# Concurrent Programming Homework 2

1. The following variables are added:
    1. numberOfInserts -> The number of Inserts performed
    2. numberOfRemoves -> The number of Removes performed
    3. numberOfSearches -> The number of Searches performed

    All the variables are initialized to 0.

2. Invariant that captures the synchronization requirement:

    Let numberOfInserts = nI, numberOfRemoves = nR, numberOfSearches = nS;
    (nI >= 0) ^ (nR >= 0) ^ (nS >= 0) ^ (nI > 0 ^ nR == 0) ^ (nI == 0 ^ nR > 0)

    i. All the variables should be greater than or equal to zero - (nI >= 0) ^ (nR >= 0) ^ (nS >= 0)
    ii. Insert can happen only when there are no remove operations and vice versa - (nI > 0 ^ nR == 0) ^ (nI == 0 ^ nR > 0)

3. void start_insert( ) {

    < await (numberOfInserts == 0 &&  numberOfRemoves == 0)

        numberOfInserts++; >

    }

    void  end_insert( ) {

    < numberOfInserts--;

    notifyAll( ); >

    }

    void start_search( ) {

    < await (numberOfRemoves == 0)

        numberOfSearches++; >

    }

    void  end_search( ) {

    < numberOfSearches--;

    notifyAll( ); >

    }

```
void start_deletion( ) {

< await (numberOfInserts == 0 && numberOfRemoves == 0 && numberOfSearches == 0)

        numberOfRemoves++; >

}

void end_deletion( ) {

< numberOfRemoves--;

notifyAll( ); >

}
```

4. Yes. The insert and the search methods have conflicting operations and data races. Node "first" is shared by these methods, i.e., while search tries to read the variable, insert tries to update the same, resulting in conflict and data races. An element will be found in the list depends on whether insert occurs first or search. If insert occurs first and later the same element is searched, the element would be found. But if an element is searched first and insert operation for the same element occurs later, the element would not be found.

   No. The nodes created are not safely published. To be safely published, the node "first" should be made volatile.

5. See Implemented Code in JAVA.

6. This class can be used, where initially the data is present and thereafter infrequently modified. Also, when there are more reads (search) than writes (insert).

   PROS:

   i. Search operation is faster since, multiple threads can perform search operations concurrently.

   CONS:

   ii. In case of frequent insert and remove operations, the data is exclusively locked most of the time and thus, there is little increase in concurrency.
   iii. In case of short insert and remove operations, the overheads of the lock implementation can be very high.
   iv. Search operation is faster since, multiple threads can perform search operations concurrently.