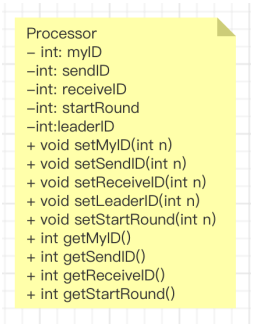
Report of Assignment1

3.1 Implementing the Asynchronous-Start and Terminating LCR Algorithm

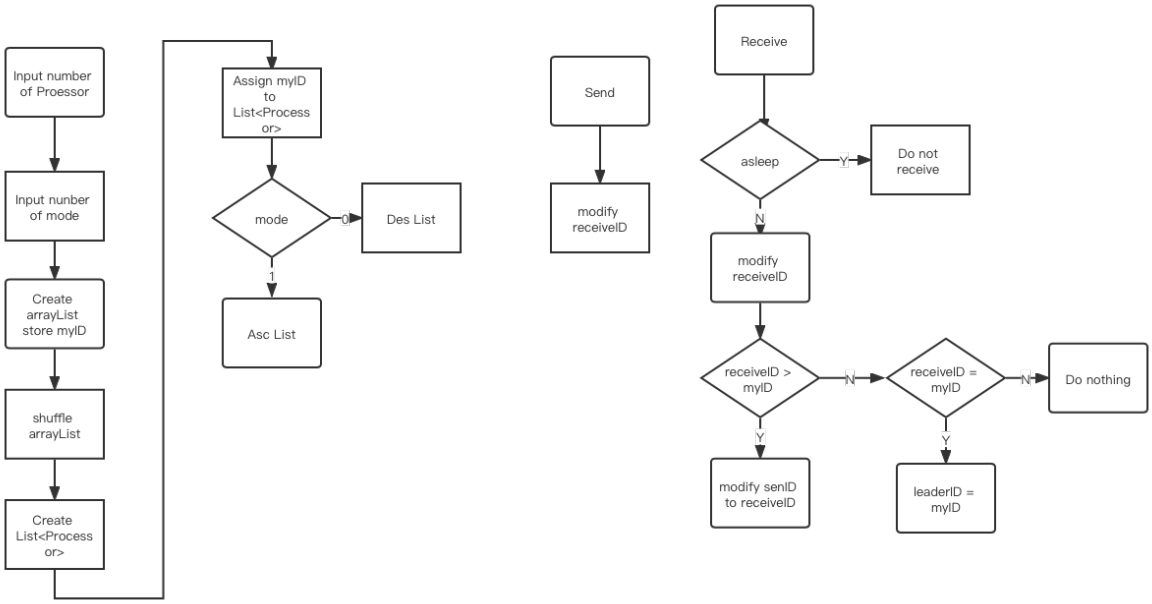
1. Problem Analysis

To better realize the functions, two classes are designed in 3.1.

1. UML diagram in class Processor



2. Flow chart in main class



2. Implementing Souce Code

java

```java
int num_processor; //denote number of processor in a ring

List<Processor> processes = new LinkedList<>();

ArrayList<Integer> arr = new ArrayList<Integer>();
for(int i = 0; i < 3*num_processor; i++){
    arr.add(i);
}
Collections.shuffle(arr);

//Build a List, contain n processors, ranged from 0 to 3*n
for (int i = 0; i < num_processor; i++) {
    processes.add(new Processor(arr.get(i)));
}
```

1. Build a List contains the objects of Processor with input number of processors in it. Here we use variable num_processor to denote the number of processors in a ring.
2. First, create a List called processes.
3. Second, create an integer ArrayList, which used to create myID of each proessor. The range of myID is [0, 3*num_processor). And the range can be modified.
4. Importantly, we use function javaCollections.shuffle(arr) to assign the myID in ArrayList randomly to achieve universality.
5. Then we build a List to contain Processors. We assign the myID from arr from 0 to num_processor to all the processors in the List processes.
6. Finally, we realized the function of creating a List of Processors (a Ring), of which the myID of each processor is assigned randomly and the range is [0, 3*num_processor).

java

```java
//send
processes.get(0).setReceiveID(processes.get(processes.size() - 1).getSendID());
for (int i = 1; i < processes.size(); i++) {
    processes.get(i).setReceiveID(processes.get(i - 1).getSendID());
}
```

java

```java
//receive
for (int i = 0; i < processes.size(); i++) {
    if (processes.get(i).getStartRound() <= round) {
        //receiveID > myID,  set sendID to receiveID
        if (processes.get(i).getReceiveID() > processes.get(i).getMyID()) {
            processes.get(i).setSendID(processes.get(i).getReceiveID());
        } else if (processes.get(i).getReceiveID() == processes.get(i).getMyID()) {
            leader = processes.get(i).getInID();
            break;
        } else {
```

```
                processes.get(i).setSendID(processes.get(i).getMyID());
            }
        }
        message++;
    }
```

A processor should have two behavious in a round: Send and Receive

1. When sending message, we change the receiveID of each Processor. When receiving, we compare the receiveID with myID and modify the sendID if needed.
2. For behaviour Send, it is not limited by the status of Processor, which means if a processor is asleep, it will still send message.
3. For behaviour Receive, it is limited by status, which means is a processor is asleep, it will not receive any message. In receive function, it has three posibilities. (a) receiveID > myID, change sendID (b)receiveID = myID, set leaderID and get out of the circulation (c)receiveID < myID, don't do anything.
4. Beforehead, we set leaderID = -1, and as soon as leaderID != -1, get out of circulation and get round and message.

3.2 Implementing a Leader_Election Algorithm for Rings of Rings

1. Problem Analysis

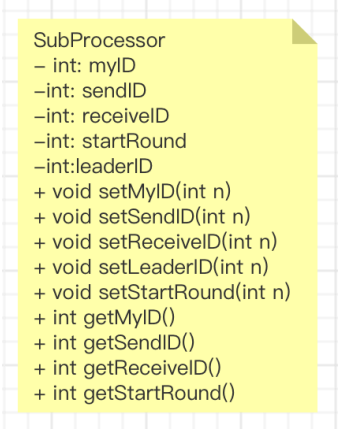
To better demonstrate a rings of rings structure, we build a new class SubProcessor.

And add a new variable attribute in Pricessor class to solve if this processor is interface or not.

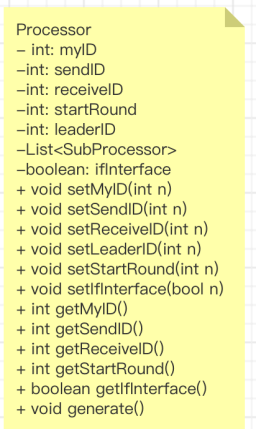
If yes, this proecssor should has a List which demonstrate its subRing.

1. UML

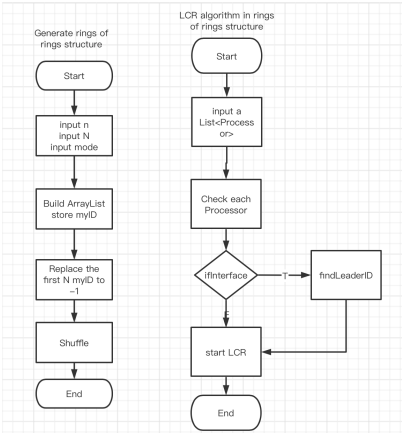
UML of class SubProcessor



UML of class Processor



3. Flow Chart



2. Results and plots

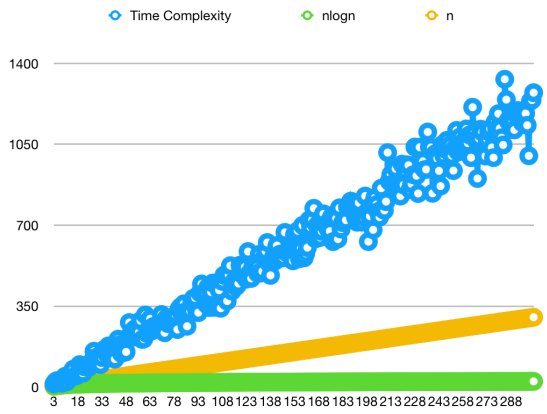
1. Condition 1:

   a) Number of processors in each subRing is random. Ranges [3,5)

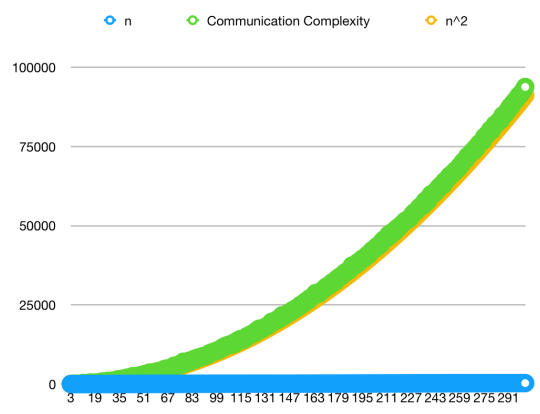
   b) Number of processors in main ring is n

   c) Number of interface in a main ring is N = n/2

   d) myID assignment setting is Des



x-axis: number of processors in main ring



x-axis: number of processors in main ring

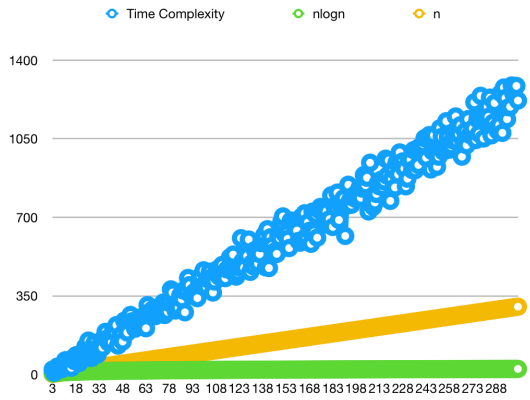
2. Condition 2

a) Number of processors in subRing is random. Ranged [3, 10)

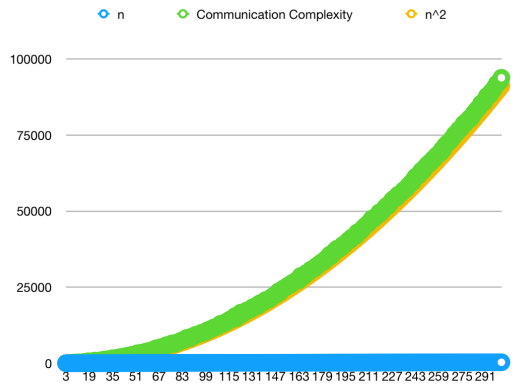
b) Number of processors in main ring is n

c) Number of interface in a main ring is N = n/2

d) myID assignment setting is Asc



x-axis: number of processor in main ring



x-axis: number of processor in main ring