Algorithm complexity (N1 = ShowExit)

Having two conditionals, I went with the algorithmic complexity of the else (Which is the longest)

```
private String showExit() {
    String msg = ""; // 1

    PriorityQueueNode neNodeE;
    PriorityQueueNode eNodeE;

    if(operationManually==1){ // 1

      int nEPassengersQueueInt = nEpassengersExit.size();

      int ePassengersQueueInt = ePassengersExit.size();

      for(int i=0;i<nEPassengersQueueInt;i++){
        neNodeE = new
PriorityQueueNode<>(nEpassengersExit.getHead().getItem(),calculateExitNEPassengers(
nEpassengersExit.getHead().getItem(), i + 18));
        nePassengerEntrance.insert(neNodeE);
        nEpassengersExit.dequeue();
      }

      for(int i=0; i<ePassengersQueueInt;i++){
        eNodeE = new PriorityQueueNode<>(ePassengersExit.getHead().getItem(),
calculateExitEPassengers(ePassengersExit.getHead().getItem(), i));
        ePassengerEntrance.insert(eNodeE);
        ePassengersExit.dequeue();
      }
```

```
        msg = "-----Exit order-----\n";


        int passEExit = ePassengerEntrance.occupedSize();

        int passNExit = nePassengerEntrance.occupedSize();



        for (int i = 0; i < passEExit; i++) {

           EPassenger passenger =ePassengerEntrance.maximum().getElement();

           ePassengerEntrance.extractMax();

           if(passenger.isPreference()) {

              msg += i + 1 + ". " + passenger.getName() + " " + passenger.getSeat() + " " + " |
Presenta discapacidad |" + "\n";


           }

           else {

              msg += i + 1 + ". " + passenger.getName() + " " + passenger.getSeat() + " " + " |
No presenta discapacidad |"+"\n";


           }



        }

        msg += "-----Exit order non executive-----\n";


        for (int i = 0; i < passNExit; i++) {

           NEPassenger passenger =nePassengerEntrance.maximum().getElement();

           nePassengerEntrance.extractMax();
```

```
            msg += i+1 + ". " + passenger.getName() + " " + passenger.getSeat() +  " " + " | 
No presenta discapacidad | "+ "\n";


        }
    }else{
        ePassengersQueue = new Queue<>(); // 1
        nEpassengersQueue = new Queue<>(); // 1


        for (NEPassenger nePassenger : nePassengers) { // n
            Node<NEPassenger> p = new Node<>(nePassenger); // n - 1
            nEpassengersQueue.enqueue(p.getItem()); // 6(n-1)
        }
        int nEPassengersQueueInt = nEpassengersQueue.size(); // 1
        for (EPassenger ePassenger: ePassengers){ // n
            Node<EPassenger> p = new Node<>(ePassenger); // n - 1
            ePassengersQueue.enqueue(p.getItem()); // 6(n-1)
        }
        int ePassengersQueueInt = ePassengersQueue.size(); // 1


        for(int i=0;i<nEPassengersQueueInt;i++){ // n
            neNodeE = new 
PriorityQueueNode<>(nEpassengersQueue.getHead().getItem(),calculateExitNEPassenge
rs(nEpassengersQueue.getHead().getItem(), i + 18)); // n -1
            nePassengerEntrance.insert(neNodeE); // n -1
            nEpassengersQueue.dequeue(); // 5(n-1)
        }


        for(int i=0; i<ePassengersQueueInt;i++){ // n
            eNodeE = new PriorityQueueNode<>(ePassengersQueue.getHead().getItem(),
calculateExitEPassengers(ePassengersQueue.getHead().getItem(), i)); // n - 1

            ePassengerEntrance.insert(eNodeE); // n - 1

            ePassengersQueue.dequeue(); // 5(n-1)

        }


        msg = "-----Exit order-----\n"; // 1


        for (int i = 0; i < ePassengersQueueInt; i++) { // n

            EPassenger passenger =ePassengerEntrance.maximum().getElement(); // n - 1

            ePassengerEntrance.extractMax(); // (2n + 4)(n-1) = 2n^2 - 2n + 4n - 4 = 2n^2 +
2n - 4

            if(passenger.isPreference()) { // n - 1

                msg += i + 1 + ". " + passenger.getName() + " " + passenger.getSeat() + "
Presenta discapacidad" + "\n"; // n -1


            }

            else {

                msg += i + 1 + ". " + passenger.getName() + " " + passenger.getSeat() + " No
presenta discapacidad"+"\n";


            }

        }


        for (int i = 0; i < nEPassengersQueueInt; i++) { // n

            NEPassenger passenger =nePassengerEntrance.maximum().getElement(); // n -1

            nePassengerEntrance.extractMax(); // (2n + 4)(n-1) = 2n^2 - 2n + 4n - 4 = 2n^2 +
2n - 4
```

```
            msg += i + 19 + ". " + passenger.getName() + " " + passenger.getSeat() + "\n"; //
n - 1

        }

      }



      return msg; // 1

    }
```

T(n) = 1 + 1 + 1 + 1 + n + (n-1) + 6(n-1) + 1 + n + (n-1) + 6(n-1) + 1 + n + (n-1) + (n-1) + 5(n-1) + n + (n-1) + (n-1) + 5(n-1) + 1 + n + (n-1) + 2n^2 + 2n - 4 + (n-1) + (n-1) + n + (n-1) + 2n^2 + 2n - 4 + (n-1) + 1

T(n) = 4n^2 + 16n - 8

ENQUEUE = T(n) = 6

```
  public Node<E> dequeue(){

          if(isEmpty())

          {

                  return null;

          }else

          {

          Node<E> aux = head;

          head = head.getNext();


          size--;


          return aux;
```

```
        }
    }
```

DEQUEUE = T(n) = 5

```
    public void enqueue(E e){
        Node<E> nodeVVVVVVV = new Node<E>(e);


        if (isEmpty())
        {
            head = nodeVVVVVVV;
            tail = head;
        }
        else
        {
          tail.setNext(nodeVVVVVVV);
        }
            tail = nodeVVVVVVV;


            size++;
    }
```

Algorithm complexity (N2 = ShowEntrance)

```
private String showEntrance() {
        String msg;
        PriorityQueueNode neNode;
        PriorityQueueNode eNode;
```

```
    if(operationManually==1){ // 1
      int c=0;


      while(!nEpassengersQueue.isEmpty()){
        c++;
        neNode = new
PriorityQueueNode<>(nEpassengersQueue.getHead().getItem(),calculateEntranceNEPass
engers( nEpassengersQueue.getHead().getItem(),c));
        nePassengerEntrance.insert(neNode);


        nEpassengersQueue.dequeue();
      }
      int b=0;
      while(!ePassengersQueue.isEmpty()){
        b++;
        eNode = new PriorityQueueNode<>(ePassengersQueue.getHead().getItem(),
calculateEntranceEPassengers( ePassengersQueue.getHead().getItem(), b));
        ePassengerEntrance.insert(eNode);


        ePassengersQueue.dequeue();
      }


      msg = "-----Entrance order-----\n" +
          "Executive/Disabled group\n" +
          "Please present yourself in the respective order\n\n";
      int ePassengerEntranceInt = ePassengerEntrance.occupedSize();
      for (int i = 0; i < ePassengerEntranceInt; i++) {
        EPassenger passenger =ePassengerEntrance.maximum().getElement();
```

```
            ePassengerEntrance.extractMax();

            if(passenger.isPreference())

                msg += i + 1 + ". " + passenger.getName() + " " + passenger.getSeat()+ " " +
"DISCAPACIDAD" + " " + "miles: " + passenger.getMiles()+ "\n";

            else msg += i + 1 + ". " + passenger.getName() + " " + passenger.getSeat()  + " " +
"miles: " +passenger.getMiles()+ "\n";



        }


        msg += "------------------------\n" +

                "Economy group\n" +

                "Please present yourself in the respective order\n\n";


        int ePassengerEntranceint = nePassengerEntrance.occupedSize();

        for (int i = 0; i < ePassengerEntranceint; i++) {

            NEPassenger passenger =nePassengerEntrance.maximum().getElement();

            nePassengerEntrance.extractMax();

            msg += i + ") " +passenger.getName() + " " + passenger.getSeat() + "\n";


        }



    }else{

        ePassengersQueue = new Queue<>();

        nEpassengersQueue = new Queue<>();


        for (NEPassenger nePassenger : nePassengers) {
            Node<NEPassenger> p = new Node<>(nePassenger);

            nEpassengersQueue.enqueue(p.getItem());

        }


        for (EPassenger ePassenger: ePassengers){

            Node<EPassenger> p = new Node<>(ePassenger);

            ePassengersQueue.enqueue(p.getItem());

        }

        int c = 0;

        while(!nEpassengersQueue.isEmpty()){

            c++;

            neNode = new
PriorityQueueNode<>(nEpassengersQueue.getHead().getItem(),calculateEntranceNEPass
engers( nEpassengersQueue.getHead().getItem(),c));

            nePassengerEntrance.insert(neNode);


            nEpassengersQueue.dequeue();

        }

        int b=0;

        while(!ePassengersQueue.isEmpty()){

            b++;

            eNode = new PriorityQueueNode<>(ePassengersQueue.getHead().getItem(),
calculateEntranceEPassengers( ePassengersQueue.getHead().getItem(), b));

            ePassengerEntrance.insert(eNode);


            ePassengersQueue.dequeue();

        }
        msg = "-----Entrance order-----\n" +
                "Executive/Disabled group\n" +
                "Please present yourself in the respective order\n\n";
        int ePassengerEntranceInt = ePassengerEntrance.occupedSize();
        for (int i = 0; i < ePassengerEntranceInt; i++) {
            EPassenger passenger =ePassengerEntrance.maximum().getElement();
            ePassengerEntrance.extractMax();
            if(passenger.isPreference())
                msg += i + 1 + ". " + passenger.getName() + " " + passenger.getSeat()+ " " +
"DISCAPACIDAD" + " " + "miles: " + passenger.getMiles()+ "\n";
            else msg += i + 1 + ". " + passenger.getName() + " " + passenger.getSeat()  + " " +
"miles: " +passenger.getMiles()+ "\n";



        }


        msg += "------------------------\n" +
                "Economy group\n" +
                "Please present yourself in the respective order\n\n";


        int ePassengerEntranceint = nePassengerEntrance.occupedSize();
        for (int i = 0; i < ePassengerEntranceint; i++) {
            NEPassenger passenger =nePassengerEntrance.maximum().getElement();
            nePassengerEntrance.extractMax();
            msg += i + ") " +passenger.getName() + " " + passenger.getSeat() + "\n";


        }
```

```
      }



      return msg;
   }
```

We come to the conclusion that the algorithmic complexity for the two methods is the same because the same format is used in both, only organized in a different way, therefore, I am going to skip the calculation and put the same equation.

T(n) = 1 + 1 + 1 + 1 + n + (n-1) + 6(n-1) + 1 + n + (n-1) + 6(n-1) + 1 + n + (n-1) + (n-1) + 5(n-1) + n + (n-1) + (n-1) + 5(n-1) + 1 + n + (n-1) + 2n^2 + 2n - 4 + (n-1) + (n-1) + n + (n-1) + 2n^2 + 2n - 4 + (n-1) + 1

T(n) = 4n^2 + 16n - 8