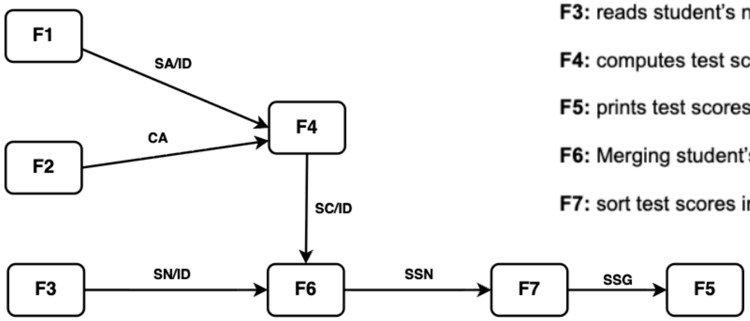
Problem (1)

Part(A):



F1: reads student's test answers together with student's IDs.

F2: reads the correct answers for the test.

F3: reads student's names together with their IDs.

F4: computes test scores.

F5: prints test scores & student names in the order as they are read from an input pipe.

F6: Merging student's test scores with their names and IDs.

F7: sort test scores in **ascending** order with respect to scores with student names.

SA: Student's Answers

ID: Student's IDs

SC: Student's scores

SN/ID: Student's Names and their ID's

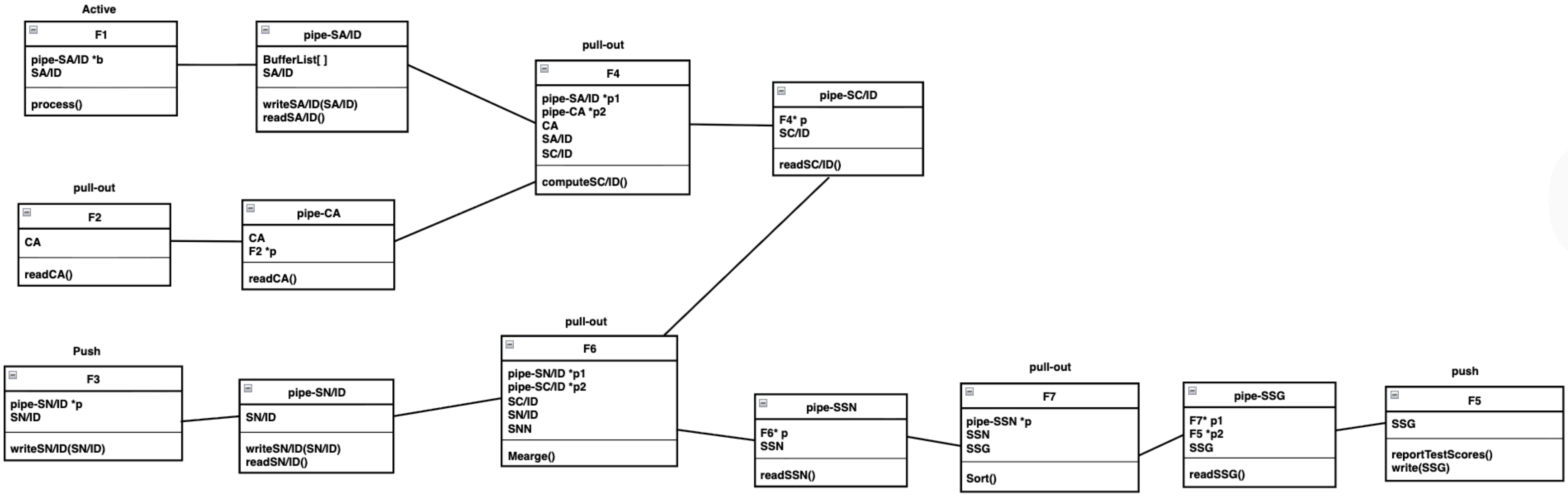
CA: Correct Answers

SC/ID: Student's scores with their IDs.

SSN: Student's Names, IDs, and their Test scores.

SSG: Student's Names, IDs and Test scores Sorted in **ascending** order with respect to score.

Problem 1 - Part(B)



Problem 1 - Part(B) Cont’d.

Filters:

```
Class F1 {
Pipe-SA/ID *b
SA/ID // Student's names together with student’s IDs.

process(){
Loop{
Read student’s names with their IDs into SA/ID
b=> writeSA/ID(SA/ID)
EndLoop
} } }

Class F2 {
CA // correct answers for the test
readCA(){
//Read the Correct Answer into CA
Return CA;
}
}

Class F3 {
pipe-SN/ID *p
SN/ID // Students names together with students IDs

writeSN/ID(SN/ID){
//Read students names with IDs into SN/ID
p=>writeSN/ID(SN/ID);
} }

Class F4 {
Pipe-SA/ID *p1
Pipe-CA * p2
SA/ID // Students test answers together with students IDs
CA// correct answers for the test
SC/ID // Students test scores with their IDs

computeSC/ID(){
SA/ID = p1=> readSA/ID();
CA = p2 => readCA();
if( CA && SAID NOT empty)
        //Compute  students grads and store it in SC/ID with ID;
        return SC/ID;
Endif    }
```

```
Class F5 {
SSG // Student names, IDs and test scores sorted in descending grade order

reportTestScores(){
If SSG is not empty
Print SSG;
}
writeSSG(SSG){
Store incoming SSG  into SSG
}
}
```

```
Class F6 {
Pipe-SN/ID *p1
Pipe-SC/ID *p2
SSN    // Students names , ID and test scores
SN/ID // Students names together with students IDs
SC/ID // Students test scores together with IDs

Mearge() {
SN/ID = p1=> readSN/ID();
SC/ID = p2 =>readSC/ID();

if( SC/ID and SN/ID NOT empty)
        SSN = Merge Students Name && ID && scores depending on ID
        Return SSN;
EndIf
}
}
```

```
Class F7 {
Pipe-SSN *p

SSN // Student names, ID and test scores
SSG // Student names, ID and test scores in ascending order depending on the score grade

Sort()
{
SSN= p=> read_SSN();
If (SSN NOT empty)
        SSG = //Sort SNN values(students name, ID and scores) in ascending grade order;
        Return SSG;
ENDIf;  }   }
```

Pipes:

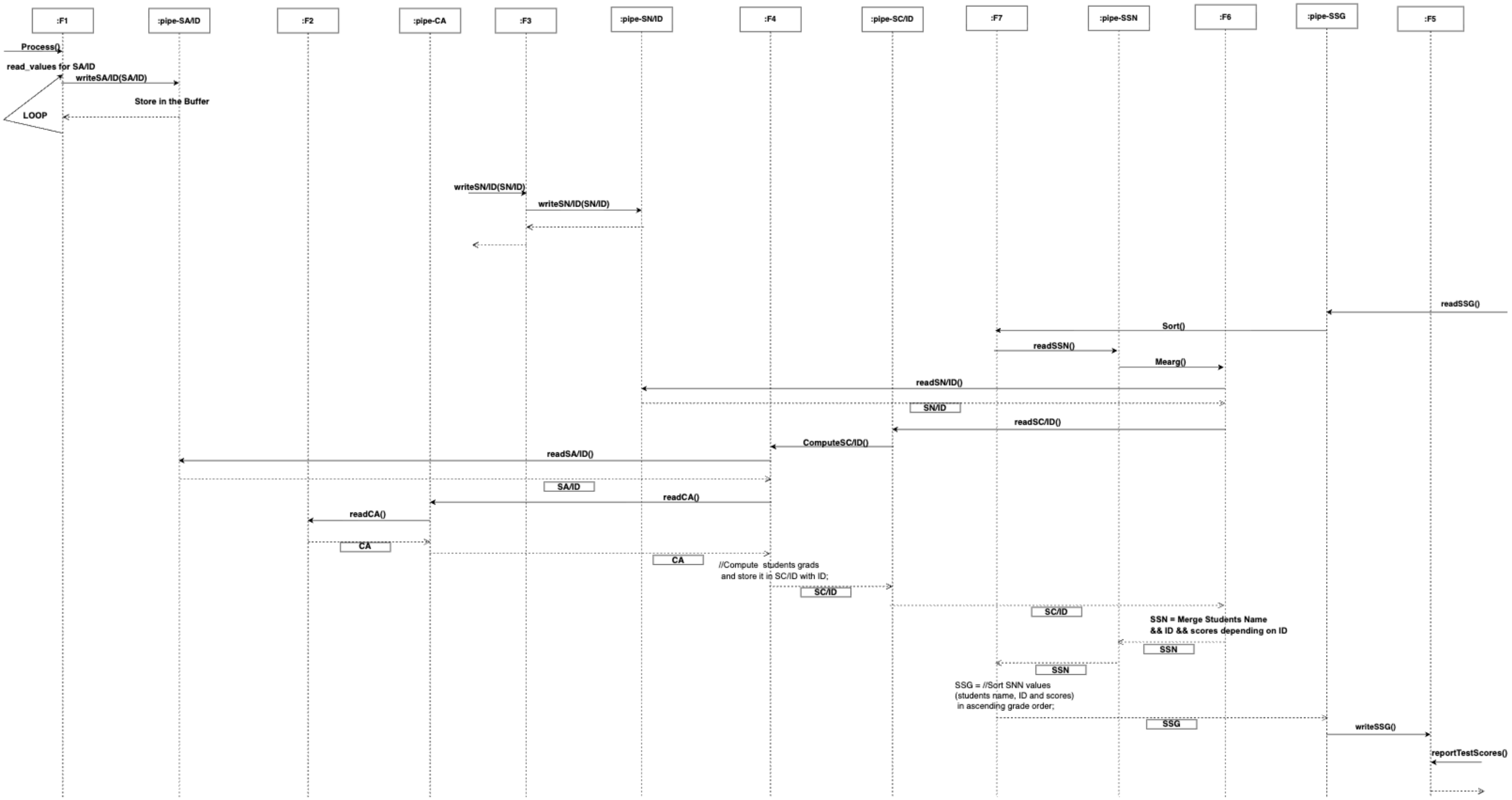
```
Class Pipe-SA/ID {
BufferList [ ]
SA/ID
write_SA_ID(SA/ID){
Store SA/ID in BufferList;
}

read_SA_ID(){
SA/ID = read values of SA/ID from BufferList;
Return SA/ID;
}
}
```

```
Class Pipe-CA {
F2 *p
CA //correct answers for the test


readCA() {
CA = =>readCA()
If CA Not null
return CA;
}
}
```

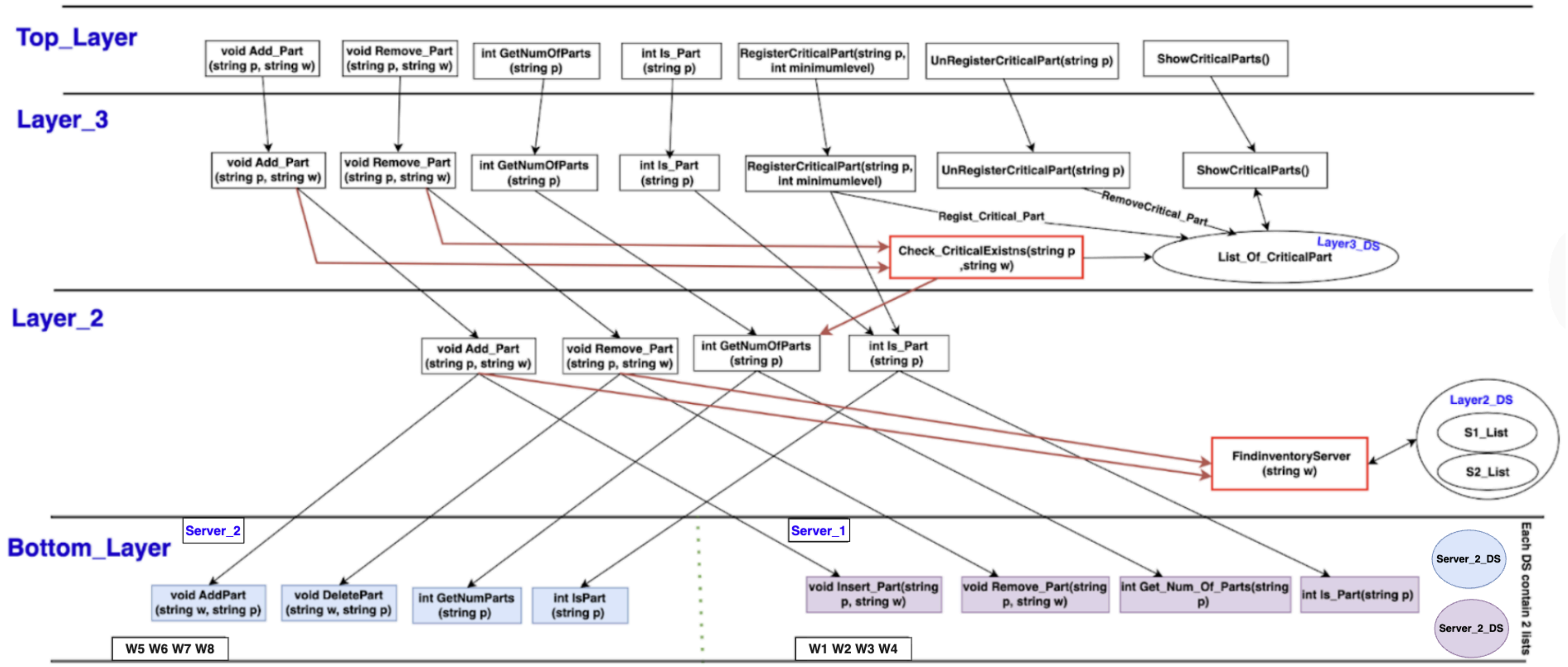
```
Class Pipe-SN/ID {
SN/ID

writeSN/ID(SN/ID){
Store the SN/ID into the SN/ID
}

readSN/ID(){
return SN/ID;
} }
```

```
Class Pipe-SC/ID {
F4 *p
SC/ID // Students test scores together with IDs

read_SC/ID(){
SC/ID = p=> computeSC/ID();
Return SC/ID;
}
}

Class Pipe-SSN {
F6 *p
SSN // Student names, Ids and Test scores

read_SSN(){
SSN = p=> Merge();
Return SSN;
}
}

Class Pipe-SSG{
F7 *p1
F5 *p2
SSG // Student names, IDs, and test scores Sorted in ascending order;

read_SSG(){
SSG = p1=>Sort();
p2=>writeSSG(SSG);

}
}
```

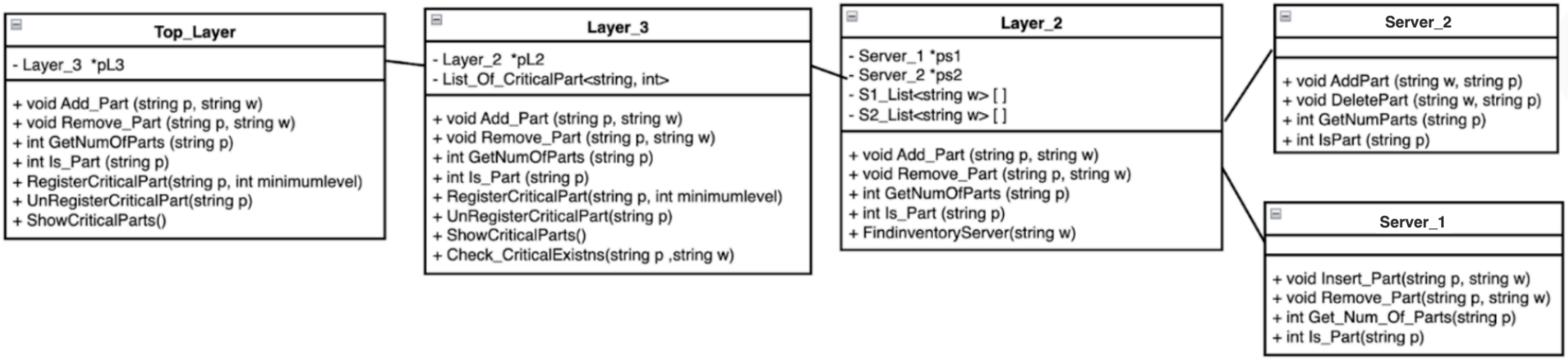
Problem 1- Part(B) - Sequance Digram:



Problem 2


Top_Layer
void Add_Part (string p, string w)
void Remove_Part (string p, string w)
int GetNumOfParts (string p)
int Is_Part (string p)
RegisterCriticalPart(string p, int minimumlevel)
UnRegisterCriticalPart(string p)
ShowCriticalParts()
Layer_3
void Add_Part (string p, string w)
void Remove_Part (string p, string w)
int GetNumOfParts (string p)
int Is_Part (string p)
RegisterCriticalPart(string p, int minimumlevel)
UnRegisterCriticalPart(string p)
ShowCriticalParts()
RemoveCritical_Part
Regist_Critical_Part
Check_CriticalExistns(string p ,string w)
Layer3_DS
List_Of_CriticalPart
Layer_2
void Add_Part (string p, string w)
void Remove_Part (string p, string w)
int GetNumOfParts (string p)
int Is_Part (string p)
Layer2_DS
S1_List
S2_List
FindinventoryServer (string w)
Bottom_Layer
Server_2
Server_1
void AddPart (string w, string p)
void DeletePart (string w, string p)
int GetNumParts (string p)
int IsPart (string p)
void Insert_Part(string p, string w)
void Remove_Part(string p, string w)
int Get_Num_Of_Parts(string p)
int Is_Part(string p)
Server_2_DS
Server_2_DS
Each DS contain 2 lists
W5 W6 W7 W8
W1 W2 W3 W4

Problem 2 (Class Diagram):


Top_Layer
- Layer_3 *pL3
+ void Add_Part (string p, string w)
+ void Remove_Part (string p, string w)
+ int GetNumOfParts (string p)
+ int Is_Part (string p)
+ RegisterCriticalPart(string p, int minimumlevel)
+ UnRegisterCriticalPart(string p)
+ ShowCriticalParts()
Layer_3
- Layer_2 *pL2
- List_Of_CriticalPart<string, int>
+ void Add_Part (string p, string w)
+ void Remove_Part (string p, string w)
+ int GetNumOfParts (string p)
+ int Is_Part (string p)
+ RegisterCriticalPart(string p, int minimumlevel)
+ UnRegisterCriticalPart(string p)
+ ShowCriticalParts()
+ Check_CriticalExistns(string p ,string w)
Layer_2
- Server_1 *ps1
- Server_2 *ps2
- S1_List<string w> [ ]
- S2_List<string w> [ ]
+ void Add_Part (string p, string w)
+ void Remove_Part (string p, string w)
+ int GetNumOfParts (string p)
+ int Is_Part (string p)
+ FindinventoryServer(string w)
Server_2
+ void AddPart (string w, string p)
+ void DeletePart (string w, string p)
+ int GetNumParts (string p)
+ int IsPart (string p)
Server_1
+ void Insert_Part(string p, string w)
+ void Remove_Part(string p, string w)
+ int Get_Num_Of_Parts(string p)
+ int Is_Part(string p)

Problem 2 (Pseudo-code):

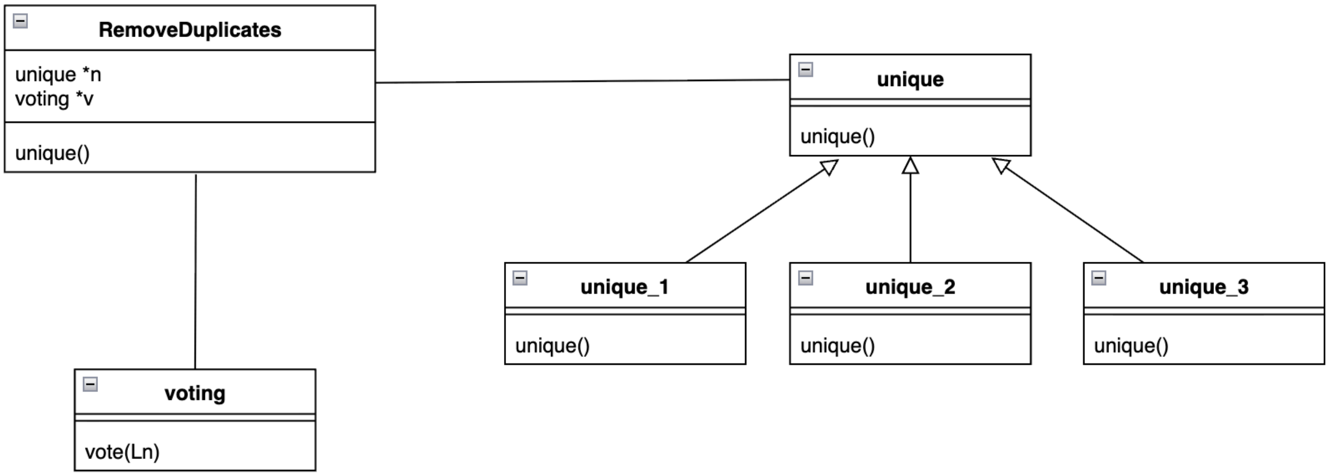
```
Class Top_Layer{
Layer_3 *pL3

void Add_Part (string p, string w){
pL3 => Add_Part  (p, w);
}



void Remove_Part (string p, string w){
pL3 => Remove_Part  (p, w);
}

int GetNumOfParts (string p){
Return pL3=> GetNumOfParts (p);
}

 int Is_Part (string p){
Return pL3=>  Is_Part (p);
}

RegisterCriticalPart(string p, int minimumlevel){
pL3=> RegisterCriticalPart(p, minimumlevel);
}

UnRegisterCriticalPart(string p){
pL3=> UnRegisterCriticalPart (p);
}

ShowCriticalParts() {
pL3=> ShowCriticalParts();
}
}

Class Layer_3{
Layer_2  *pL2
List_Of_CriticalPart<string, int>

void Add_Part (string p, string w){
pL2 =>Add_Part ( p, w);
}
```

```
void Remove_Part (string p, string w){
if( Check_CriticalExistns(p,w) == False) Then
    pL2=> Remove_Part (p, w);
Else
   Reject the Remove request, because it is below the minimum level.
EndIf;
}

int GetNumOfParts (string p){
Return pL2=> GetNumOfParts (p);
}
int Is_Part (string p){
Return pL2=> Is_Part (p);
}

RegisterCriticalPart(string p, int minimumlevel){
If (pL2 => Is_Part(p) == True) Then
   Insert p and minimumlevel into List_Of_CriticalPart;
EndIf
}

UnRegisterCriticalPart(string p){
Remove p from List_Of_CriticalPart;
}

ShowCriticalParts(){
For each <p, minimumlevel> in List_Of_CriticalPart
Display P and minimumlevel.
}

Check_CriticalExistns(string p ,string w){
if( p exist in List_Of_CriticalPart and pL2=> GetNumOfParts(p) > minumlemel)
    Return True;
Else
    Return False;
Endif
}
}

Class Layer_2{
Server_1 *ps1
Server_2 *ps2
S1_List<string w> // contains warehouses of server_1, For example w1,2,3,4.
S2_List<string w> // contains warehouses of server_2, For example w5,6,7,8.
```

```
void Add_Part (string p, string w){
if( FindinventoryServer(w) == “2”) Then
    ps2 =>AddPart(w,p);
Else if(FindinventoryServer(w) == “1”) Then
     ps1=> Insert_Part(p,w);
Endif
}


void Remove_Part (string p, string w){
if(FindinventoryServer(w) == 2) Then
   ps2=> DeletePart(w,p);
Else if(FindinventoryServer(w) == 1) Then
   ps1=> Remove_Part(p,w);
Endif
}

int GetNumOfParts (string p){
Int s2_count = ps2=> GetNumParts(p);
Int s1_count = ps1 => Get_Num_Of_Parts(p);
Int result = s2_count + s1_count;
Return result;
}

int Is_Part (string p){
if(ps2=> IsPart(p) == true/1) or (ps1=> Is_Part(p) == true/1) Then
       Return true/ 1;   // Yes It is here.
Else
       Return false/ 0; // No it’s not here.
Endif
}

FindinventoryServer(string w){
if( w is in S1_List) Then
      Return 1;
Else if (w is in S2_List) Then
      Return 2;
Else
Server Not Found in the System
Endif
}  }
```

Problem 3

Part(1) N-version architecture

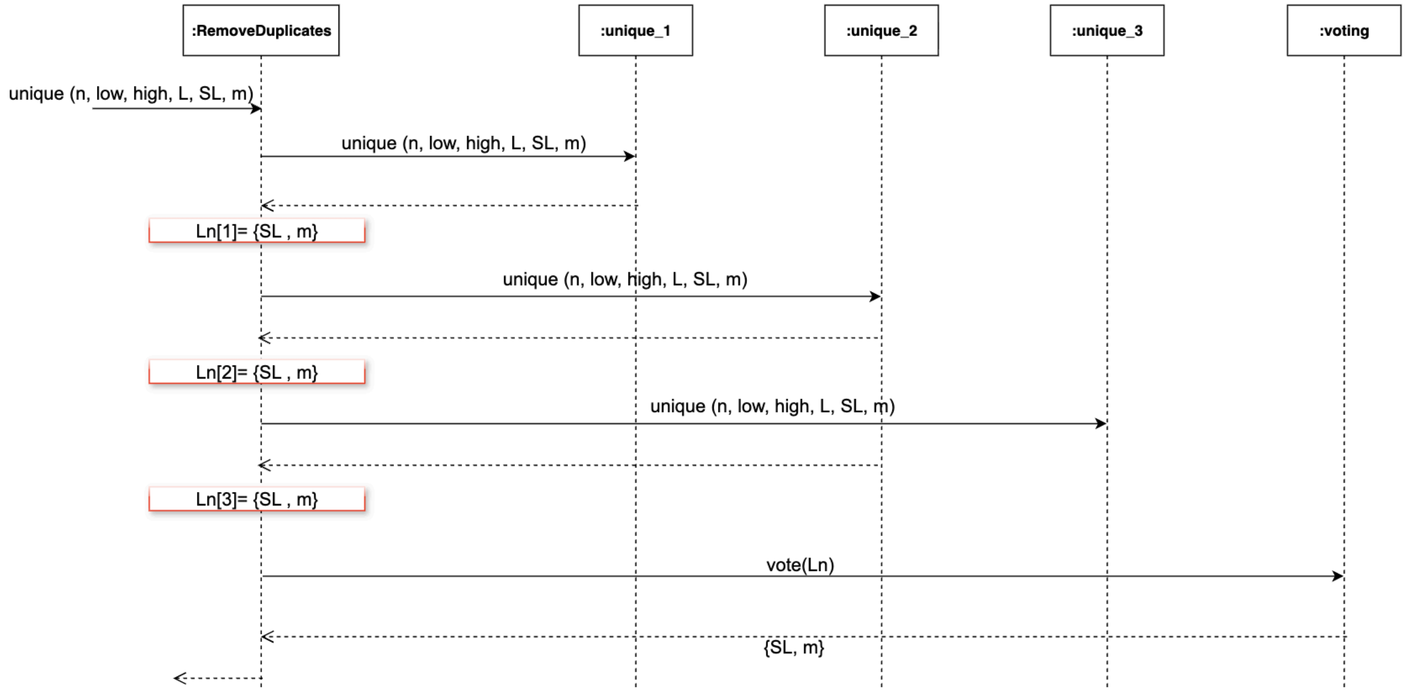


Part(1) pseudo-code

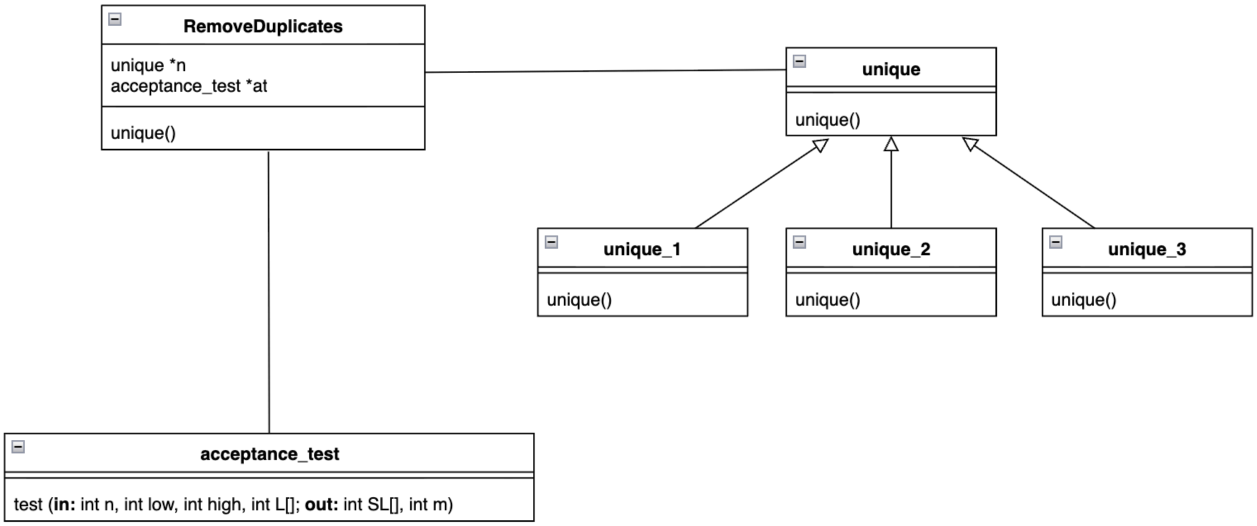
```
Class RemoveDuplicates {
unique *n // points to the unique objects arrray
voting *v  // a pointer to the voting object

void unique (in int n, int low, int high, int L[]; out int SL[], int m){
Ln is a{ int[], int} List // {SL: array of integers, m: single integer value} // Storing the outputs we have SL & m
n[];  // array of objects of type unique
n[1] = new unique_1();
n[2] = new unique_2();
n[3] = new unique_3();

For i=1 to 3
n[i]=> unique(n, low, high, L , SL, m)
Ln[i]= {SL , m};     // storing the results for each unique
EndFor
{SL , m} = v=> vote(Ln);  //storing the last final result
}
}
```

```
Class voting {
{int[], int} vote(in: Ln){ // receive list of 3 objects, each object contains output which is {SL , m}
if( Ln[1] == Ln[2]) Then
        return Ln[1];
Else if (Ln[2] == Ln[3]) Then
        return Ln[2];
Else if( Ln[1] == Ln[3]) Then
        return Ln[3];
Endif
r= Random(1,3) // create a random number between 1 to 3.
return Ln[r] // Randomly select one of the Lns and return it;
}
}
```

Problem 3 - Part(1) Sequence Diagram


:RemoveDuplicates
:unique_1
:unique_2
:unique_3
:voting
unique (n, low, high, L, SL, m)
unique (n, low, high, L, SL, m)
Ln[1]= {SL , m}
unique (n, low, high, L, SL, m)
Ln[2]= {SL , m}
unique (n, low, high, L, SL, m)
Ln[3]= {SL , m}
vote(Ln)
{SL, m}

Problem 3 - Part(2) Recovery-Block architecture



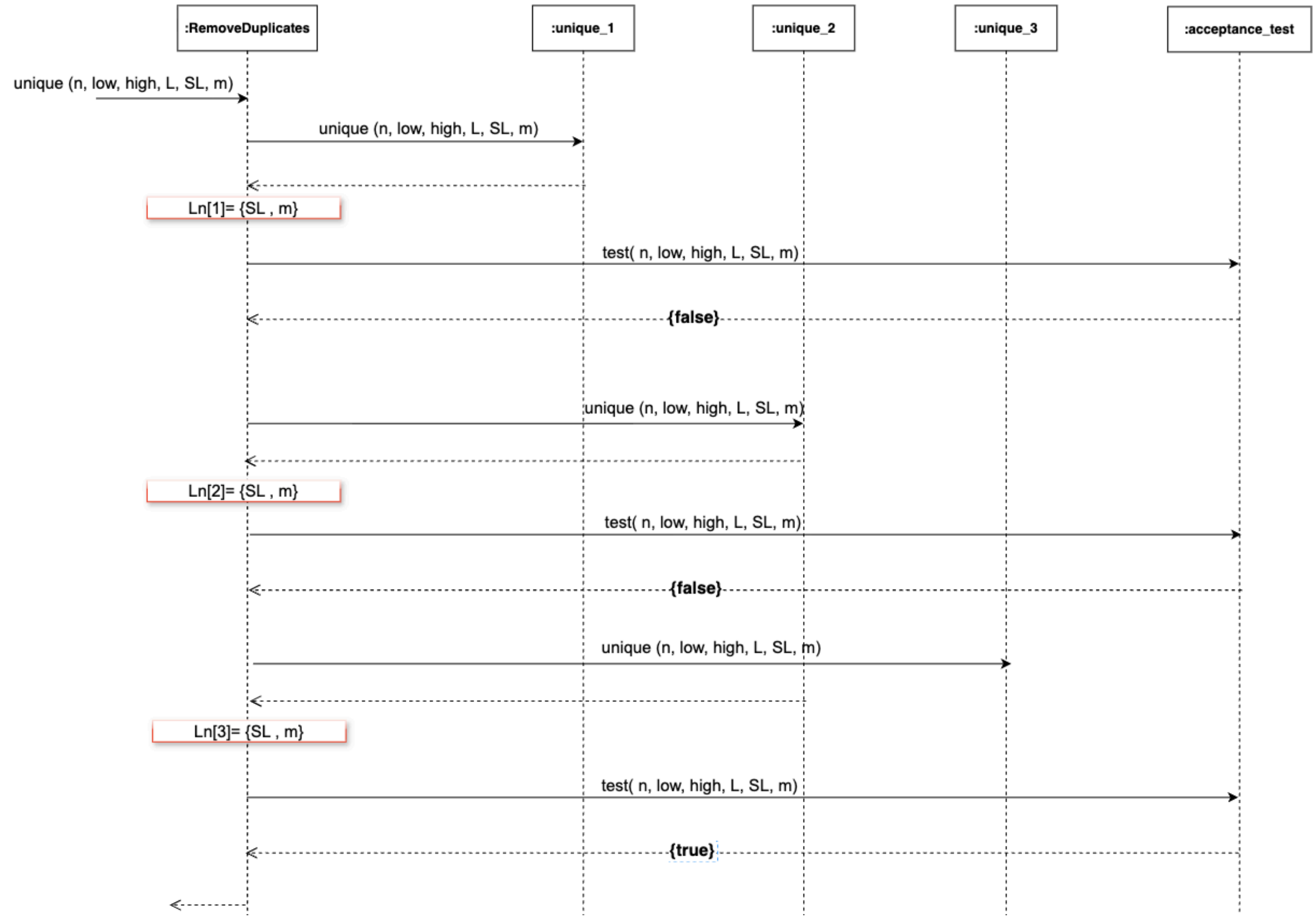
Problem 3 - Part(2) - pseudo-code

```
Class RemoveDuplicates {
unique *n
acceptance_test * at    //  a pointer to acceptance_test object.

void unique (in int n, int low, int high, int L[]; out int SL[], int m){
Ln is a{ int[], int} List // {SL: array of integers, m: single integer value}
 n[];  // array of objects of type unique
n[1]= new unique_1();
n[1]=> unique(n, low, high, L , SL, m) // by reference.
Ln[1]= {SL , m};  // we are storing the output here(returning values).
testResult = at=> test( n, low, high, L, SL, m);
if( testResult == true) Then
          Return true;
Endif
n[2]= new unique_2();
n[2] => unique(n, low, high, L , SL, m)
Ln[2]=  {SL , m}; // we are storing the output here(returning values).
testResult = at=> test( n, low, high, L, SL, m);
if( testResult == true) Then
          Return true;
Endif
n[3]= new unique_3();
n[3] => unique(n, low, high, L , SL, m)
Ln[3]=  {SL , m}; // we are storing the output here(returning values).
testResult = at=> test( n, low, high, L, SL, m);
if( testResult == true) Then
        Return true;
Endif
// if all tests are false
r= Random(1,3)                    // create a random integer number between [1 to 3].
{SL , m} = Ln[r]    // Randomly select on of Lns
}
}
```

Class acceptance_test {

```
boolean test (in: int n, int low, int high, int L[]; out: int SL[], int m){

    // Step 1: Build the expected correct list
    SL_correct[]          // the correct list of integers
    Seen{}                // a set to store values that already seen in the low-high range
    for i from 0 to n-1:
        value = L[i]
        if value >= low AND value <= high:
            if value NOT in Seen:
                add value to SL_correct
                add value to Seen
            else:              // duplicated inside range → skip it
                continue
        else:                  // value outside range → keep exactly as is
            add value to SL_correct
    endfor
    // Step 2: Check m matches size of SL
    if m != length(SL_correct):
        return false
    // Step 3: Check SL contains exactly the same elements
    for i from 0 to m-1:
        if SL[i] != SL_correct[i]:
            return false
    endfor
    // If all checks passed
    return true
}
}
```

Part(2) Sequence Digram


:RemoveDuplicates
:unique_1
:unique_2
:unique_3
:acceptance_test
unique (n, low, high, L, SL, m)
unique (n, low, high, L, SL, m)
Ln[1]= {SL , m}
test(n, low, high, L, SL, m)
{false}
unique (n, low, high, L, SL, m)
Ln[2]= {SL , m}
test(n, low, high, L, SL, m)
{false}
unique (n, low, high, L, SL, m)
Ln[3]= {SL , m}
test(n, low, high, L, SL, m)
{true}

End of File….