that consumers must wait a long time before obtaining assistance. Queues that are quickly expanding, lengthy wait times, and rising consumer unhappiness are the results. However, as one might imagine, adding more cashiers does not always provide the desired outcomes. With each additional cashier, resource costs rise along with the likelihood of issues like excessive idle rates. Particularly in the highly competitive corporate world of today, a resource that is not used properly might be viewed as waste.

Therefore, it becomes crucial to identify a solution that strikes a compromise between meeting client expectations and maximising resource efficiency. Optimising the number of cashiers using statistical analysis will assist firms in making the best decisions to satisfy consumer demands without wasting resources. To put it simply, ensuring good performance in service necessitates having a suitable number of cashiers as well as the ability and flexibility to alter this number in accordance with actual demands. Customers will be happy and resources will be employed efficiently as a result of a well-thought-out choice.