**ASSIGNMENT 2**

**QUEUES SIMULATOR**

**NUME : OSTAFIE STANCA**

**GRUPA: 30227**

**PROFESOR LABORATOR : DAN MITREA**

CUPRINS

[1. OBIECTIVUL TEMEI 3](#_Toc66881492)

[2. ANALIZA PROBLEMEI, MODELARE, SCENARII, CAZURI DE UTILIZARE 3](#_Toc66881493)

[3. PROIECTARE 4](#_Toc66881494)

[4. IMPLEMENTARE 6](#_Toc66881495)

[5. REZULTATE 7](#_Toc66881496)

[6. CONCLUZII 7](#_Toc66881497)

[7. BIBLIOGRAFIE 8](#_Toc66881498)

# 1. **OBIECTIVUL TEMEI**

Obiectivul principal al acestei teme este proiectarea si implementarea corecta a unei aplicatii ce are ca scop analiza sistemele bazate pe coadă pentru a determina și minimiza timpul de așteptare al clienților. Aceasta aplicatie trebuie sa genereze o lista de clienti si sa gestioneze repartizarea acestora la “cozi” intr-un mod cat mai efficient. Odata gestionati, clientii trebuie impartiti astfel incat sa obtinem un timp de asteptare cat mai redus.

Vom amintii si urmatoarele obiective secundare ale temei:

* La pornire, aplicatia va genera o lista de clienti, aleator, fiind nevoie de urmatoarele date pentru fiecare client : ID (un identificator unic pentru fiecare client generat in lista), arrival time ( acest camp va retine timpul la care clientul trebuie repartizat la cozi) si service time ( este timpul pe care clientul il va petrece la coada din momentul in care ajunge sa fie procesat). Variabilele arrival time si service time trebuie sa mai respecte de asemenea inca un criteriu la generare. Arrival time trebuie sa faca parte dintr-un interval [min Arrival Time, max Arrival Time], introdus la rularea aplicatiei. De asemenea, service time trebuie sa apartina intervalului [ min Service Time, max Service Time], interval introdus inainte de rularea aplicatiei.
* Implementarea clasei “Queue” ne va permite sa decidem functionalitatea thread-urilor pornite cu scopul de a procesa clientii.
* Implementarea clasei “Schedular” ne va permite sa pornim thread-urile pentru cele Q cozi. De asemenea, aceasta clasa contine si apelul functiei menite sa repartizeze un client la una din cele Q cozi create.
* Implementarea clasei “SimulationManager” ne va permite sa descriem functionalitatea thread-ului principal. Acesta trebuie sa distribuie o lista de N clienti , generata aleator, la Q cozi, in functie de timpul de asteptare la fiecare coada.

# **2. ANALIZA PROBLEMEI, MODELARE, SCENARII, CAZURI DE UTILIZARE**

* **Analiza problemei**

Scopul acestei teme este de a realiza o aplicatie capabila sa simuleze un scenariu din lumea reala folosindu-se de conceptul de Coada ce are ca obiectiv principal sa ofere un loc pentru ca un „client” să aștepte înainte de a primi un „serviciu”. Programul va prelua informatiile minime din interfata grafica pentru a putea simula scenariul : N (numarul de client generati aleator), Q ( numarul total de cozi/threads disponibile), simulation period( timpul total de simulare al programului), intervalul [min Arrival Time, max Arrival Time] si intervalul [min Service Time, max Service Time]. De asemenea, pentru a putea simula cat mai corect acest scenariu este nevoie sa tinem cont de cerintele minime ale unei programari eficiente.

* **Modelare**

Este important de retinut ca utilizatorul va furniza datele esentiale pentru a putea simula scenariul. Cu alte cuvinte utilizatorul este regizorul acestui scenariu, astfel ca este important sa introduca datele corespunzatoare, in campul corespunzator. Pentru o functionare corecta, datele trebuie introduse sub forma de cifre, alte forme de caractere nefiind permise. Datele din interfata sunt preluate sub forma de text si transformate in tip-uri intregi. Odata pornita aplicatia, aceasta va afisa in timp real rezultatul scenariului introdus, prezentand fiecare amanunt al rularii pana cand nu vor mai exista clienti de simualt.

* **Scenariul**

Pentru a functiona corect, programul va incepe afisarea simularii doar in momentul in care se acceseaza butonul “Start” din interfata grafica. Este nevoie ca utilizatorul sa introduca datele in campurile prezente in interfata inainte de a accesa acest buton. In partea inferioara a ferestrei se va face afisarea simularii, afisandu-se in timp real lista cu clientii ce asteapta sa fie procesati, precum si modul in care sunt procesati in cele Q cozi. Se va observa ca aplicatia se opreste atunci cand se ajunge la timpul indicat de variabila simulation period sau cand in cele Q cozi nu mai sunt clienti de procesat. La timpul 0, se vor putea observa cel mai bine cei N clienti generate, care la momentul respectiv asteapta sa fie procesati.

* **Cazuri de utilizare**

La rularea acestei aplicatii se va deschide o intefata grafica ce este compusa din sapte JTextField, doua JButton si un camp JTextArea. Design-ul pentru aceasta interfata este unul simplu si usor de inteles de catre orice utilizator. Folosim variabilele JTextField pentru a introduce datele necesare simularii. Aceste date vor fi preluate si modelate in conformitatea cu cerintele problemei,. Cele doua butone sunt folosite pentru a pornii simularea scenariului impus de problema sau pentru a reseta campurile si a permite pornirea unei noi simualri, cu date diferite. Fiecare buton are un nume sugectiv ceea ce va facilita munca cu aceasta interfata.

Dupa ce se introduce datele necesare in cele sapte casete prezente in interfata, utilizatorul trebuie sa acceseze butonul de Start pentru a putea pornii afisarea in campul JTextArea. Informatia din campul JTextArea va fi prezentata in grupuri dupa cum urmeaza : pe prima linie se va afisa timpul de simulare ( aceasta apartine intervalului [0, simulation period]), pe a doula linie se vor afisa clientii ce inca asteapta sa fie procesati urmand ca pe urmatoarele Q linii sa se afiseze fiecare coada cu clientii ce sunt procesati la coada respective, sau “closed” in cazul in care coada respectiva nu proceseaza niciun client. Afisarea se va incheia in momentul in care se termina de procesat cei N clienti. In cazul in care cei N clienti au fost procesati mai repede de simulation period simulrea se va incheia inainte de simulation period. In cazul in care se ajunge la simulation period si totusi mai exista clienti de procesat, aplicatia isi va incheia executia, chiar daca nu s-au procesat toti clientii.

# 3. PROIECTARE

Acest proiect respecta arhitecura clasica Model View Controller (MVC), fiind impartiti in mai multe pachete denumite adecvat. In pachetul “strategyPattern” avem clasa “ConcreteStrategyTime”, interfata “Strategy” si enumeratia “SelectionPolicy” , in pachetul “dataModel” se afla clasele “Client”, “Queue”, “Scheduler”, “SimulationManager” si “Main”, iar in pachetul “graphicalUserInterface” avem clasele “View” si “Controller”.

Principala clasa a acestui proiect este clasa “Scheduler” care contine o lista de cozi , strategia pe care o vom aplica in repartizarea clientilor la cozi si numarul maxim de cozi/threads pe care le vom “lansa” la executia programului. Aceasta clasa contine constructorul de clasa “public Scheduler (int maxNoServers, int simulationPeriod)” in care se lanseaza cele “maxNoServers” threads. De asemenea, aceasta clasa contine si urmatoarele metode public void dispatchClient(Client t), public void changeStrategy(SelectionPolicy policy) si public List<Queue> getQueue().

Clasa “Client” implementeaza clasa “Comparable” si contine urmatoarele atribute : idClient, arrivalTime, processingPeriod si totalServiceTime. In acesta clasa se gasesc metode pentru gettere si settere, precum si metoda compareTo(Client o) suprascrisa din interfata “Comparable”.

Clasa “Queue” implementeaza interfata Runnable deoarece prezinta functionalitatea thread-urilor deschide pentru a simula conceptul de Coada. Aceasta clasa contine ca atribute urmatoarele campuri: coada de clienti definita ca BlockingQueue<Client> clients si o variabila waitingPeriod definita ca un AtomicInteger pentru a stoca timpul de asteptare al cozii. Metodele cele mai importante descrise in aceasta clasa sunt : metoda run() suprascrisa din interfata Runnable si settere si gettere pentru campurile mentionate.

Clasa “SimulationManager” implementeaza interfata Runnable. Aceasta clasa va porni thread-ul principal al programului ce va rula in paralele cu celel Q threads pentru cozi. Clasa are urmatoarele atribute : simulationPeriod, maxServiceTime, minServiceTime, maxArrivalTime, minArrivalTime, noOfClients, maxNoOfQueues, Scheduler scheduler, lista de clienti generati aleator percum si un View view. Printre metodele acestei clase amintim : metoda run() suprascrisa din interfata Runnable, generateNRandomClients(), printInterface(List<Queue> queues, int time).

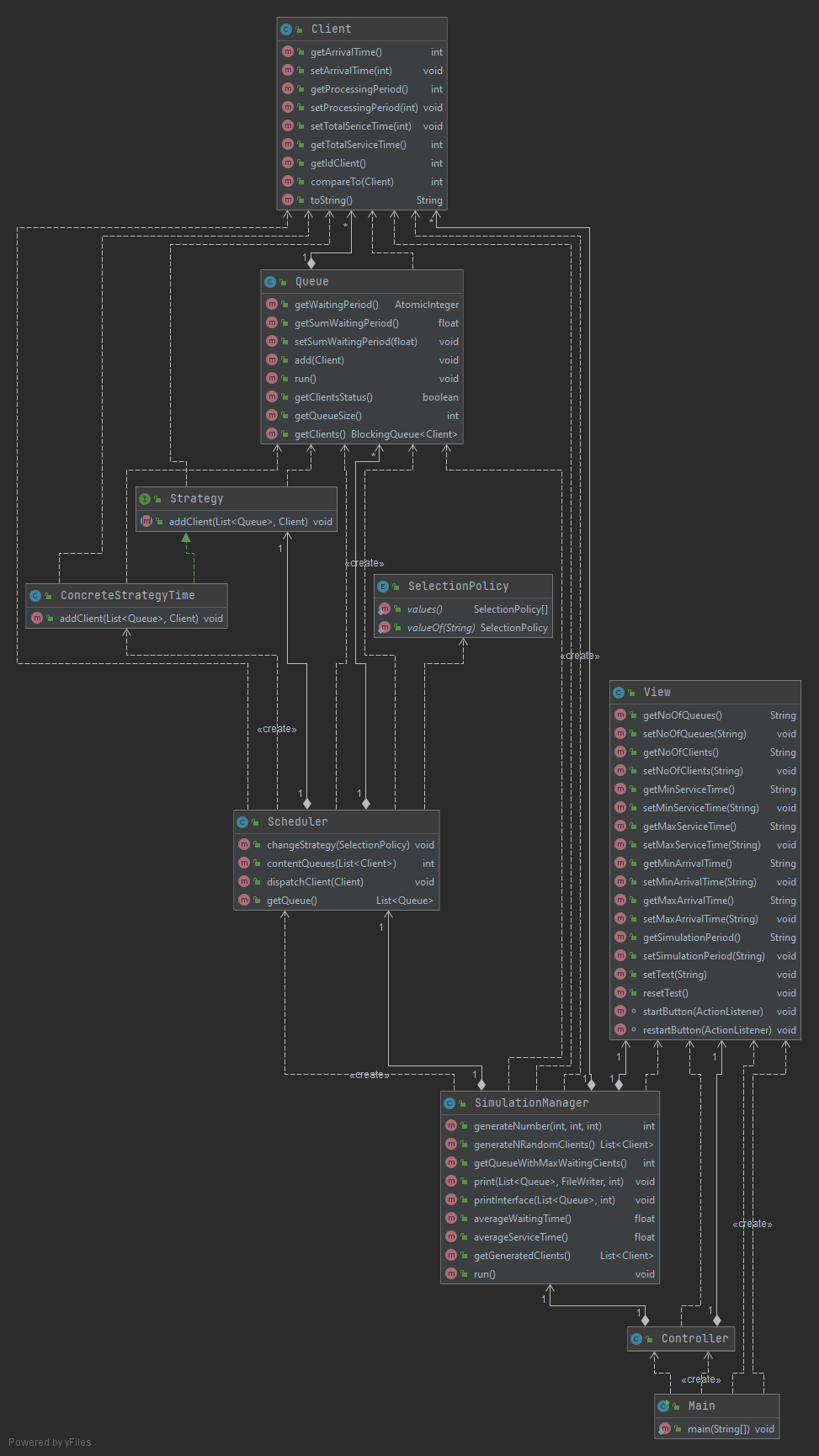
Interfata “Strategy” contine metoda addClient(List<Queue> queues, Client t).

Clasa “ConcreteStrategyTime” implementeaza interfata Strategy prezentata anterior. Aceasta clasa contine un constructor gol si suprascrierea metodei addClient(List<Queue> queues, Client t) in care se repartizeaza un client la una din cele Q cozi respectand cerintele problemei.

Clasa View contine elemente de structura necesare pentru interfata grafica a proiectului. Aceasta face interactiunea cu aplicatia mai practica si mai rentabila pentru utilizator. In cadrul acestei clase am utilizat elemente precum : JTextField( utilizat pentru prelucrarea textului), JButton, JTextArea, JPanel( intermediar). Tot in aceasta clasa decidem modul in care aranjam aceste structuri in interfata finala.

Clasa Controller este clasa in care se traduce interactiunea utlizatorului cu interfata grafica in actiuni ce vor si executate de catre partea de Model (clasa “SimulationManager”).

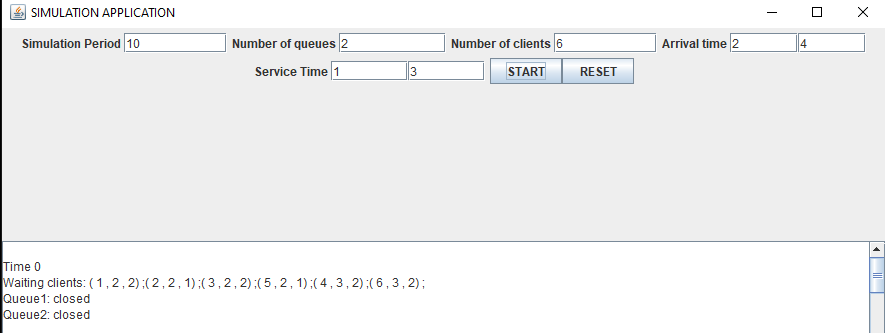
Clasa Main este cea in care se leaga toate componentele mentionate mai sus, permitandu-ne sa rulam aplicatia.



# 4. IMPLEMENTARE

* Clasa “Client” are trei atribute : atributul idClient (int), atributul arrivalTime (int) si atributul processingPeriod(int). ). Pentru aceste doua atribute avem implementate metode de set si get cu ajutorul carora setam valoarea unui atribut sau obtinem valoarea stocata in acesta. Tot in aceasta clasa suprascriem metoda compareTo() pentru a putea ordona o lista de clienti crescator in functie de timpul de sosire al fiecarui client.
* Clasa “Queue” are ca atribute o coada de clienti numita “clients” si atributul waitingPeriod (AtomicInteger). In aceasta clasa am suprascris metoda “run()” mostenita din interfata Runnable si am implementat metode pentru a facilita munca cu aceste cozi.
* Metoda “run()” este executata de catre thread-urile create astfel : Thread thread = new Thread(queue). Aceasta metodat are ca rol procesarea clientilor aflati la coada. Cat timp avem clienti in coada aceasta metodat va extrage un client din structura de BlockingQueue clients si va efectua pas cu pas, cu ajutorul unui for pana la this.clients.element().getProcessingPeriod, urmatoarele actiuni : pentru inceput va pune thread-ul sa astepte o secunda apoi va decrementa processingPeriod pentru clientul curent , cat si atributul waiting al cozii curente. La final, dupa ce cele this.clients.element().getProcessingPeriod secunde trec si clientul curent este procesat, se va continua cu urmatorul client din BlockinQueue pana cand coada ajunge sa fie empty.
* Metoda “getClientsStatus()” va returna “true” daca lista de clientii din coada curenta este goala si “false” in cazul in care in coada inca mai exista clienti de procesat.
* Metoda “getQueueSize()” returneaza size-ul listei de clienti pentru coada curenta.
* Clasa “Scheduler” are ca atribute o lista de cozi numita “queues”, o strategie “strategy” si atributele “maxNoServers” (int), “simulationPeriod” (int). In aceasta clasa avem implementate urmatoarele metode:
* Constructorul calsei “Scheduler” va initializa campurile “maxNoServers” si “simulationPeriod”, apoi va crea maxNoServers cozi, queue, pe care le va adauga la lista de cozi. Dupa ce sunt create, constructorul va crea si porni cate un Thread pentru fiecare structura de Queue din queues. Astfel, cele maxNoServers vor procesa in paralele clientii.
* Metoda “contentQueues(List<Client> generatedClients)” verifica daca in lista de cozi mai exista sau nu clienti de procesat. Metoda va parcurge queues si va verifica continutul listei de clienti pentru fiecare coada. Daca in cozi se mai proceseaza clienti atunci metoda va returna valoarea 0, altfel va semnala ca s-au procesat toti clientii din cozi si va returna 1. Este important de mentionat ca pentru a functiona corect, aceasta metoda verifica prima data ca nu mai exista clienti in asteptare in lista generatedClients.
* Metoda “dispatchClient(Client t)” are ca scop repartizarea clientului t la una din cele maxNoServers cozi. Pentru a efectua aceasta opertie, programul apeleaza metoda addClient din interfata “Strategy”.
* Clasa “ConcreteStrategyTime” implementeaza interfata Strategy. Aceasta clasa contine metoda addClient(List<Queue> queues, Client t) suprascrisa din interfara. In aceasta metoda trebuie sa decidem la ce coada din lista de queues este cel mai potrivit sa adaugam un client. Metoda va parcurge lista de queues si va afla timpul de procesare minim. In functie de acest timp vom crea o lista cu coziile ce au timpul de asteptare egal cu timpul minim de asteptare aflat anterior si vom afla care este numarul minim de clienti ce sunt procesati la aceste cozi. Dupa ce aflam aceste doua informatii vom merge si vom cauta in queues prima coada care respecta cele doua cerinte : au timpul de asteptare egal cu timpul minim de asteptare si numarul de clienti din coada este egal cu numarul minim de clienti. Daca exista mai multe cozi ce respecte aceste doua cerinte programul va adauga clientul in prima coada pe care o gaseste. Tot la acest pas timpul de asteptare al cozii la care vom adauga clientul va creste cu timpul de servire al clientului.
* Clasa “SimulationManager” contine o lista de clienti a caror informatii sunt generate aleator, o structura de tip Scheduler “scheduler”, o structura de tip View numita “view” si urmatoarele atribute : “minArrivalTime” (int), “maxArrivalTime” (int) , “minServiceTime” (int), “maxServiceTime” (int), “noOfClients” (int), “maxNoOfQueues” (int) si “simulationInterval” (int). Aceasta clasa va descrie functionarea unui Thread principal responsabil cu repartizarea clientilor la cele maxNoOfQueues.
* Metoda suprascrisa “run()” va prelucra o lista de noOfClients numita generatedNClients. Aceasta metoda va rula atata timp cat variabila currentTime este mai mica sau egala cu variabila simulationInterval si mai exista clienti de procesat in cele maxNoOfQueues. Pentru inceput se va verifica daca clientul curent are timpul de sosire egal cu currentTime. Daca da, metoda va repartiza clientul la o coada apeland metoda din clasa Schedulre “dispatchClient(Client t)” urmand ca mai apoi sa elimine clientul din lista de asteptare. Dup ace gata de procesat toti clientii care au timpul de sosire egal cu currentTime, thread-ul va lua o pauza de o secunda si va continua procesarea clientilor pana cand va atinge conditiile de oprire.
* Clasa “View” genereaza partea de interfata grafica ce se poate vizualiza la rularea aplicatiei. In aceasta clasa se descrie modul de aranjare al componentelor in interfata grafica si adaugam ActioListener() pentru butoane.
* Clasa “Controller” se coupa cu implementarea “evenimentelor” in momentul in care se efectueaza oa actiune asupra unei componente din interfata ce genereaza un ActionListener().

# 5. REZULTATE



Vom parcurge un exemplu folosindu-ne de urmatoarele date:

Simulation period = 10

Q = 2, N = 6

Arrival Time ⸦ [2,4] ; Service Time ⸦ [1 , 3]

Vom folosi urmatoarea lista de clienti : (1, 2, 2,) ; (2, 2, 1); (3, 2, 2); (5, 2 , 1) ; (4, 3, 2); (6, 3,2);

Time 2 – la acest timp se vor adauga la cozi clientii : (1, 2, 2); (2, 2, 1) ; (3, 2, 2); (5, 2, 1)

Waiting Clients : (4, 3, 2) ; (6, 3, 2)

* In acest moment ambele cozi sunt inchide astfel ca prima data se va adauga clientul (1, 2, 2) la coada 1 urmand sa se adauge clientul (2, 2, 1) la coada 2. Dupa ce in ambele cozi se verifica la care dintre cele doua timpul de asteptare este mai scurt => se va adauga clientul (3, 2, 2) la coada 2 ( timp de asteptare o secunda). Se realizeaza aceeasi operatie si pentru clientul (5, 2, 1) ce va fi adaugat la coada 1 ( timp de astepatre 2).

Queue1 : (1, 2, 2) ; (5 , 2 , 1);

Queue2 : (2, 2, 1) ; (3, 2, 2);

Timp3 – la acest timp se vor adauga la cozi clientii (4, 3, 2) ; (6, 3, 2);

Waiting Clients :

* In acest moment timpul de asteptare la cele 2 cozi este egal asa ca (4, 3, 2) se va adauga la coada 1, urmand ca mai apoi (6, 3, 2) sa fie adaugat la coada 2 . Intre timp clientii deja aflati la coada sunt procesati.

Queue1 : (1, 2,1) ; (5, 2, 1); (4, 3, 2);

Queue2 : (3, 2, 2) ; (6, 3, 2);

Time 4 – deoarece la acest moment nu mai exista clienti in lista de asteptare vor fi procesati cei ce se afla la cozi.

Waiting Clients:

Queue1 : (5, 2, 1); (4, 3, 2);

Queue2 : (3, 2, 1) ;(6, 3, 2);

Time 5

Waiting Clients:

Queue1 : (4, 3, 2);

Queue2 : (6, 3, 2);

Time 6

Waiting Clients:

Queue1 : (4, 3, 1);

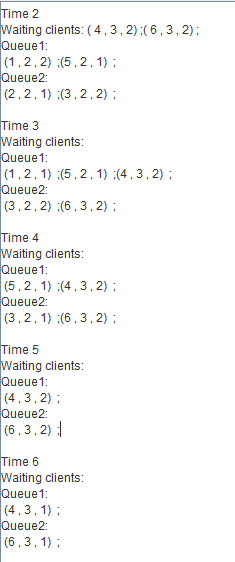
Queue2 : (6, 3, 1);

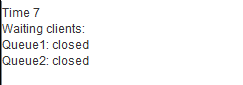
Time 7

Waiting Clients:

Queue1 : closed

Queue2 : closed



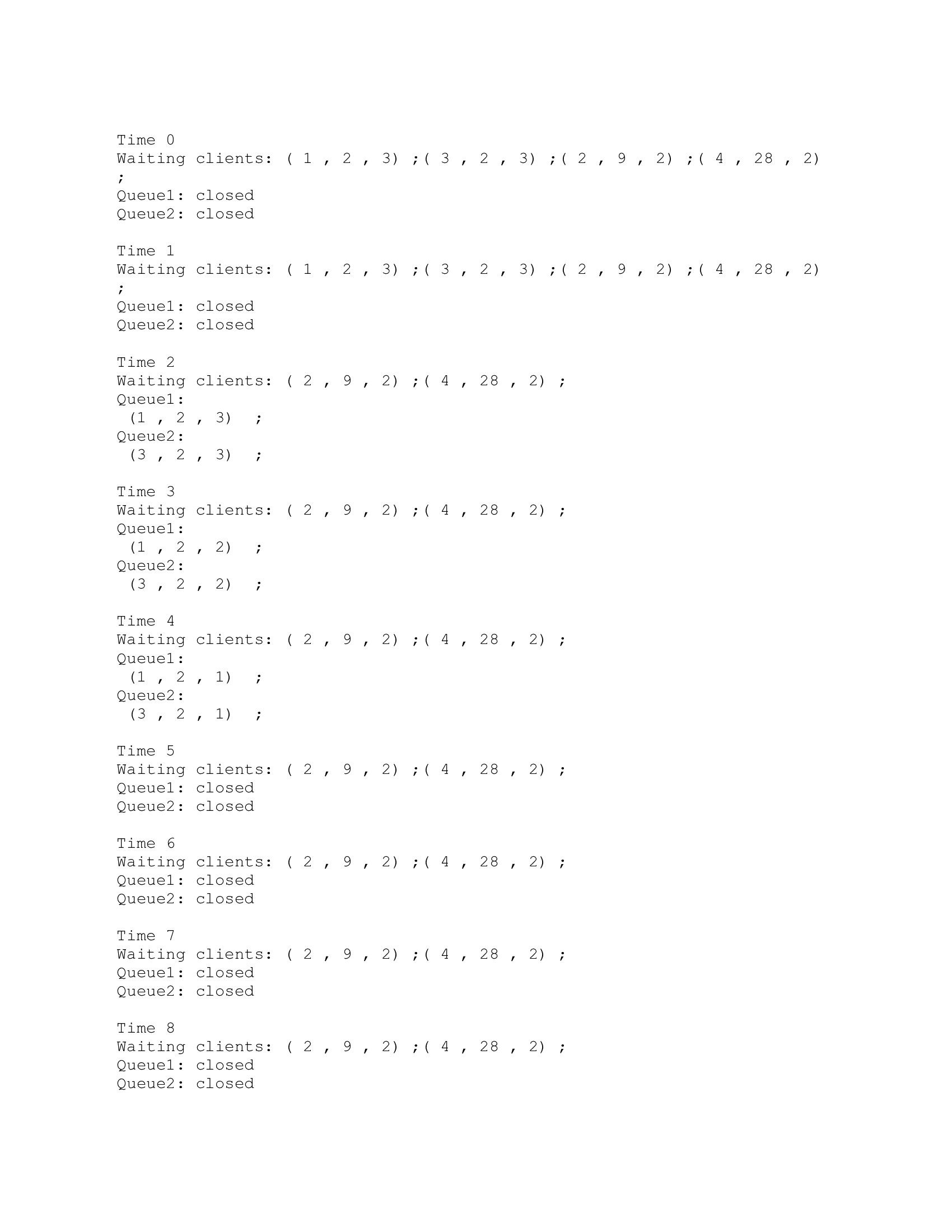


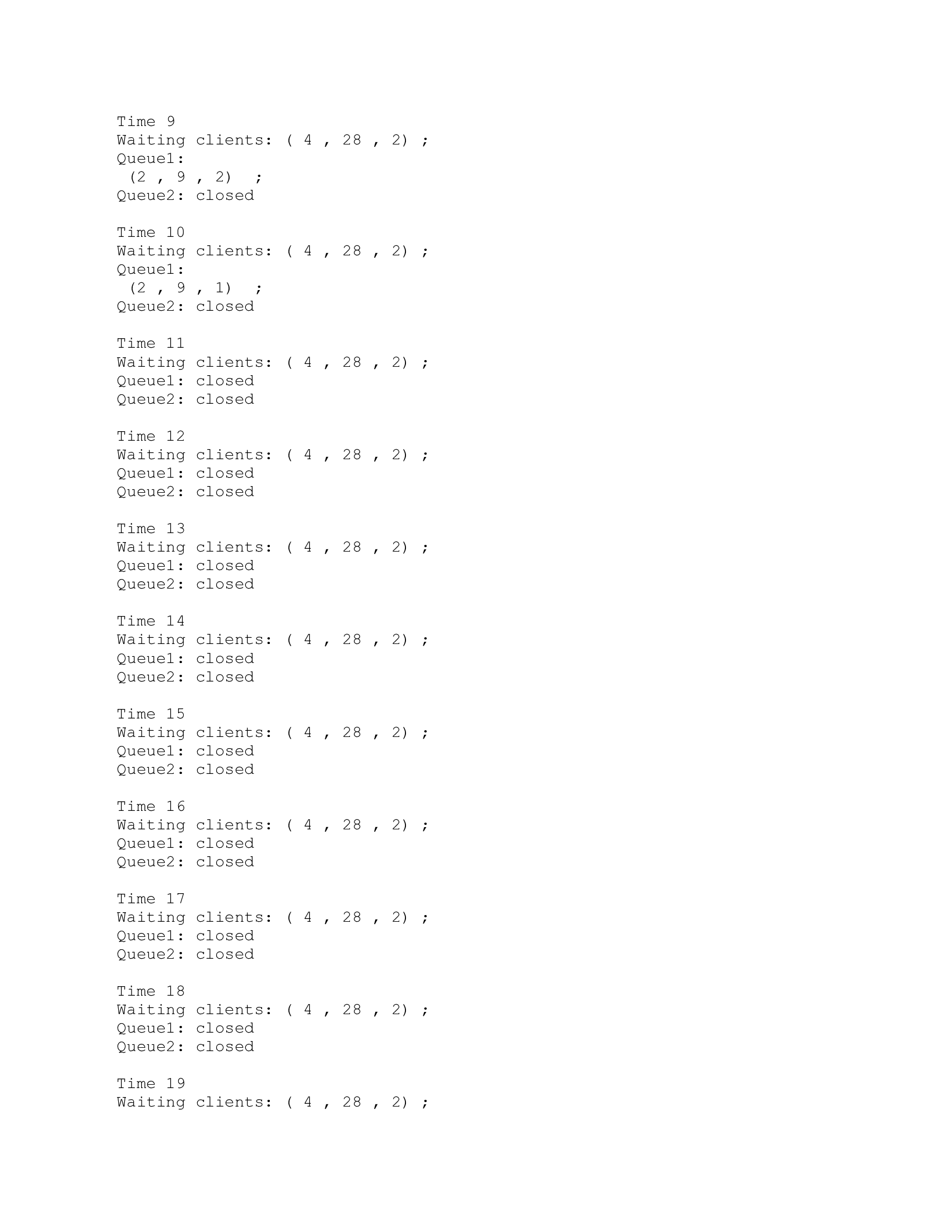
Aplicatia va afisa si :

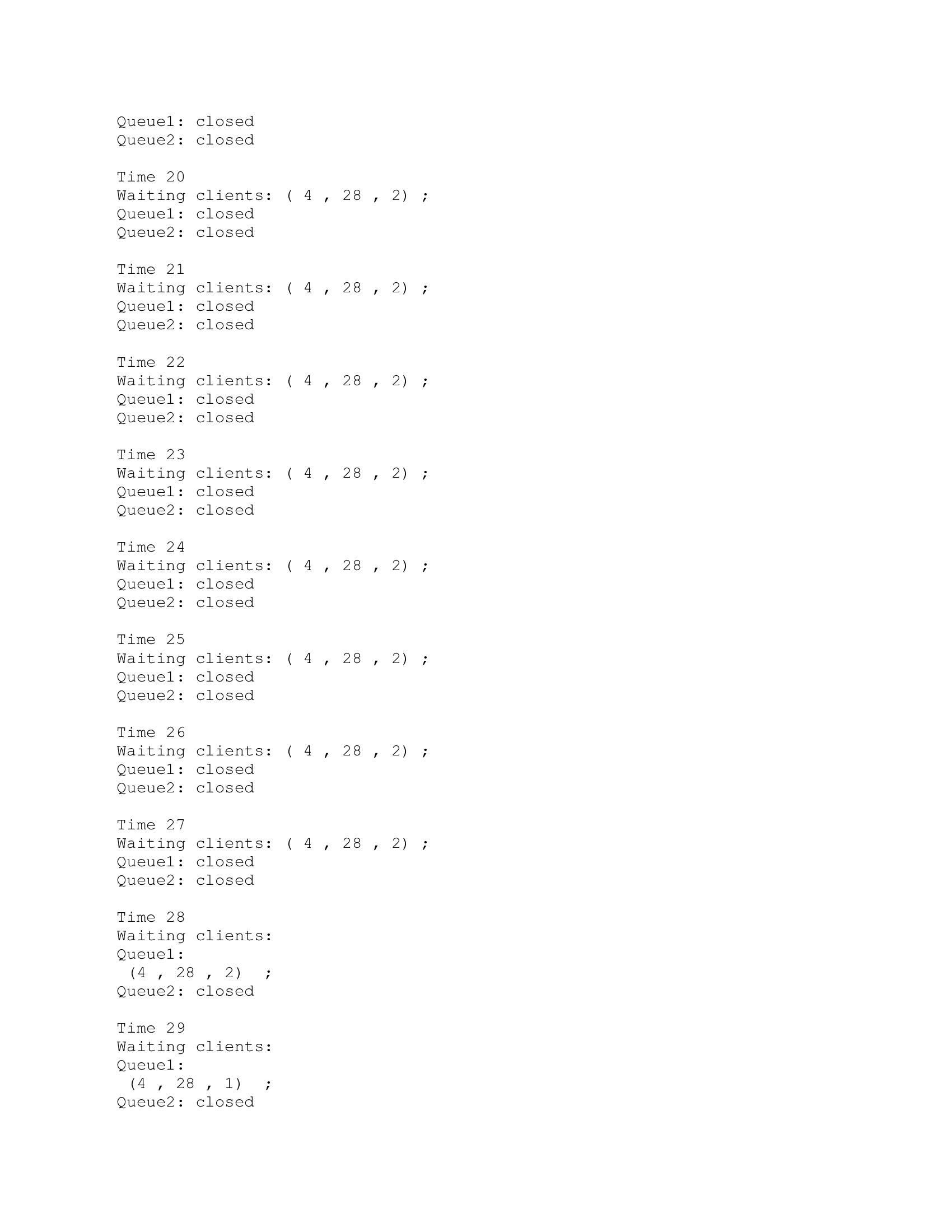


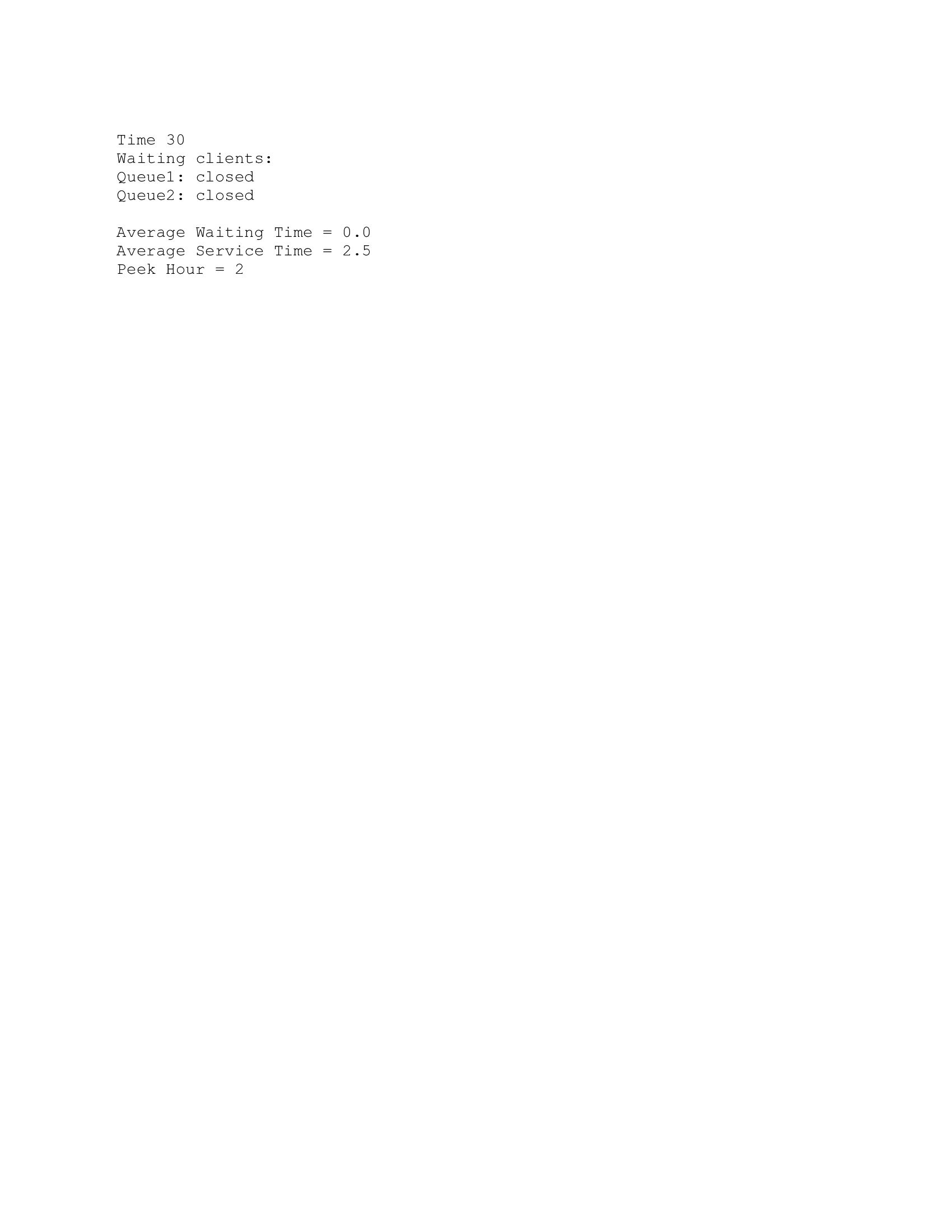
**TESTARE:**

* Test 1:

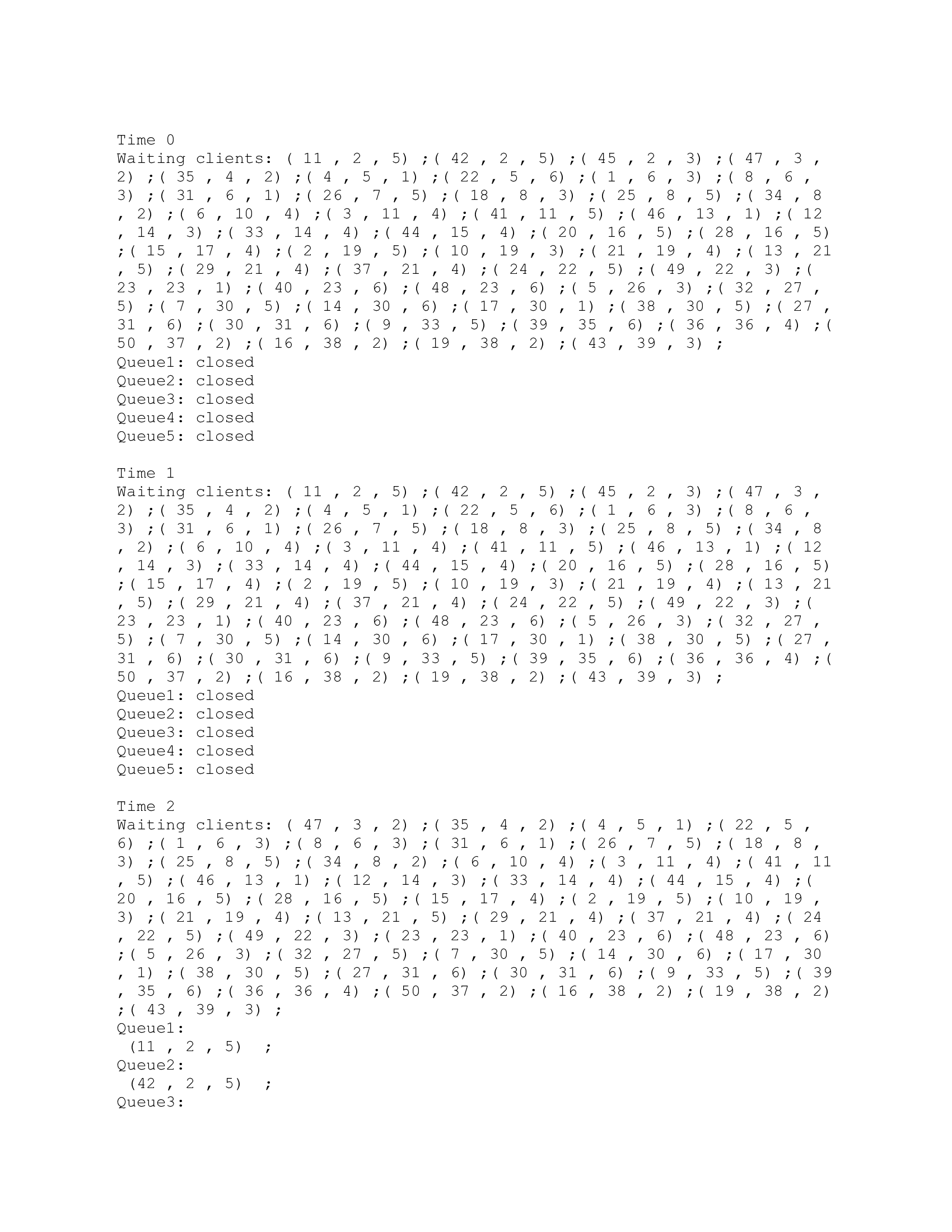


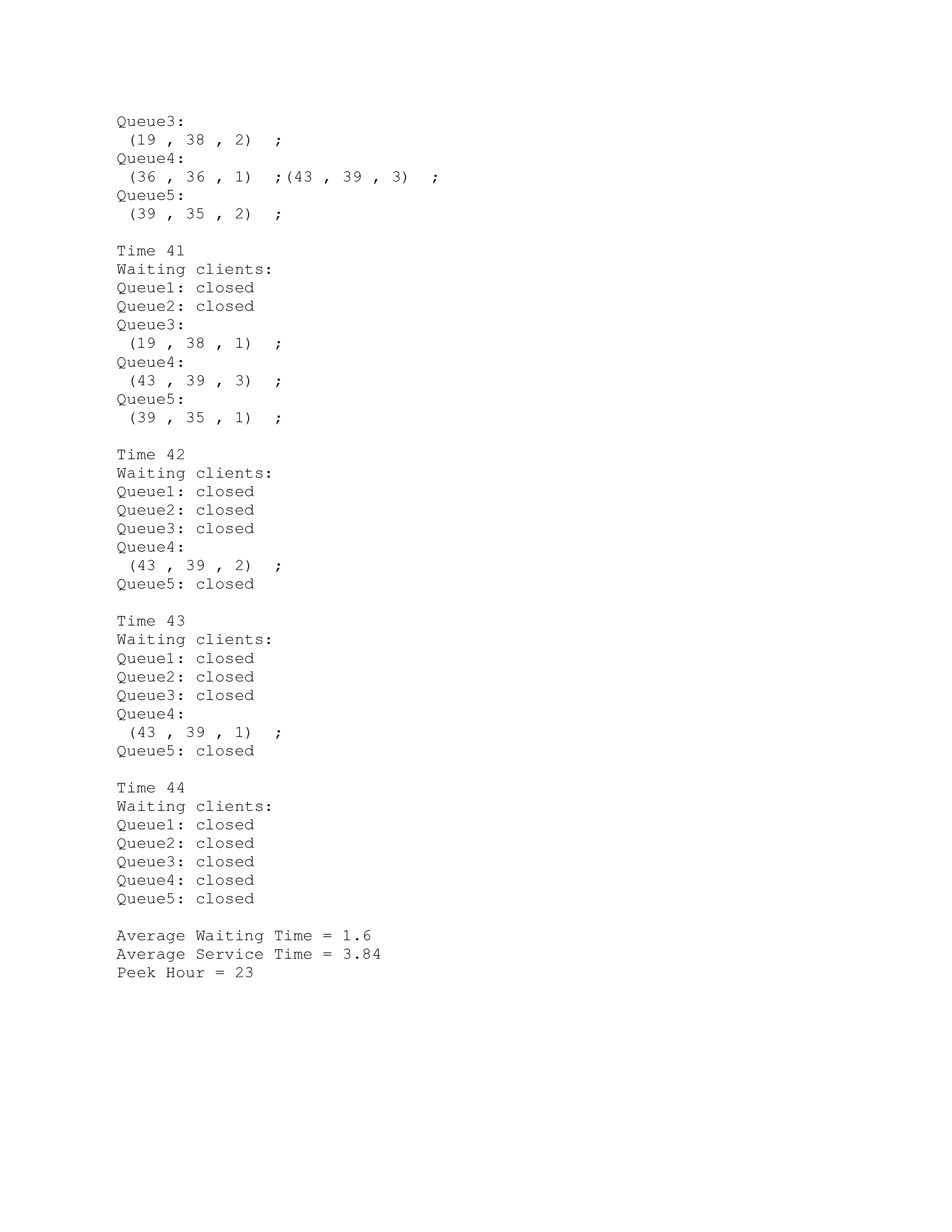
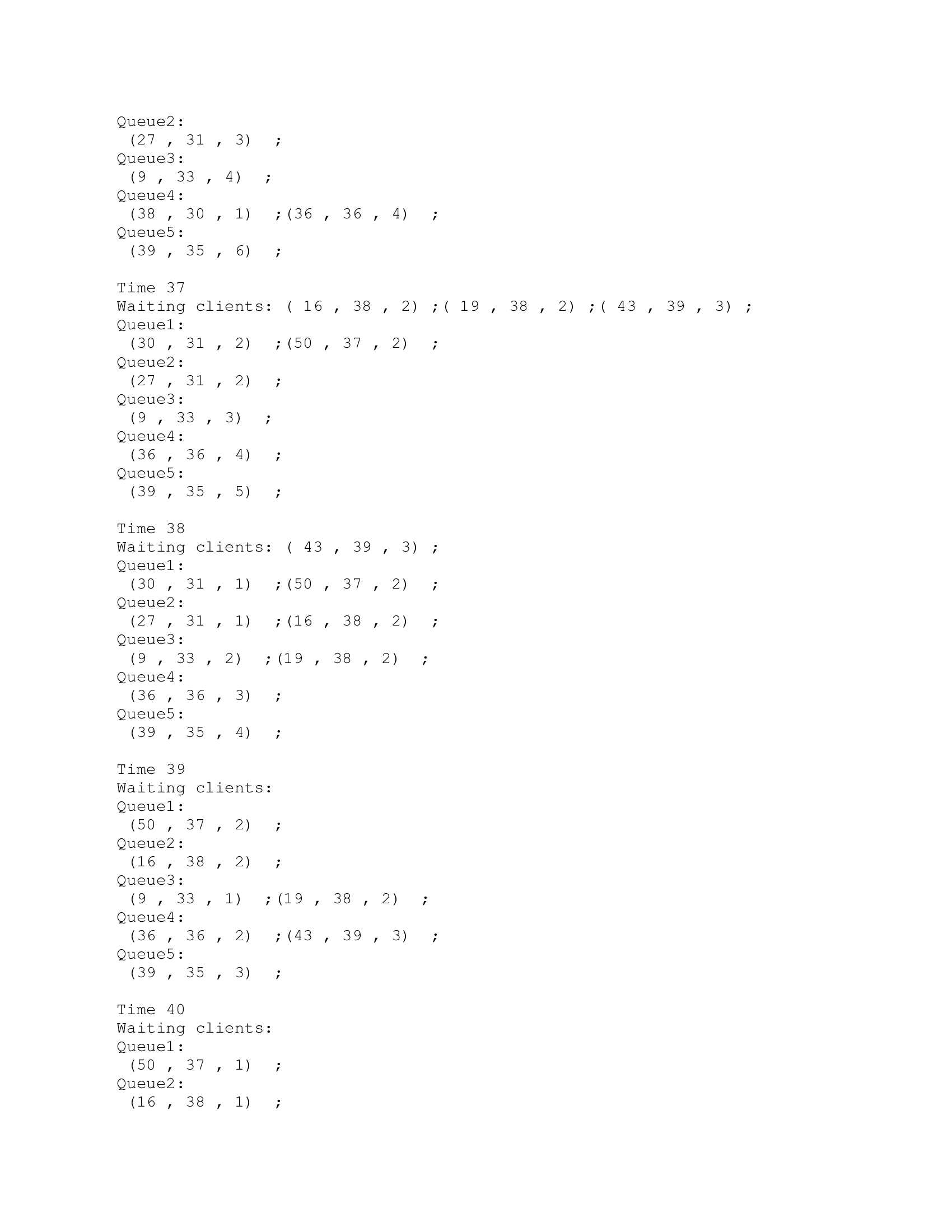
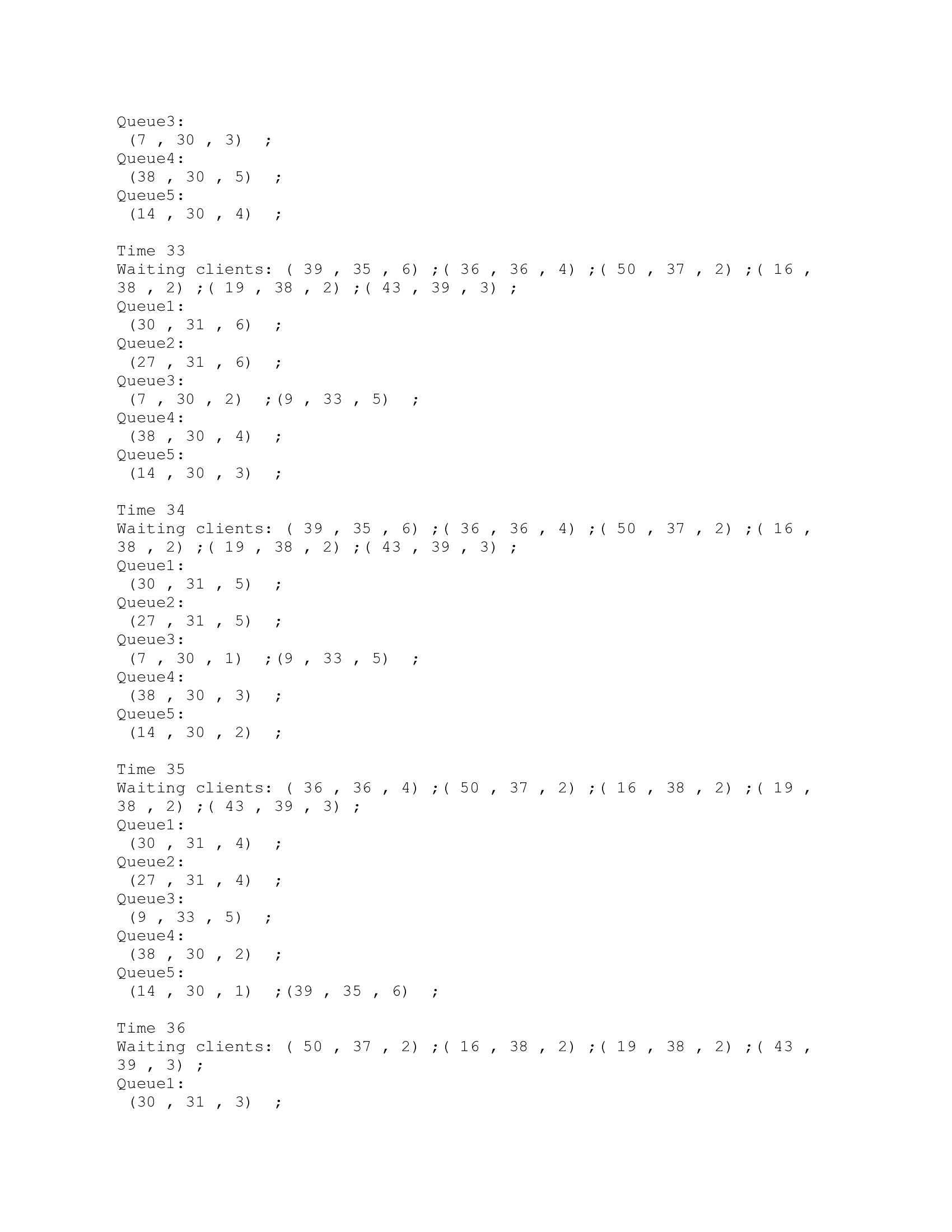
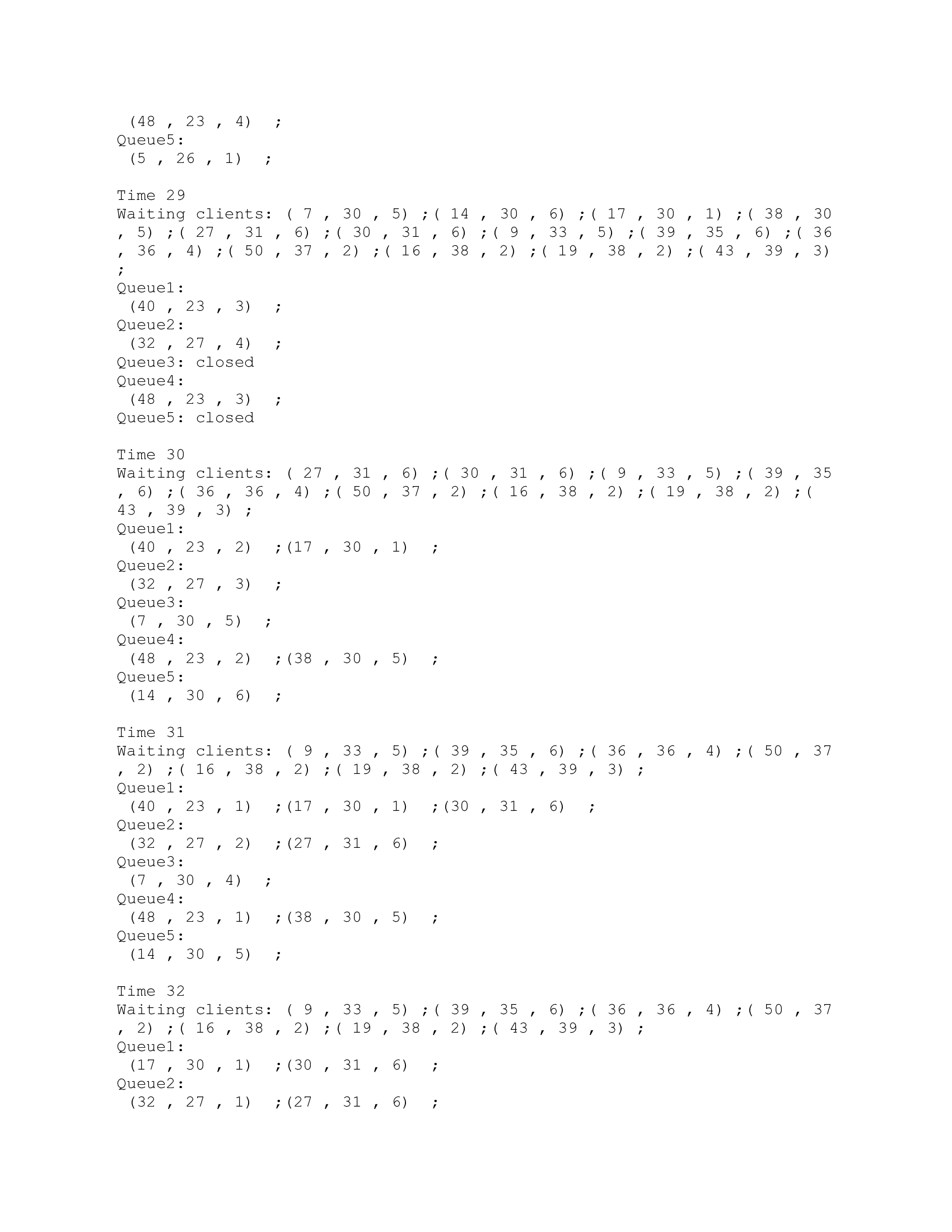
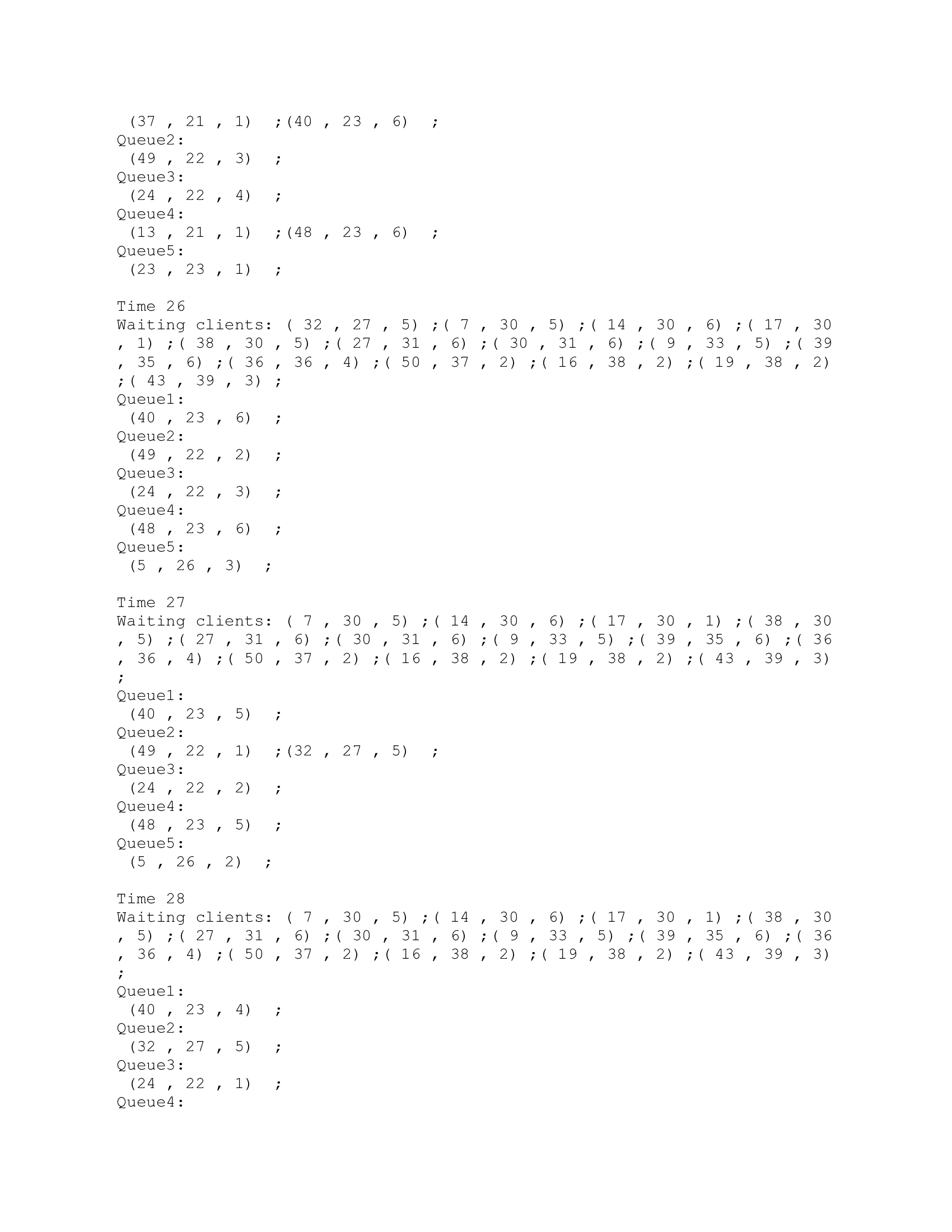
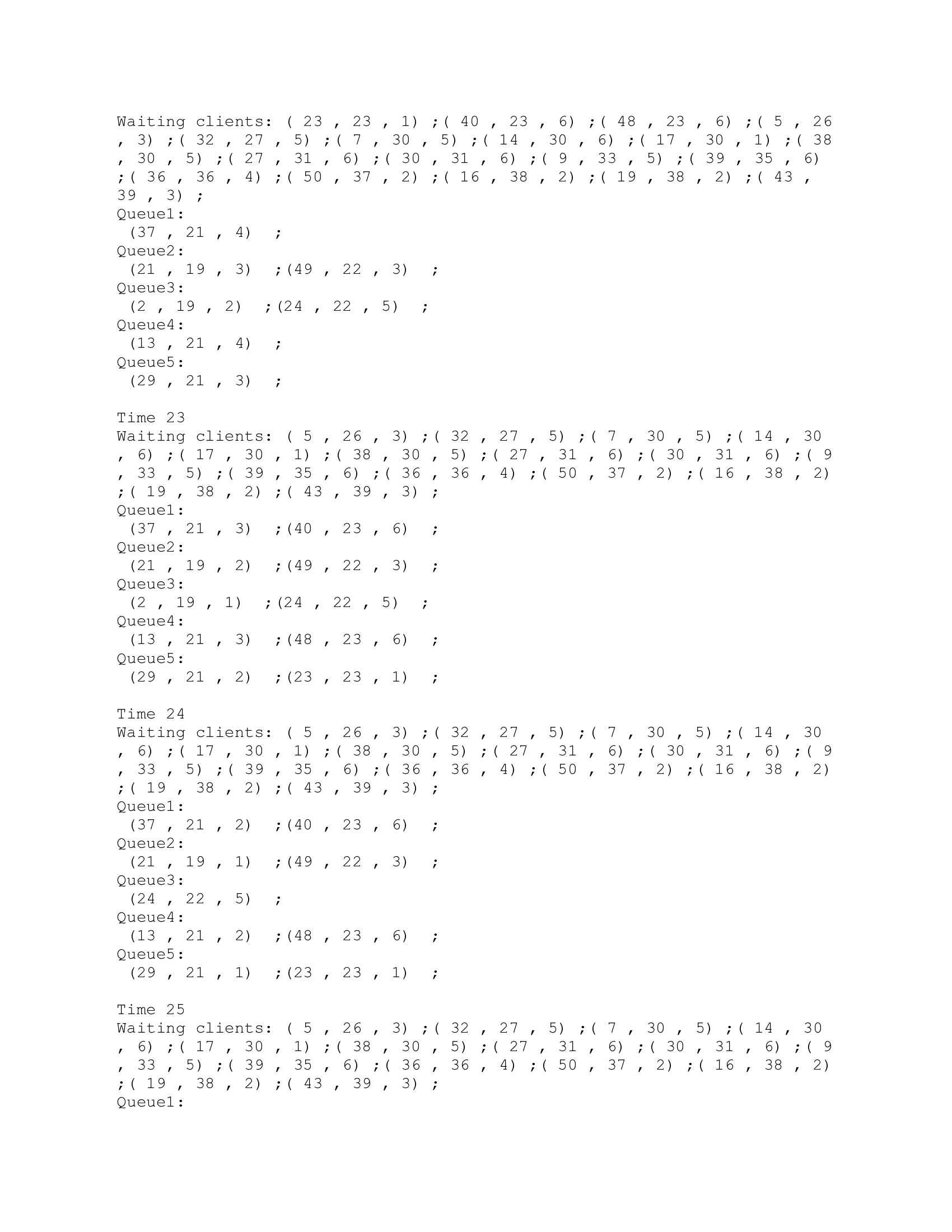
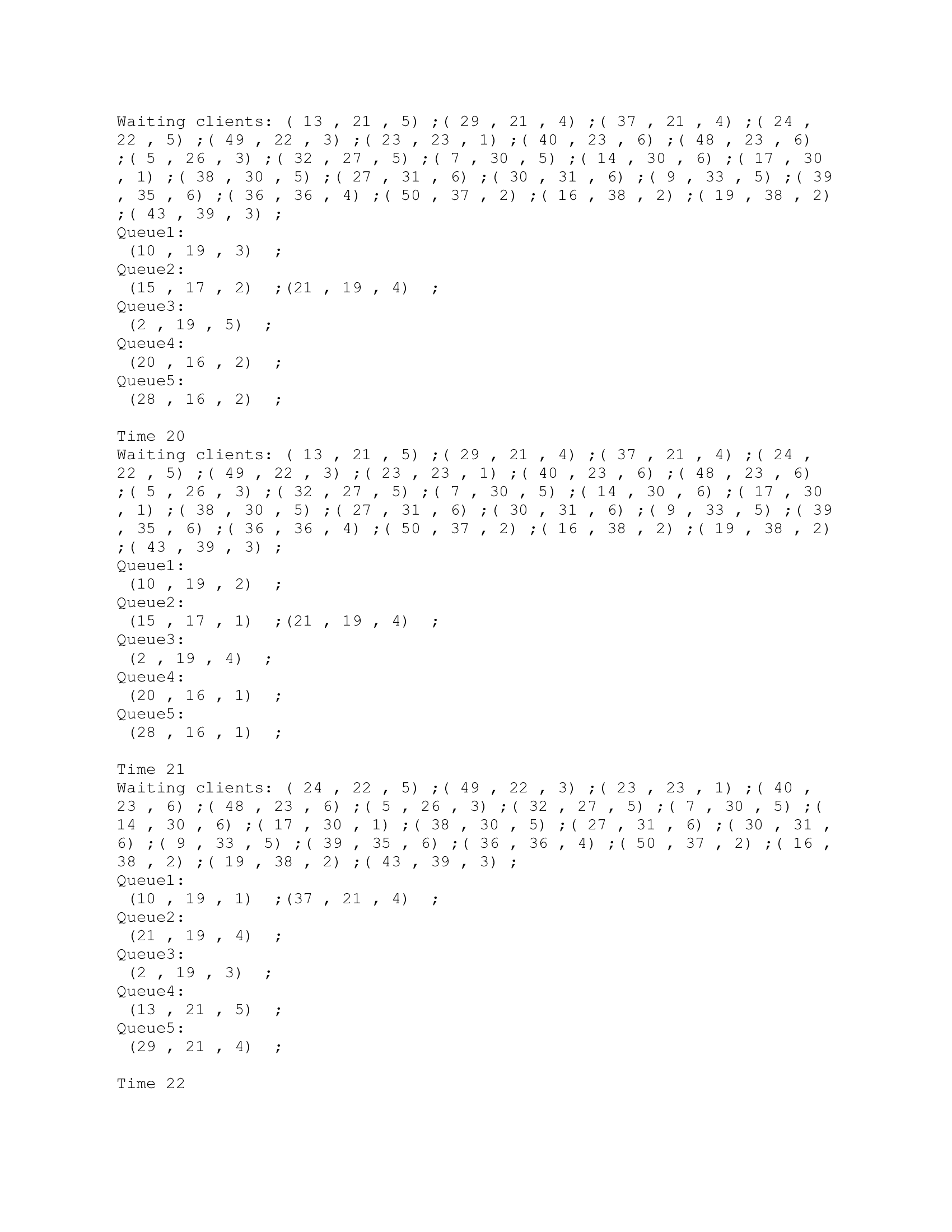
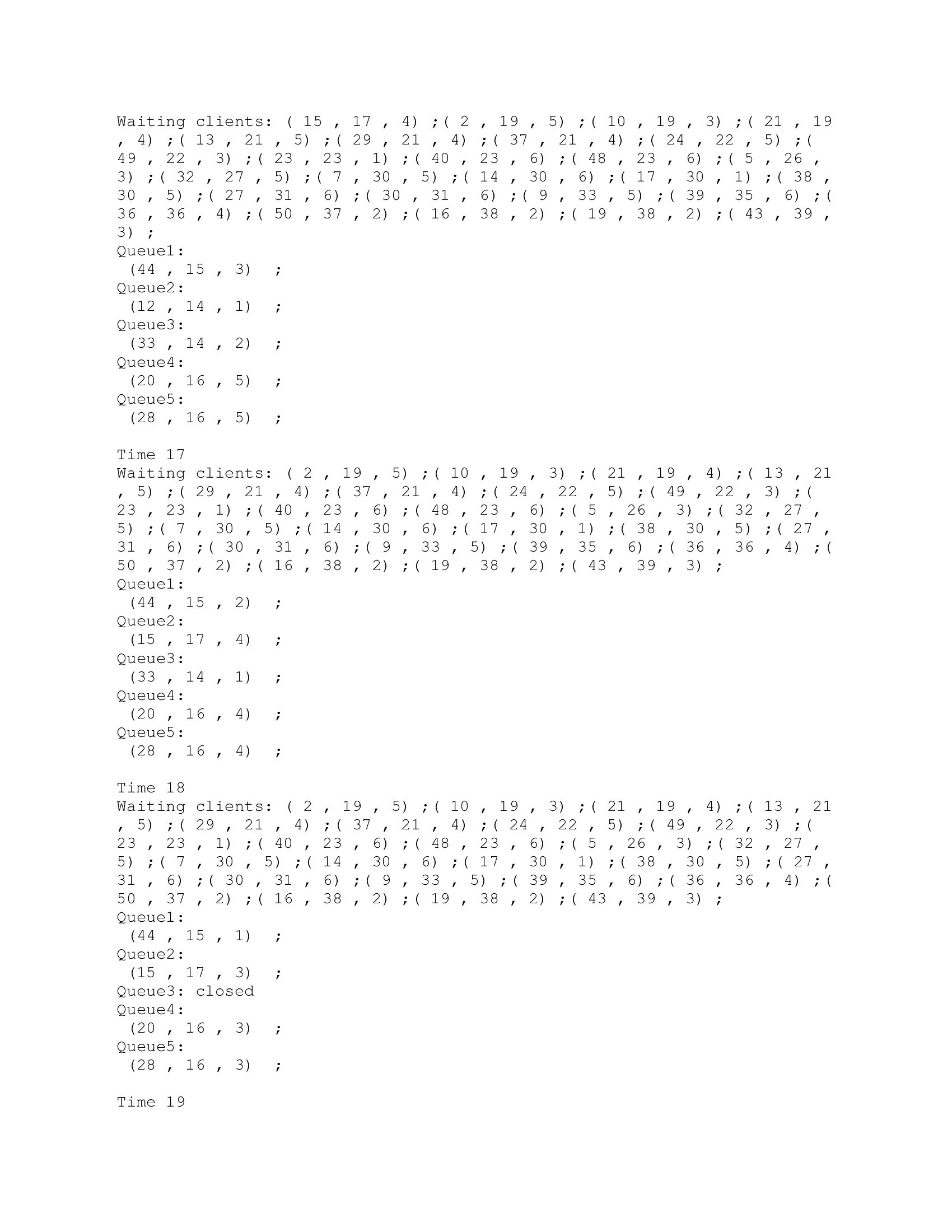
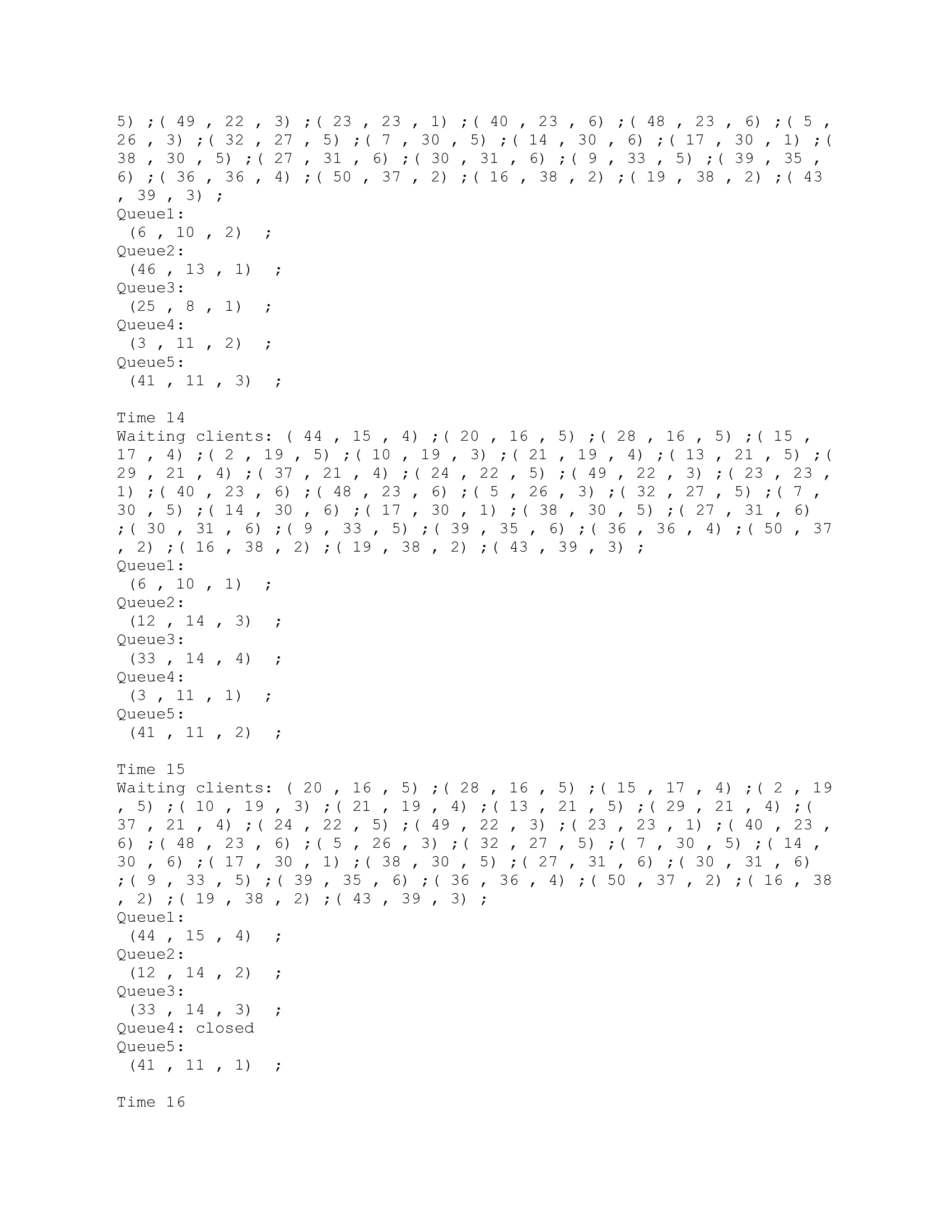
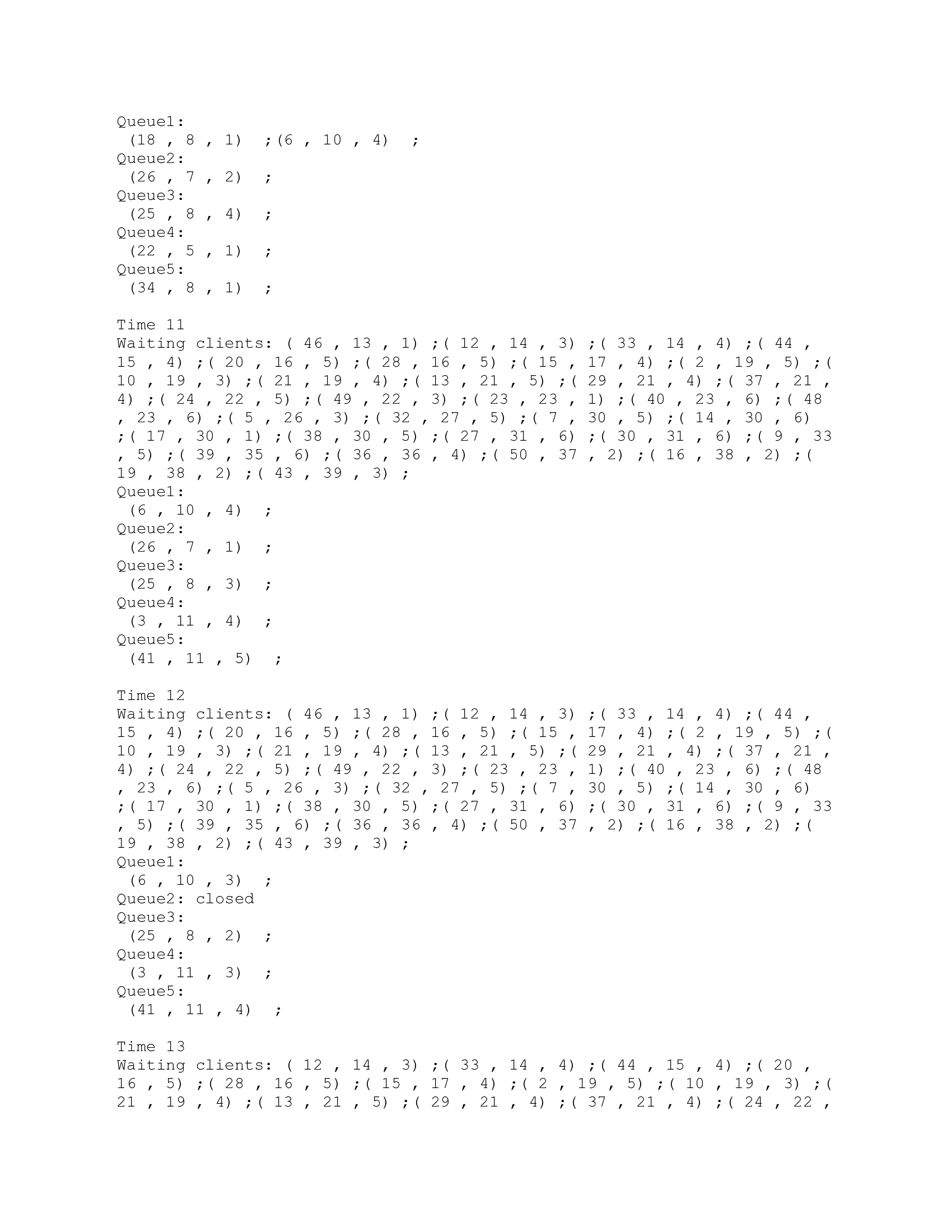
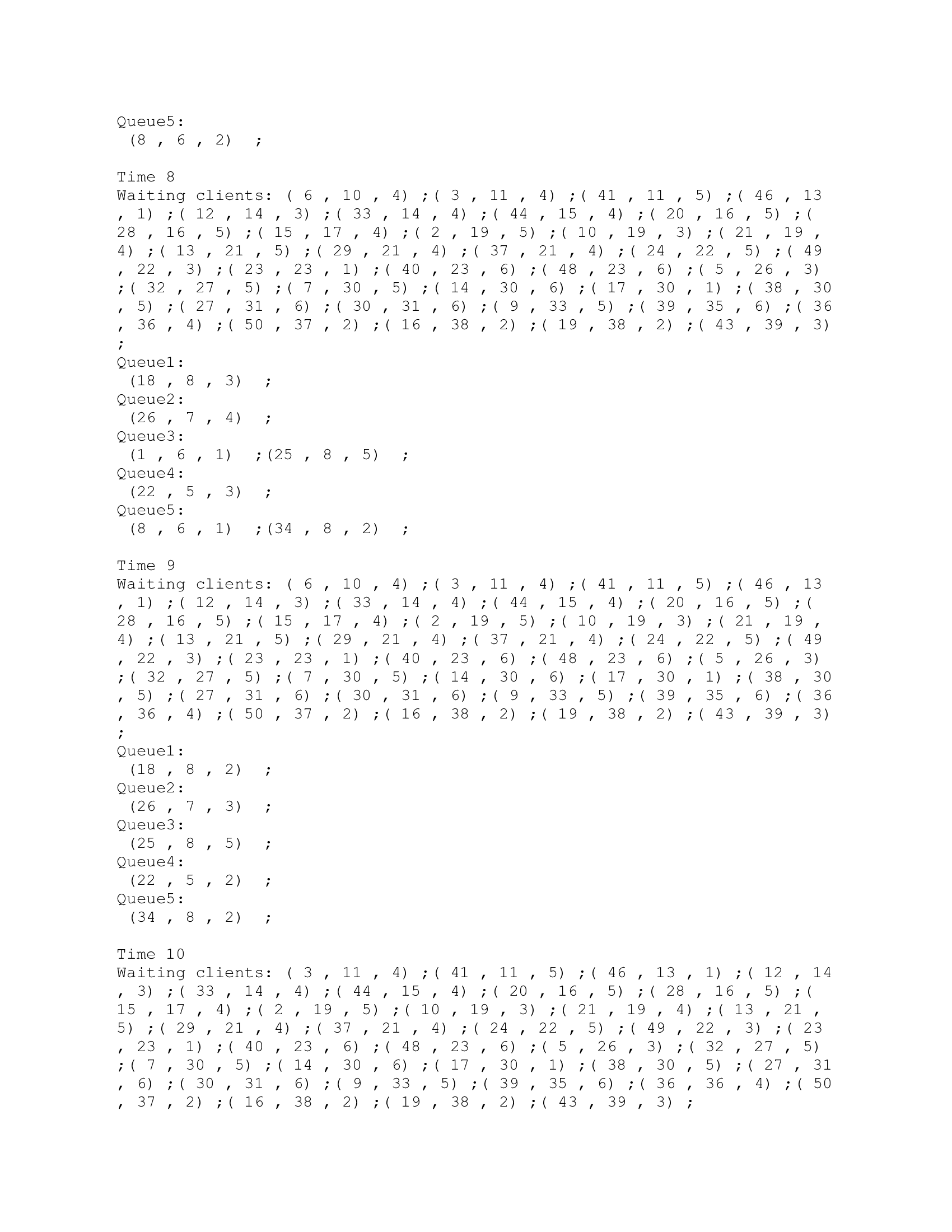
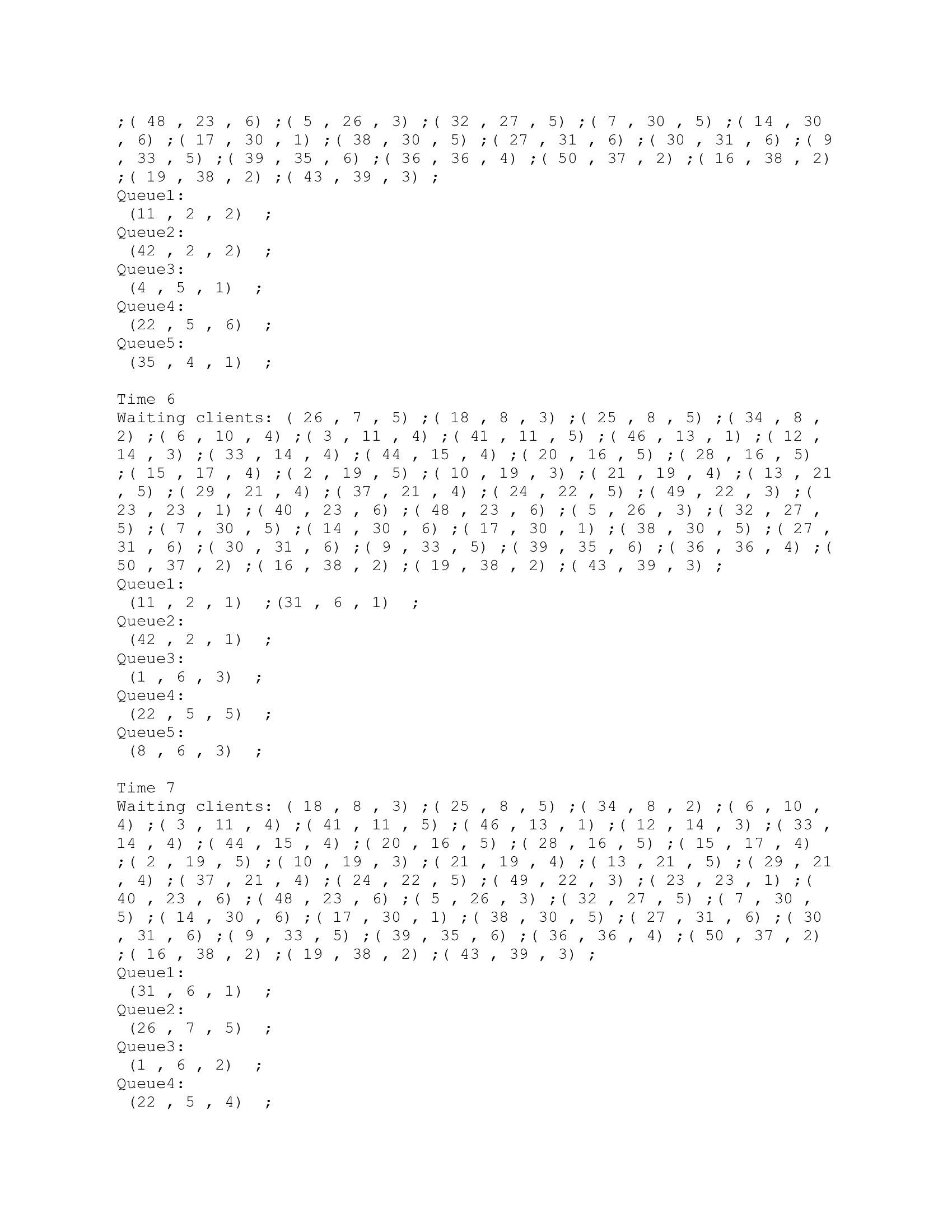
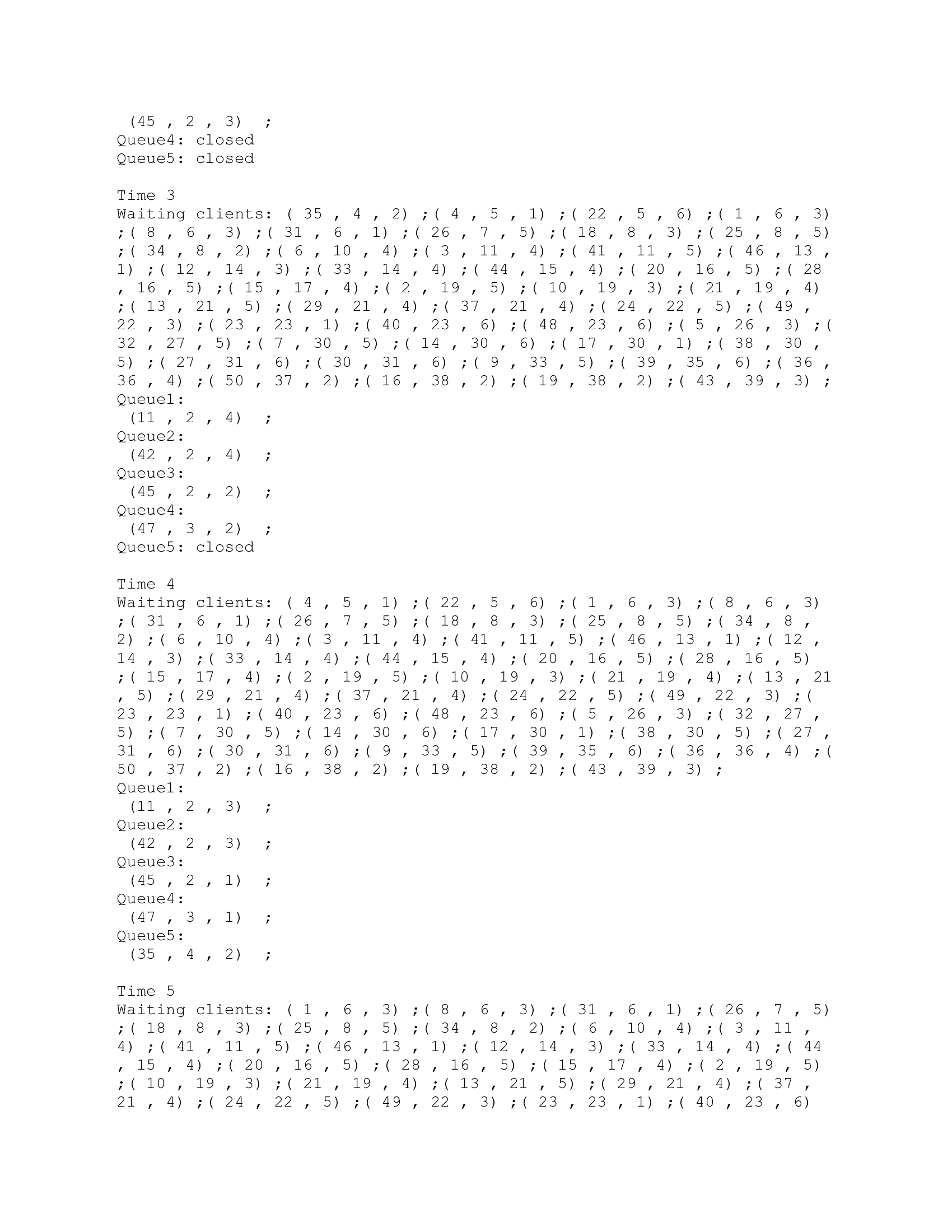
****

****

****

* Test 2





# 6. CONCLUZII

Pentru mine aceasta tema a ridicat anumite dificultati in ceea ce priveste modul in care sunt procesate Thread-urile. Pe parcurs am ajuns sa inteleg conceptul de Thread si utilitatea acestuia. In ceea ce priveste simularea scenariului impus in cerinta nu am intampinat dificultati in intelegerea cerintei, sper, reusind sa ajung la o forma finala convenabila care sa imi genereze solutii corecte. De asemenea, si am incercat sa implementez aceasta aplicatie respectand paradigmele programarii orientate pe obiecte .

# 7. BIBLIOGRAFIE

* <https://www.geeksforgeeks.org/>
* <https://stackoverflow.com/>