Logic:

1. Each site will maintain a count of the number of times it has entered the CS. The initial value will be 0.

2. Every time a request is sent out, the request gets inserted to the requestQ of the site by comparing the hit count of the request with the existing requests in the queue. The requestQ is ordered in the increasing order of hit count.

eg: If **Process_i** has a hit count greater than the hit count of **Process_j** then **Process_j** will be ordered before **Process_i**

This ensures that the process don't starve.

Data Structures

`request_queuei` – Contains list of outstanding CS requests at Site P_i

Each request queue entry `rq_entry` is a tuple of type `(pid, ts, hits)`

1. Initialization:

P_i = Process Id/Site Id

R_i = Request Set for Site with Site_Id P_i

p_i.hit = 0

2. Requesting CS

i) Build request (`req`):

```
req.pid = pid
req.ts = ts
req.hits = pi.hits // number of times the site has entered CS
```

ii) a) Send `REQUEST(TSi,req)` to all sites in Ri

c) Add `req` to queue as follows **// [improvement]**

```
// eg: q = {0,1,2,3} // showing only hits from the tuple
// req.hits = 2
// index k found = 3
// after insert q = {0,1,2,2,3}
i. find index k, such that for each m from {0 to sizeof(q)}
   req.hits > request_queue[m].hits
ii.Insert req at k
```

3) Sj Receiving Request `REQUEST(TSi,req)`

I) Check if any request exists in `request_queuej` such that,

```
 if(entry[m].Pid=Sj & entry[m].ts > TSi)
      send reply to Pi
else
      Send REPLY(TSj,Pi) //Send reply to Request sender
```

II) Add `req` to queue as follows

```
   // eg: q = {0,1,2,3} // showing only hits from the tuple
   // req.hits = 2
   // index k found = 3
   // after insert q = {0,1,2,2,3}
   i. find index k, such that for each m from {0 to sizeof(q)}
      req.hits > request_queue[m].hits
   ii.Insert req at k
```

3. Executing the critical section.

Site Si enters the CS when if the following conditions hold:

I. Si has received a message with timestamp larger than (tsi , i) from all other sites.
II. S_i's request is at the top of `request_queuei`.

4. Releasing the critical section.

I) Site Si, upon exiting the CS, removes its' request from the top of its' `request_queuei`, `increments the hit count` and sends a timestamped RELEASE message to all the sites in its `Ri`.

II) When a site Sj receives a RELEASE message from site S_i , it removes S_i 's request from its request queue.

When a site removes a request from its request queue, its own request may become the top of the queue, enabling it to enter the CS. The algorithm executes CS requests in the increasing order of timestamps.