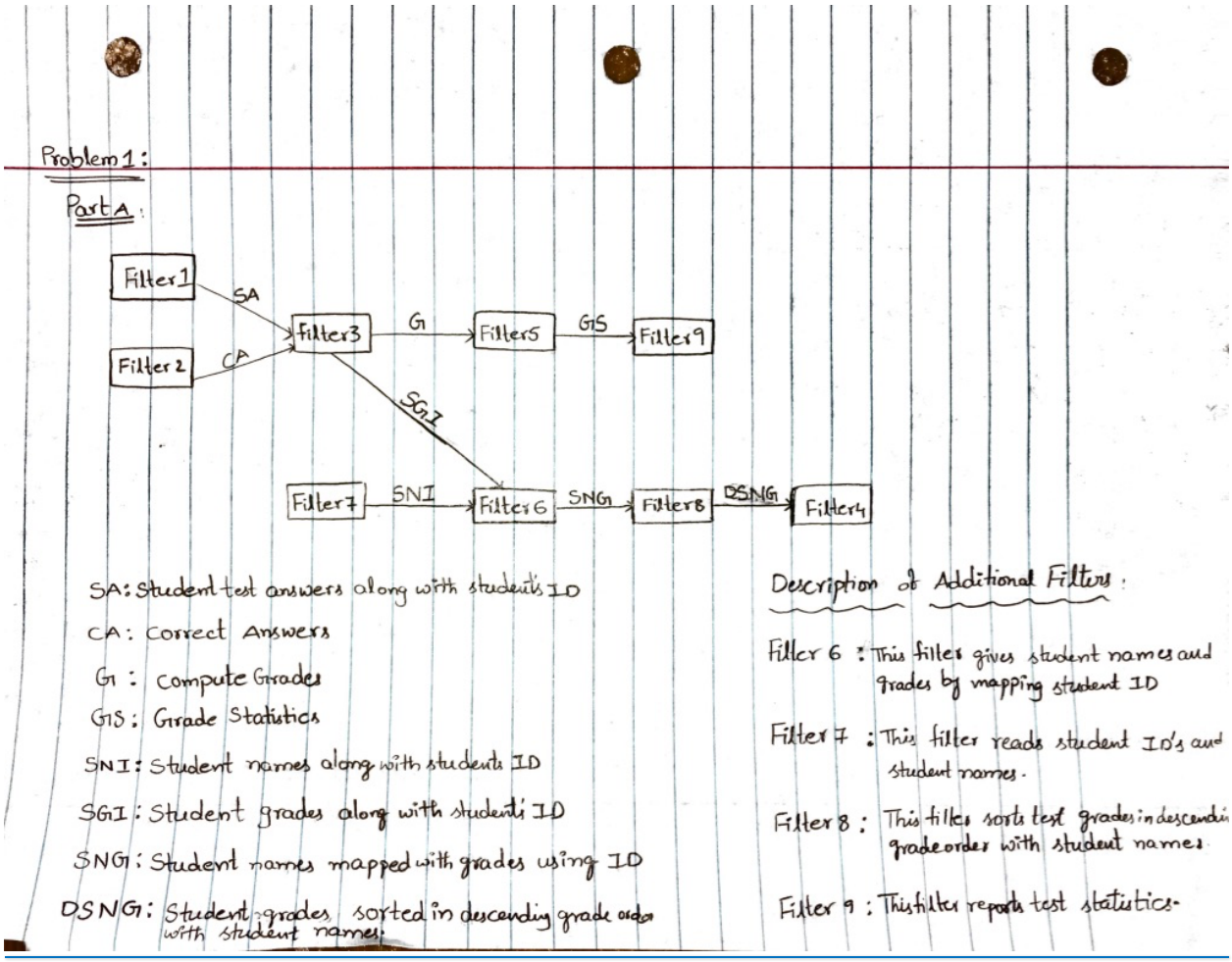
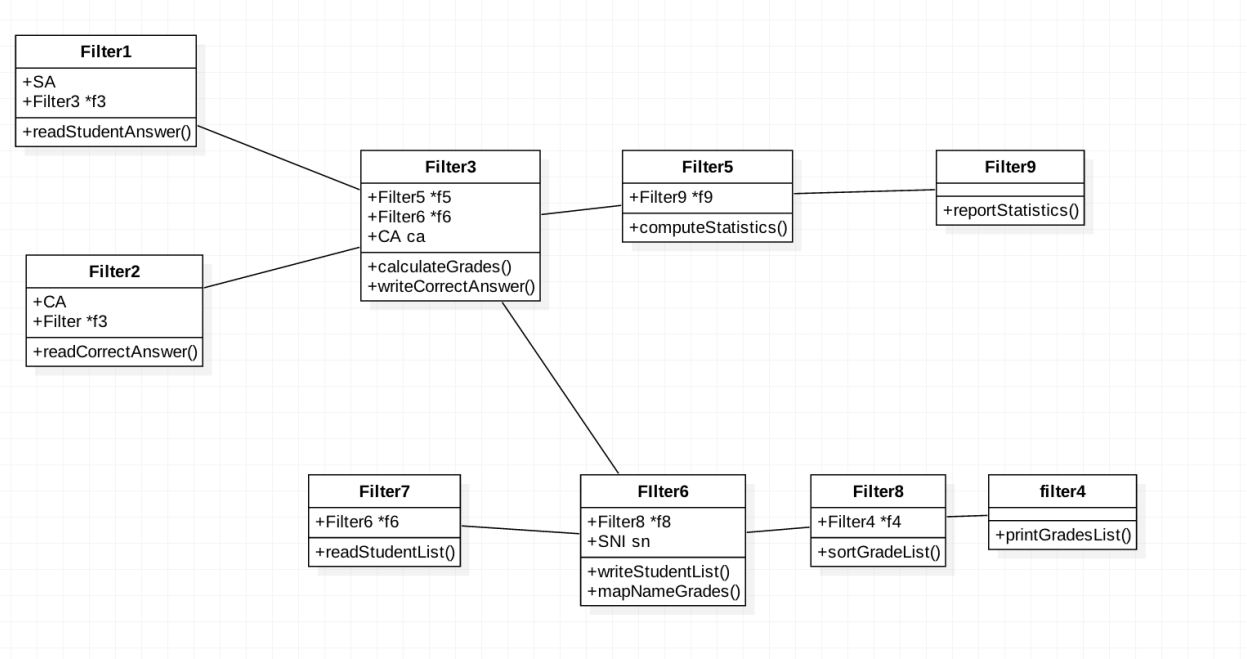
Homework 3

Problem 1:



Part B:

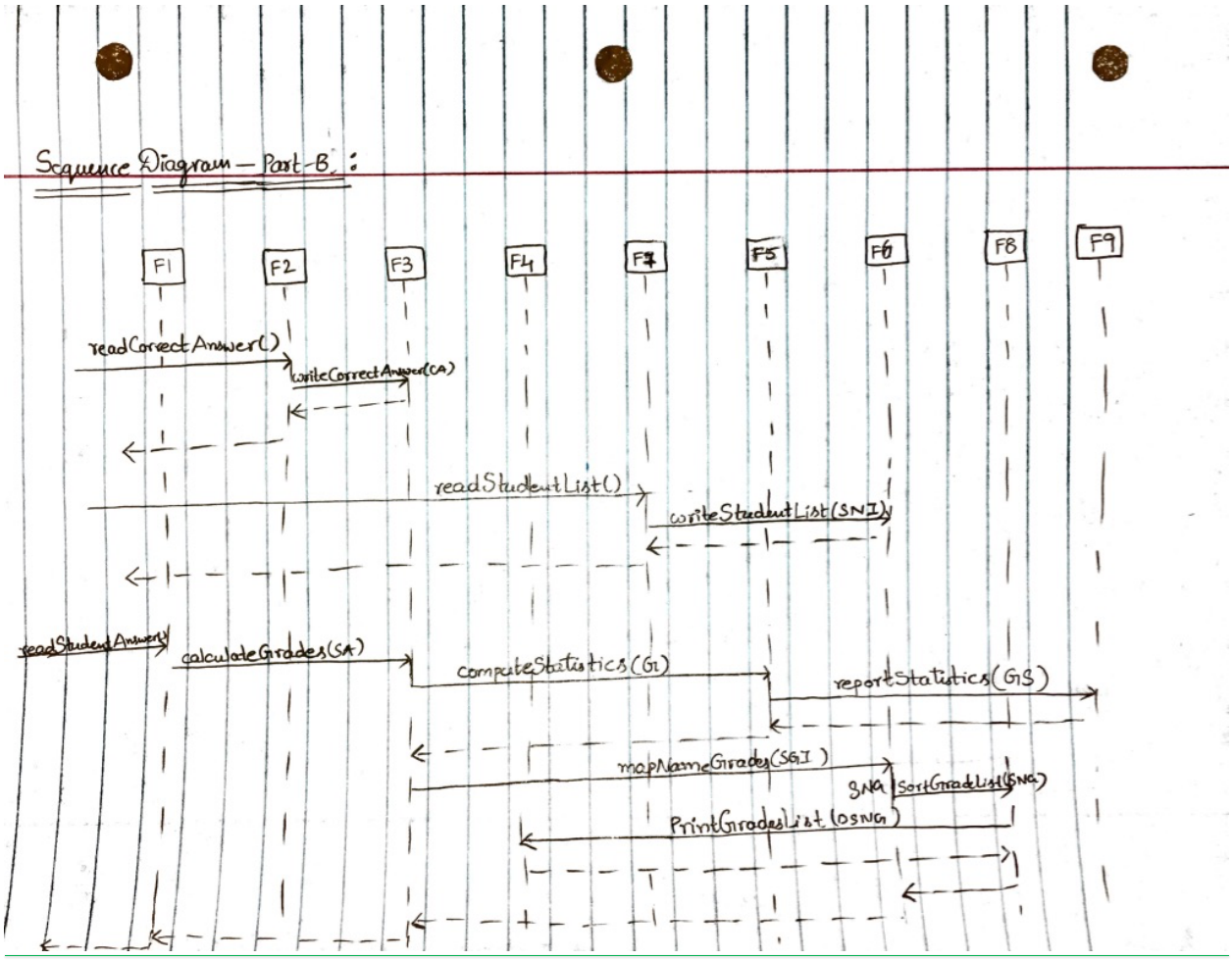


Pseudocode:

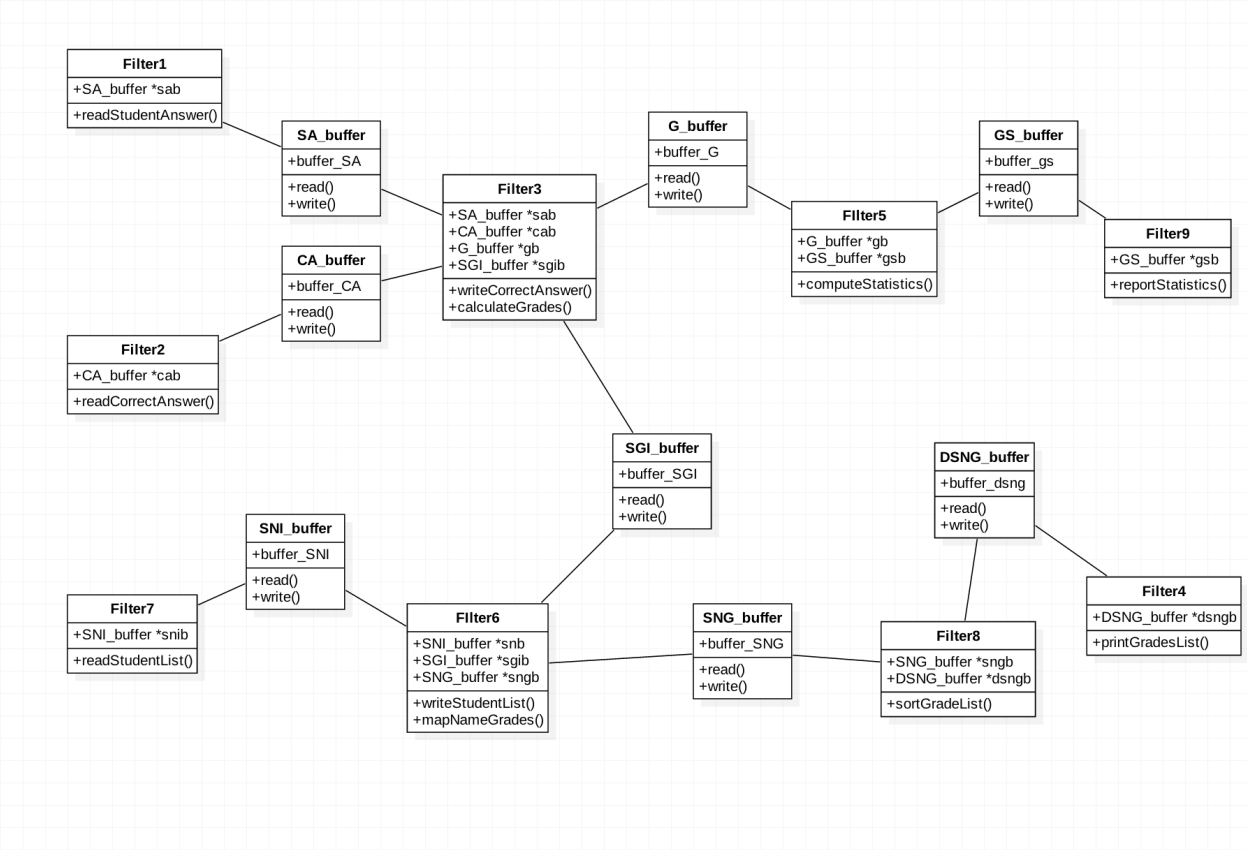
```
Class Filter1 {
  Filter3 *f3;
  readStudentAnswer(){
    SA = reads List of Student Test Answers along with Students Id
    f3 -> calculateGrades(SA)
  }
}

Class Filter2 {
  Filter3 *f3;
  readCorrectAnswer(){
    CA = reads correct answers
    f3 -> writeCorrectAnswer(CA)
  }
}

Class Filter3 {
  CA ca
  Filter5 *f5;
  Filter8 *f8;
  writeCorrectAnswer(CA ca){
    this.ca = ca;
  }
  calculateGrades(SA sa){
    SGI = calculate Grades;
    G = SGI -> getGrades();
    f8 -> writeStudentList(SGI);
    f5 -> computeStatistics(G);
  }
}

Class Filter5 {
```

```
  Filter9 *f9;
  computeStatistics(G g){
    GS = grade statistics;
    f9 -> reportStatistics(GS);
  }
}

Class Filter9 {
  reportStatistics(GS gs){
  }
}

Class Filter7 {
  Filter6 *f6;
  readStudentList(){
    SNI = student list with names & ids;
    f6 -> writeStudentList(SNI);
  }
}

Class Filter6 {
  SNI sni;
  Filter8 *f8;
  writeStudentList(SNI sni){
    this.sni = sni;
  }
  mapNameGrades(SGI sgi){
    SGI = Student Name & Student Grades List
    f8 -> sortGradeList(SNG)
  }
}

Class Filter8 {
  Filter4 *f4;
  sortGradeList(SNG sng){
    DSNG dsng = sort(sng);
    f4 -> printGradesList(dsng)
  }
}

Class Filter4 {
  printGradesList(DSNG dsng){
  //prints Sorted gradelist (Descending order)
  }
}
```

Sequence Diagram:



Part-c:



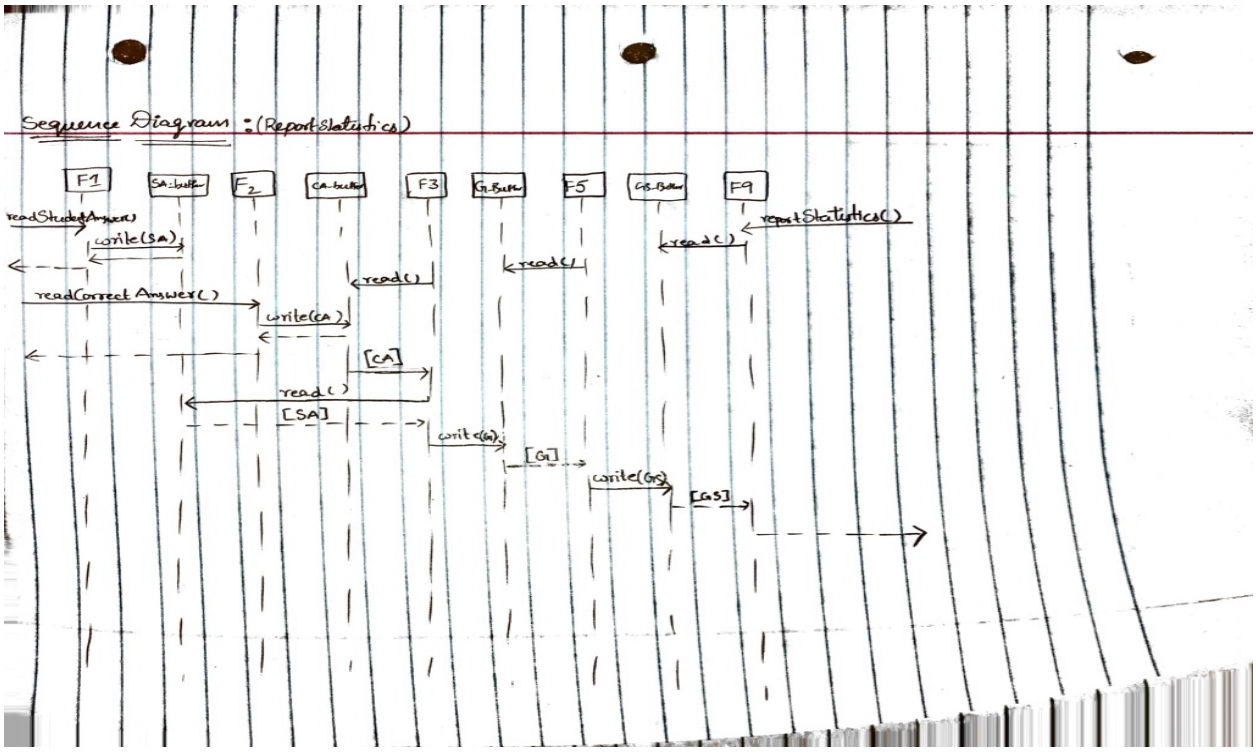
Pseudocode:

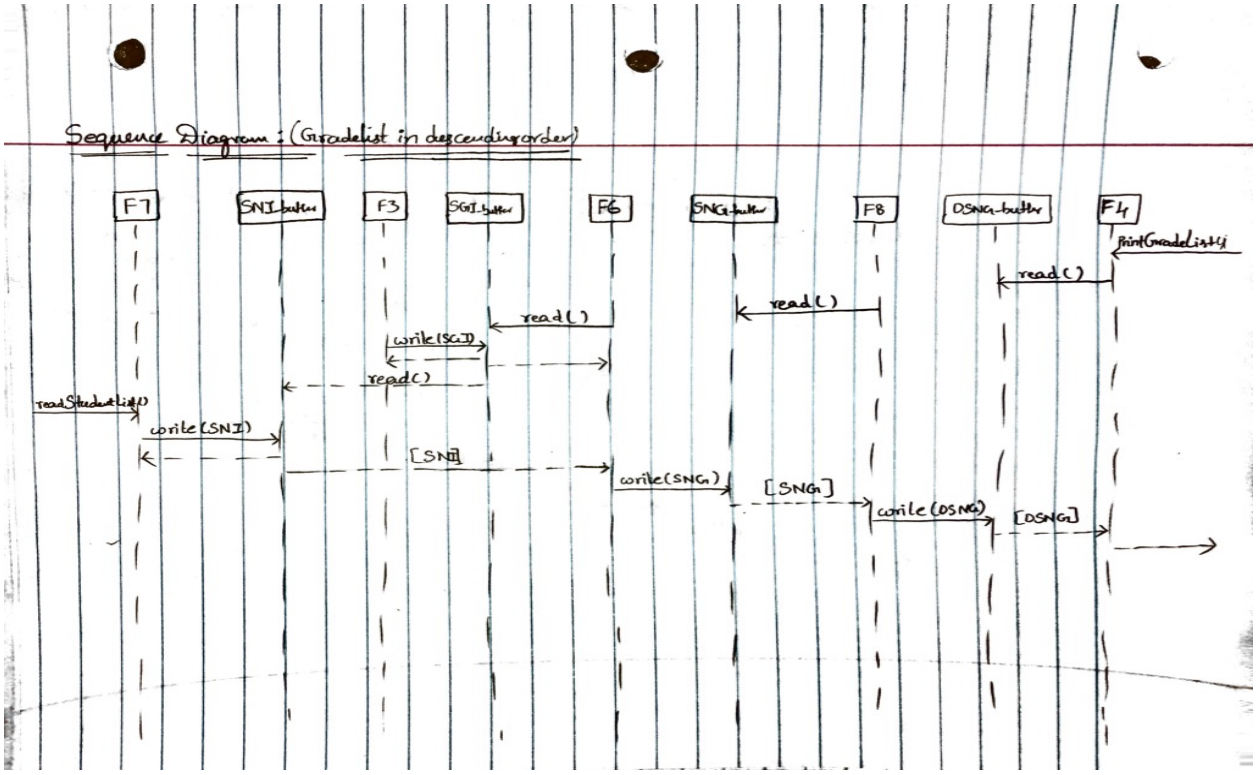
```
Pipe/Buffer Classes
    list read(){
      if buffer is empty then
        wait until the buffer is not empty
        delete list from buffer
        return list
      ENDIF
    }
    write(list){
      put list into the buffer
    }
  }

  Class Filter1 {
    SA_buffer *sab
    readStudentAnswer(){
      while(1){
        // Read student test answers in to SA
        sab -> write(SA)
      }

    }
  }

  Class Filter2 {
    CA_buffer *cab
```

```
  readCorrectAnswer(){
    while(1){
      // Read Correct answers in to CA
      cab -> write(CA)
    }

  }
}

Class Filter3 {
  SA_buffer *sab
  CA_buffer *cab
  G_buffer *gb
  SGI_buffer *sgib
  calculateGrades(){
    while(1){
      CA = cab -> read()
      SA = sab -> read()
      gb -> write(G)
      sgib -> write(SGI)
    }

  }
}

Class Filter4 {
  DSNG_buffer *dsngb
  printGradeList(){
    while(1){
      DSNG = dsngb -> read()     // Print sorted grade list
    }
  }
}

Class Filter5 {
  G_buffer *gb
  GS_buffer *gsb
  computeStatistics(){
    while(1){
      G = gb -> read()          // Compute grade statistics and load into GS objects
      gsb -> write(GS)
    }
  }
}

Class Filter9 {
  GS_buffer *gsb
  reportStatistics(){
    while(1){
      GS = gsb -> read()        // report grade statistics
    }
  }
}

Class Filter7 {
  SNI_buffer *snib
  readStudentList(){
    while(1){
      snib -> write(SNI)
    }
```

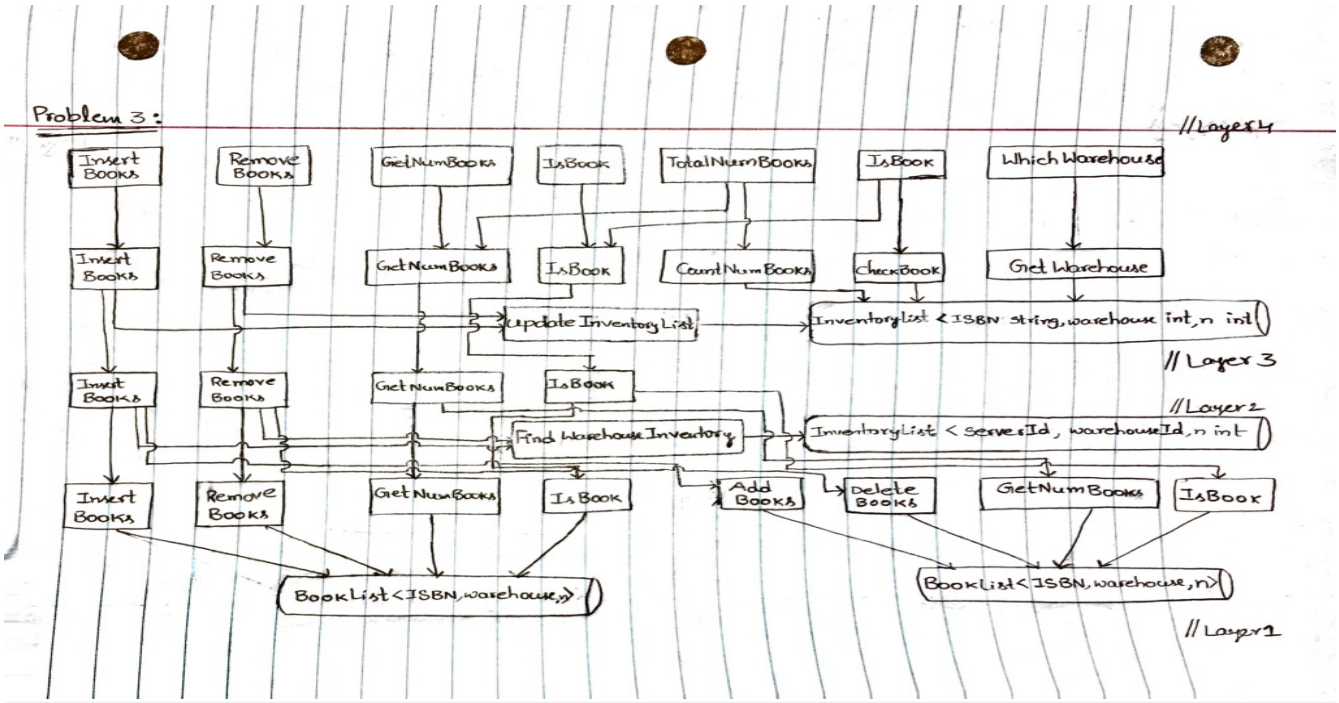
```
    }
  }

  Class Filter6 {
    SNI_buffer *snib
    SGI_buffer *sgib
    SNG_buffer *sngb
    mapNameGrades(){
      while(1){
        SGI = sgib -> read()
        SNI = snib -> read()
        sngb -> write(SNG)
      }

    }
  }

  Class Filter8 {
    SNG_buffer *sngb
    DSNG_buffer *dsngb
    sortGradeList(){
      while(1){
        SNG = sngb -> read()
        dsngb -> write(DSNG)
      }

    }
  }
```

Sequence Diagram :

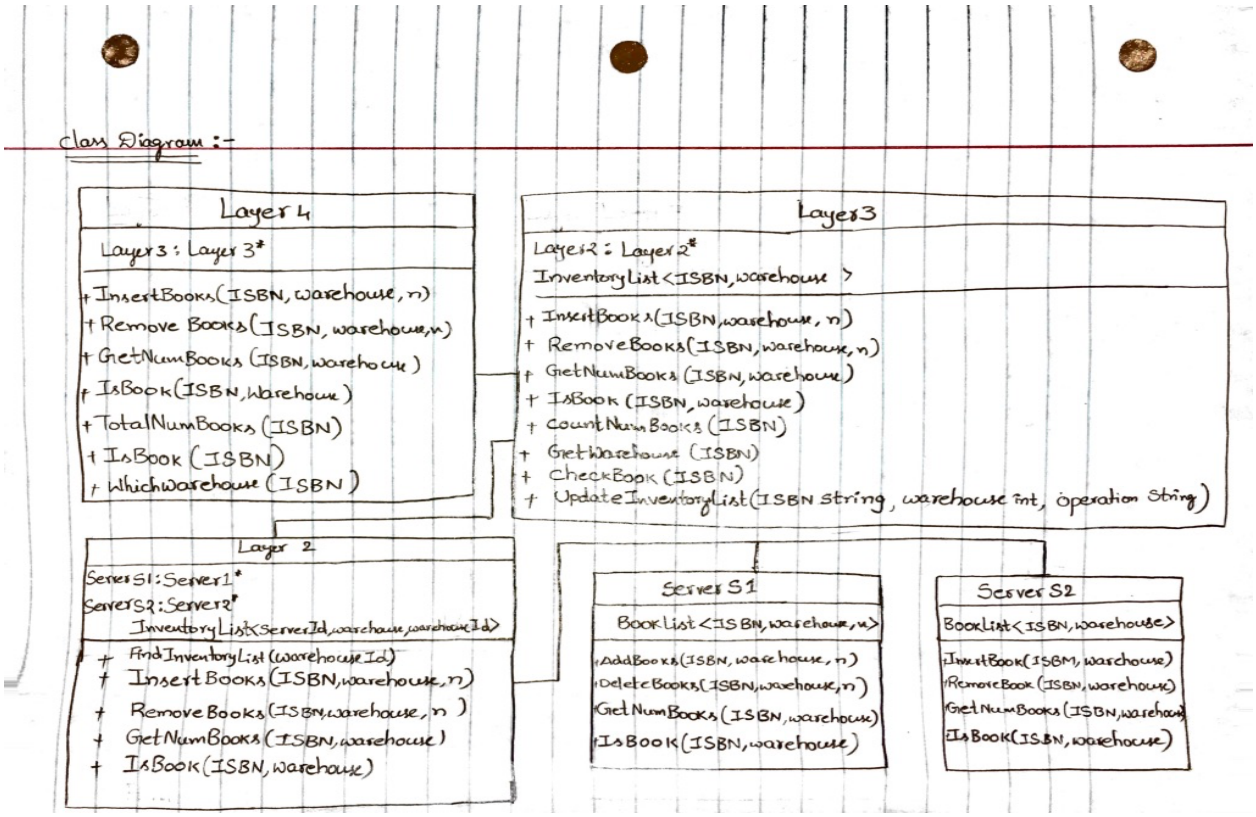



Sequence Diagram : (Gradelist in descending order)
F7
SNI-buffer
F3
SGI-buffer
F6
SNG-buffer
F8
DSNG-buffer
F4
printGradeList()
read()
read()
read()
write(SGI)
read()
readStudentList()
write(SNI)
[SNI]
write(SNG)
[SNG]
write(DSNG)
[DSNG]

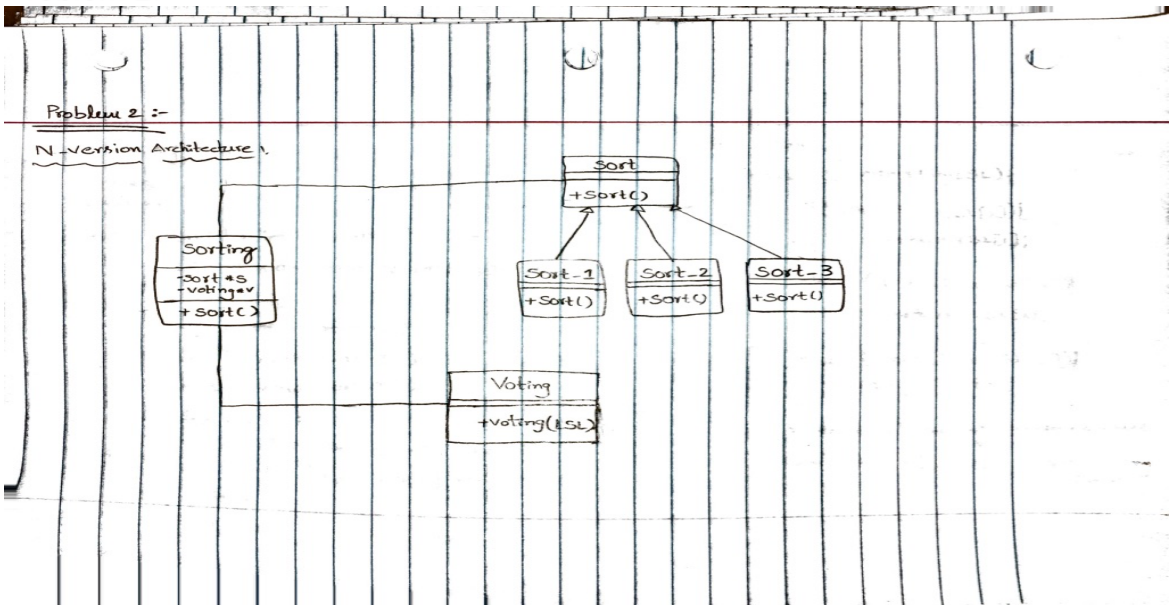
Problem 3:


Problem 3 :
//Layer4
Insert Books
Remove Books
GetNumBooks
IsBook
TotalNumBooks
IsBook
Which Warehouse
Insert Books
Remove Books
GetNumBooks
IsBook
CountNumBooks
CheckBook
Get Warehouse
Update Inventory List
InventoryList <ISBN string, warehouse int, n int>
// Layer 3
Insert Books
Remove Books
GetNumBooks
IsBook
//Layer2
Find Warehouse Inventory
InventoryList < serverId, warehouseId, n int>
Insert Books
Remove Books
GetNumBooks
IsBook
Add Books
Delete Books
GetNumBooks
IsBook
BookList<ISBN, warehouse,>
BookList<ISBN, warehouse, n>
// Layer1

Class Diagram:



Problem 2:

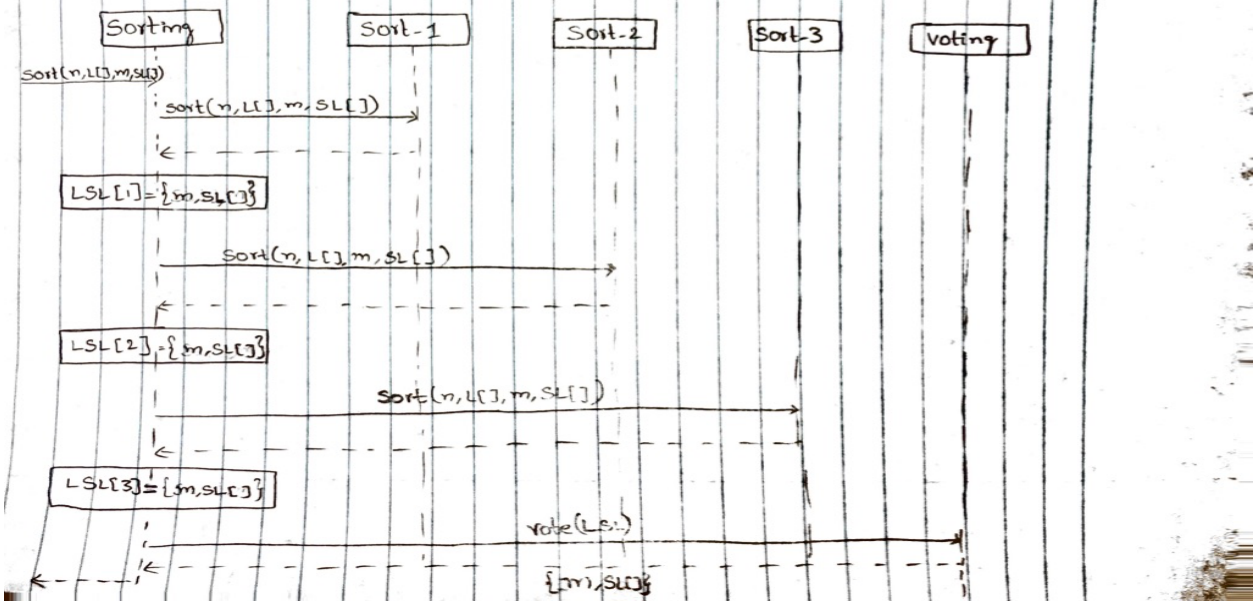


Pseudocode:

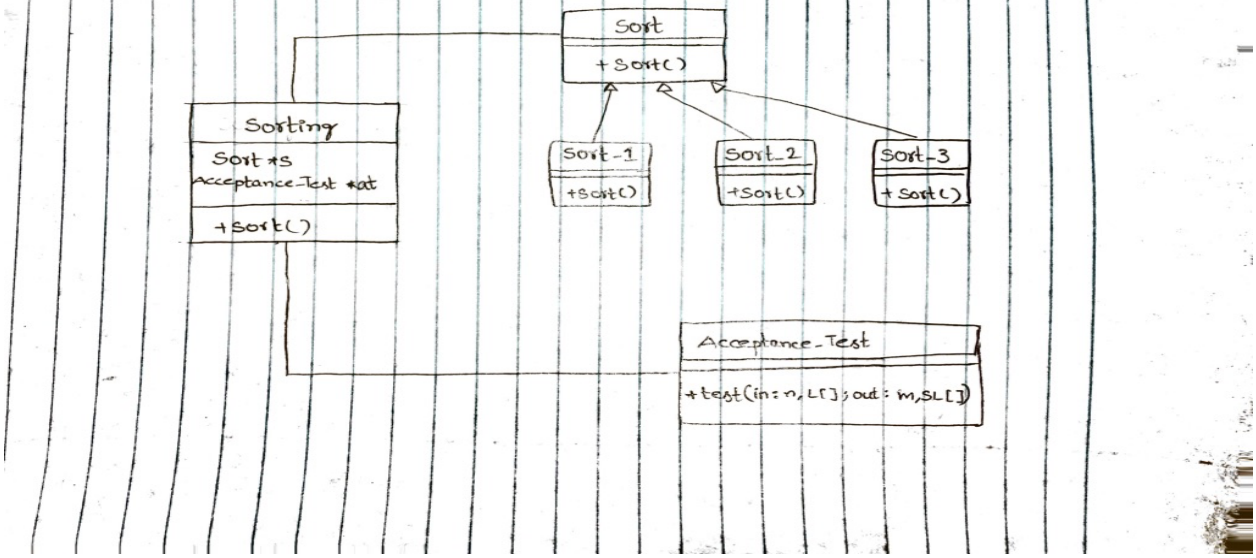
```
Class Sorting
    Sort *S
    voting *V
    void Sort(in int n, int L[]; out int m, int SL[]){
            LSL[] is an array of {m, SL[]} list
            Sort S[];
            S[1] = new Sort_1();
            S[2] = new Sort-2();
            S[3] = new Sort-3();
            for i=1 to 3
                S[i].Sort(n, L[], m, SL[]);
                LSL[i] = {SL[], m};
            EndFor
            {m, SL[]} = V.voting(LSL);
    }
```

```
class Voting
  {int[], int} voting (in LSL){
        if (LSL[1] == LSL[2]){
            return LSL[1];
        }
        else if (LSL[1] == LSL[3]){
            return LSL[3];
        }
        else if (LSL[3] == LSL[2]){
            return LSL[2];
        }
        Y = Rand(1, 3) // generate random numbers btw 1 to 3
        return LSL[Y] // randomly select one from LSL and return it
  }
```

Sequence Diagram:



Recovery Block Architecture :-



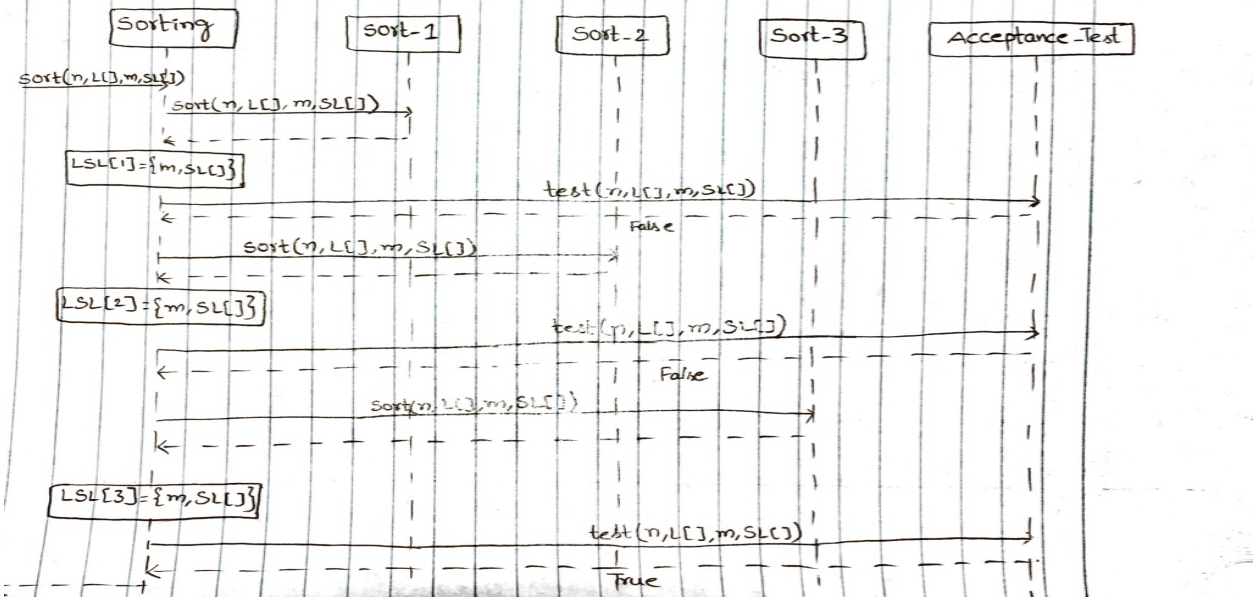
Pseudocode :-

```
class Acceptance_Test:
    boolean test(in: int n, int L[]; out: int m, int SL[])
    if(m > n)
        return false;
    else if (m != SL.length())
        return false;
    else if{
        for i = 0 to m
            for j = 1 to m
                if ((SL[i] == SL[j]), && i != j)
                    return false;
    }
    return True;
}
```

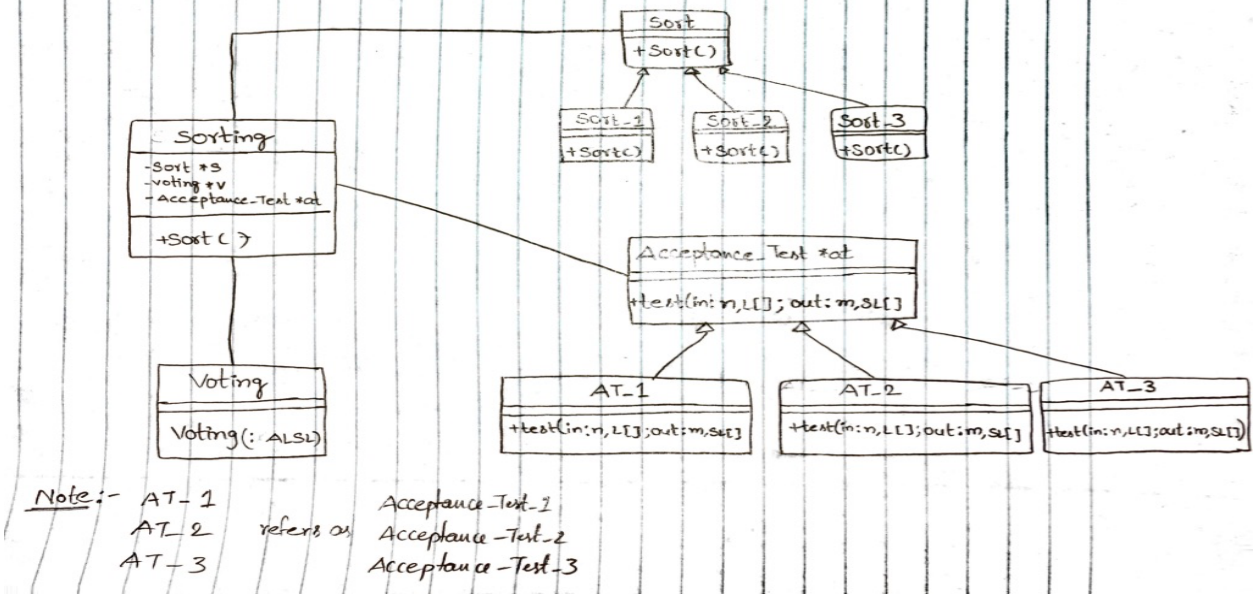
```
class Sorting
    Sort *s
    Acceptance_Test *at
    void sort (in: int n, int L[]; out: int m, int SL[])
    {
        Sort S[];
        {m, SL[]} = LSL[];
        S[1] = new Sort_1();
        S[1].Sort(n, L[], m, SL[]);
        LSL[1] = {m, SL[]}
        testResult = at.test(n, L[], m, SL[]);
        if(testResult == True){
        }   Exit;
        S[2] = new Sort_2();
        S[2].sort(n, L[], m, SL[]);
        LSL[2] = {m, SL[]}
        testResult = at.test(n, L[], m, SL[])
        if (testResult == True){
        }   Exit;
```

```
S[3] = new Sort_3();
S[3].Sort(n, L[], m, SL[]);
LSL[3] = {m, SL[]}
testResult = at.test(n, L[], m, SL[]);
if (testResult == True){
    Exit;
}
if all tests are False
        r = Rand(1, 3)  // generate random numbers
                           b/w 1 to 3
        {m, SL[]} = LSL[r]; // randomly selects
                               one from LSL
}
```

Sequence Diagram :-



N Self Checking Architecture :-



Note :- AT_1, AT_2, AT_3 refers as Acceptance_Test_1, Acceptance_Test_2, Acceptance_Test_3

Pseudo code :-

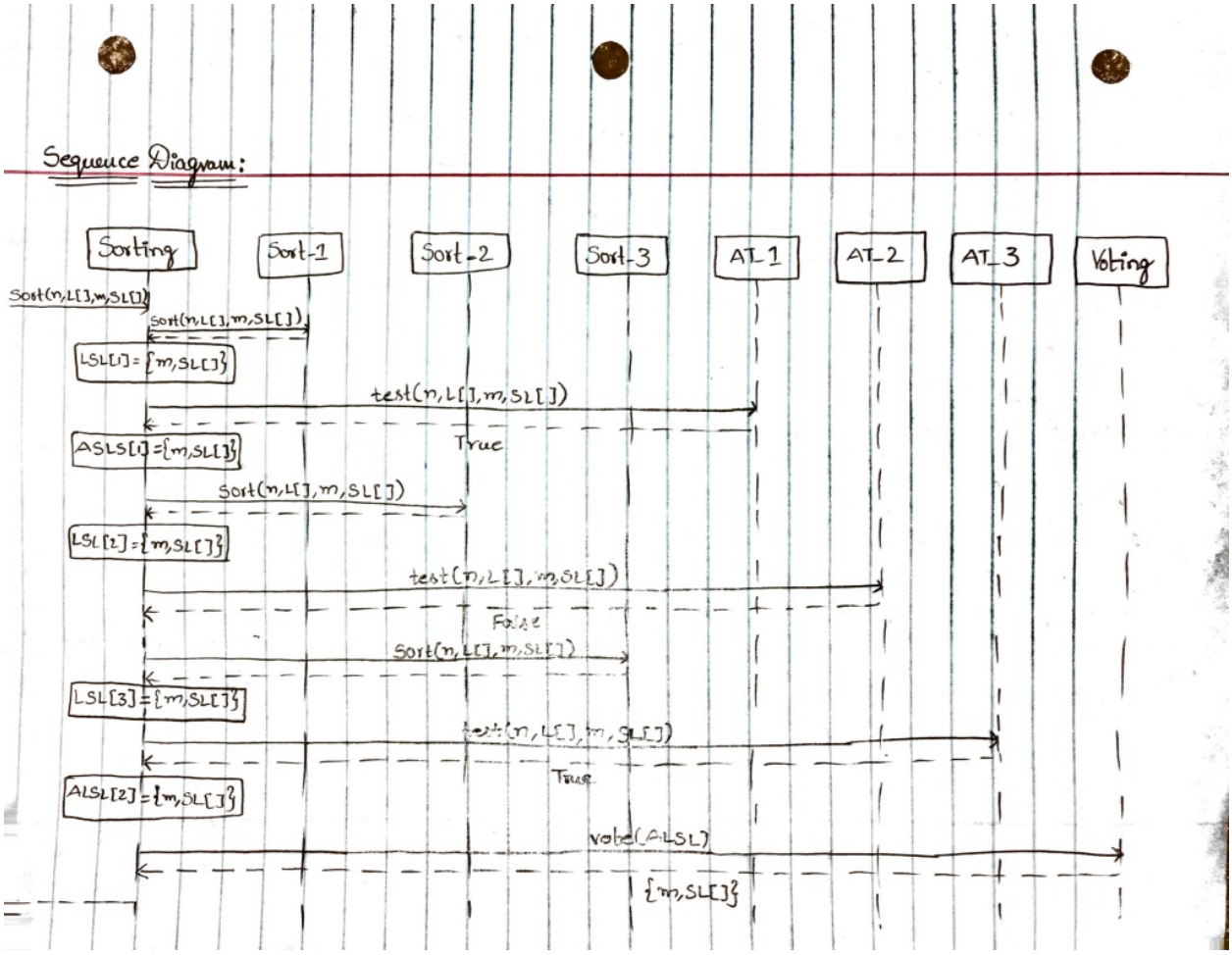
```
Class Sorting
    Sort *s
    voting *v
    Acceptance_Test *at
    Void Sort(in int n, int L[]; out int m, int SL[]){
        LSL[] is an array of {m,SL[]} list
        ALSL is an array of {m,SL[]} list
        Sort S[];
        Acceptance_Test at[];
        S[1] = new Sort_1();
        AT[1] = new AT_1();
        S[2] = new Sort_2();
        AT[2] = new AT_2();
        S[3] = new Sort_3();
        AT[3] = new AT_3();
        K=0;
        for i=1 to 3
            S[i].Sort(n,L[],m,SL[]);
        LSL[i] = {m,SL[]}
        testResult = at.test(n, L[], m, SL[]);
        if(testResult == True)
            K=K+1 ; // Increment the acceptable results
            ALSL[K] = {m,SL[]};
        ENDIf
    ENDFor
    if(K==0)
        r = Rand(1,3) //generate random numbers b/w 1 to 3
        {m,SL[]} = LS[r] //randomly select one from LSL and return
    else
        {m,SL[]} = v.voting(ALSL, K);
}
```

```
Class Voting:
  {int, int[]} vote(in: ALSL, K){
   if K==3   //if all results pass their acceptance tests
          if ALSL[1] == ALSL[2]
                return ALSL[1]
          elseif ALSL[2] == ALSL[3]
                return ALSL[2]
          else if ALSL[1] == ALSL[3]
                return ALSL[3]
          ENDIF
          r= Rand(1, 3)  // generate random numbers b/n 1 to 3
          return ALSL[r] // randomly select one from ALSL and return
   else if K = 2 then    //if two results pass their Acceptance Test
          if ALSL[1] = ALSL[2] then
                return ALSL[1]
          ENDIF
          r=Rand(1, 2) // select either 1(or)2 randomly
          return ALSL[r]
   else if K==1 then // if only one passes acceptance test
          return ALSL[1] // return only acceptable result
   ENDIF
  }
```

```
Class AT_1 :
  boolean test(in: int n, int L[]; out: int m, int SL[]){
   if(m>n)
        return False;
   else if(m!=SL.length())
        return False;
   else{
        for i=0 to m
              for j=1 to m
                    if((SL[i]==SL[j] && i!=j)
                          return False;
        }
   return True;
  }
```

```
Class AT_2 :
  boolean test(in: int n, int L[]; out: int m, int SL[]){
   if(m>n)
        return False;
   else if(m!=SL.length())
        return False;
   else{
        for i=0 to m
              for j=1 to m
                    if((SL[i]==SL[j]&& i!=j)
                          return False;
        }
   return True;
  }
```

```
Class AT_3 :
  boolean test(in: int n, int L[]; out: int m, int SL[]){
   if(m>n)
        return False;
   else if(m!=SL.length())
        return False;
   else{
        for i=0 to m
              for j=1 to m
                    if((SL[i]==SL[j]&& i!=j)
                          return False;
        }
   return True;
  }
```


Sequence Diagram:
Sorting
Sort-1
Sort-2
Sort-3
AT_1
AT_2
AT_3
Voting
Sort(n,L[],m,SL[])
sort(n,L[],m,SL[])
LSL[1]={m,SL[]}
test(n,L[],m,SL[])
True
ASLS[1]={m,SL[]}
sort(n,L[],m,SL[])
LSL[2]={m,SL[]}
test(n,L[],m,SL[])
False
sort(n,L[],m,SL[])
LSL[3]={m,SL[]}
test(n,L[],m,SL[])
True
ALSL[2]={m,SL[]}
vote(ALSL)
{m,SL[]}