The lift project:

1/ introduction

For ECM1414, Data Structures and Algorithms, I needed to create an algorithm that would make an elevator as efficient as possible. This was compared to a base case, a naïve algorithm, which it had to beat.

In my approach I judged the efficiency of both algorithm by the number of floors changes the elevator must perform of the overall process. Since the cost in time, of people entering and leaving, are minimal compared to the time it takes for the lift to climb multiple floors.

In my approach, to compare these algorithms, I retrieved statistical data from different scenarios, where both algorithms, naïve and improved, would compete with the same data. Each of the scenarios would be performed a hundred times and averaged out in order to create significant and "stable" data. The starting location of each person as well as the size of the building will be tested within each algorithm identically.

Before, we must set a few global parameters that are the same for each scenario:

-People have a random floor start and destination floor.

-People will only get in the lift if it is going in the correct direction.

-The lift has a max capacity and when reached people can no longer get in.

-Are waiting at the same time for the lift

-the experiment is finished once all people in the building have reached their destination floor, and the final score is the amount of floor changes it had to perform to complete this experiment.

The scenarios are the following:

-Firstly, The Minimal scenario. In this case the number of floors is small and resembles one of an apartment building. In addition, the number of people waiting are lesser. This test the efficiency of the algorithm handling small systems. This is where the mechanical lift fails as doing extra floors it does not scale well with

-Next, the Many People Little Floors scenario. As the title indicates, there will be more people than floors and thus we can expect there to be at least a person per floor. The size of the building is medium sized (approx.: 10 to 20). This will test the management of the capacity of the elevator. It should bring people as quickly as possible to their destination in order to free up space in it.

-The opposite scenario to the precedent, Little People Many Floors, is as expected a situation where there will be many empty floors and people scattered around them. This will

test the new algorithms ability to minimise the movement between each waiting person as well as their destination. This is where the old algorithm was the weakest as it would do extra steps (such as moving to the top) even if no one were going there.

Finally I have implemented a graphical log to view every step the lift takes, which looks like the following diagram:

```
###################### - MinMax Elevator - #################
5# #
4# #
3# #
2# #
1# U4 #                                    # -1 # \/
0# #
#########################################################################
####
```

Legend:

\/ = the lift is going down

-1 = someone in the lift has left it

U4 = someone on that floor wants to go up to floor 4

1#,2#.. = numbers of the floor

By having a frame by frame diagram of the algorithm's performance, we can see its behaviour perfectly and makes debugging and improving it easy.

2/The algorithms: The first algorithm is very simple, and we will refer to it as the mechanical elevator algorithm. It simply makes the elevator go up and down and change direction when reaching the top or bottom floor of the building. It is the base case and the algorithm to beat.

```
public void move(){

        commonMove();
        _isGoingUp = isGoUpAtExtremities();
    }
```

It is made up of two parts, the function that defines its movement relative to its direction:

```
void commonMove(){
        if (_currentFloor < _maxFloors && _isGoingUp) {
```

```
            _currentFloor += 1;
            _distance += 1;
        //   System.out.println("Move up to floor:" + _currentFloor);
        }

        if (_currentFloor > 0 && !_isGoingUp) {
            _currentFloor -= 1;
            _distance += 1;
            // System.out.println("Move down to floor:" + _currentFloor);
        }
    }
```

And its direction from when it reaches the top or bottom of the building:

```
_isGoingUp = isGoUpAtExtremities();
```

```
boolean isGoUpAtExtremities(){
        if (_currentFloor == _maxFloors) {
            System.out.println("Change direction to go down");
            return false;
        }

        if (_currentFloor == 0) {
            System.out.println("Change direction to go up");
            return true;
        }

        return _isGoingUp; // no change
    }
```

The second algorithm, called throughout the code as the MinMaxElevator, takes inspiration from the first one by copying its movement. It goes up and down between two points in the building and will turn around when it reaches it. The improvement in efficiency is made by reducing the floor and ceiling if moving there is unnecessary. For example, in a 5 story building, if the person to pick up is on floor 3, going down, he will not have to go to floor 5 to turn around and pick him up. Ideally he should arrive at floor 3, turn around, and pick him up(reminder people will only get in if the elevator's direction is the same as theirs, hence the person not getting in when the lift is going upwards).

This reduction is made by calculating the maximum and minimum floors it needs to go to between the people waiting's start destination and the people in the lift's destination.

It should be noted that this change only modifies the direction. The commonMove () function from above is once again used as it only moves the elevator in the direction its facing

Each following function will get respectively, The maximum of t

```
int getMaxPeopleWaitingStart(){
        int myMax = 0;

        for(int i=0; i<_PeopleWaiting.size();i++){
            myMax = Math.max(myMax ,_PeopleWaiting.get(i).start);
        }
        System.out.println("getMaxPeopleWaitingStart: "+ myMax);
        return myMax;
    }

    int getMaxPeopleInLiftDestination(){
        int myMax = 0;

        for(int i=0; i<_PeopleInLift.size();i++){
            myMax = Math.max(myMax ,_PeopleInLift.get(i).destination);
        }
        System.out.println("getMaxPeopleInLiftDestination: "+ myMax);
        return myMax;
    }
    int getMinPeopleInLiftDestination(){
        int myMin = _maxFloors;

        for(int i=0; i<_PeopleInLift.size();i++){
            myMin = Math.min(myMin ,_PeopleInLift.get(i).destination);
        }
        System.out.println("getMinPeopleInLiftDestination: "+ myMin);
        return myMin;
    }
    int getMinPeopleWaitingStart(){
        int myMin = _maxFloors;

        for(int i=0; i<_PeopleWaiting.size();i++){
            myMin = Math.min(myMin ,_PeopleWaiting.get(i).start);
        }
        System.out.println("getMinPeopleWaitingStart: "+ myMin);
        return myMin;
    }
```

Then to calculate the first boundary, the ceiling, we retrieve the maximum between the peopleWaiting.start and the maximum of the peopleInLift.finish. This is the same to calculate the minimum as we take the minimum of the two lists. Finally we just need to make sure that the lift stays in the building , so it stays always between floor 0 and MaxFloors.

```
boolean isGoUpAtExtremitiesMinMax(){
```

```
        if (_currentFloor == Math.min(_maxFloors,Math.max(getMaxPeopleInLiftDe
stination(), getMaxPeopleWaitingStart()))) {
            System.out.println("Change direction to go down");
            return false;
        }

        if (_currentFloor <= Math.max(0,Math.min(getMinPeopleInLiftDestination
(), getMinPeopleWaitingStart()))) {
            System.out.println("Change direction to go up");
            return true;
        }

        return _isGoingUp; // no change
    }
```

III/The tests

I used Junit testing in order to perform these tests.

1) Minimal scenario:
   Max floors = 5
   Total people = 2, random starts and destination
   Lift Capacity = 10

   This test is to judge the efficiency of such an algorithm in basic scenarios. In the best case it should bring the people to their destination with the least unnecessary steps possible.

   Results:

```
Average Distance for MinMax after [100] executions [5]
Average Distance for Mechanical after [100] executions [7]
```

Analysis: The new algorithm seems to perform 30 percent faster. Which is a lot of time to save when this kind of system is performed multiple times in a day.

2) the Many People Little Floors scenario
   Max floors = 10
   Total people = 100
   Lift capacity = 10

   The main difference in efficiency here should come from the management of the lift capacity as this is what will accelerate the process of the system. However, it is to note that the Mechanical lift already has an efficient way of managing this at it only picks up people going in the same direction as it

Average Distance for MinMax after [100] executions [59]

Average Distance for Mechanical after [100] executions [69]

Analysis: As this only minimises the boundaries it will probably do identical movements to the mechanic al lift in the first part of the experiment. That being said, once their start being floors with no people, It sa ves traveling distance by not going to these floors.

3) Little People Many Floors scenario

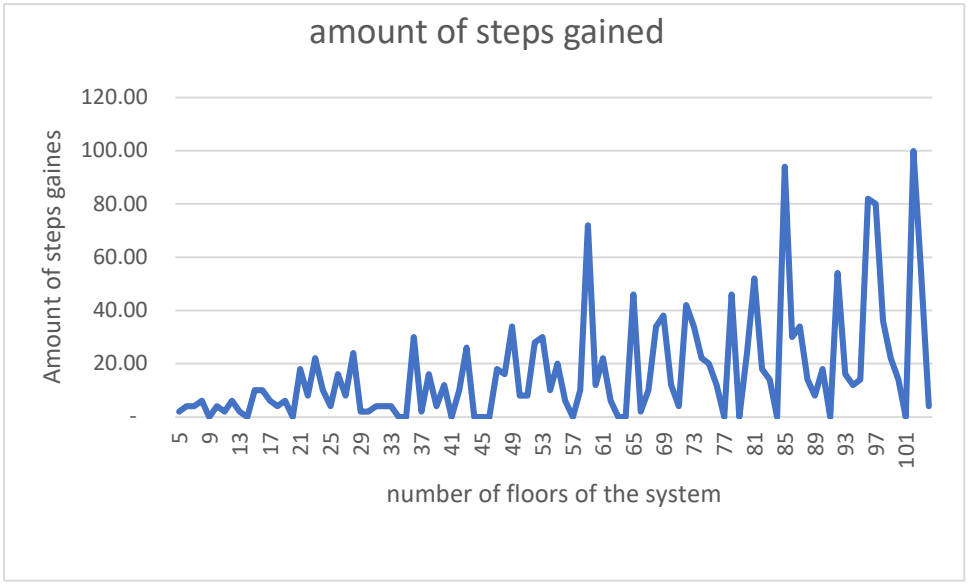
Max floors = 100
Total people = 15
Lift capacity = 7

This tests if the algorithm can minimise movement between each person and their destination. If it also has a good way of managing its capacity, then it can win a lot of time as this would reduce the distance it must perform by a lot.

Average Distance for MinMax after [100] executions [188]

Average Distance for Mechanical after [100] executions [195]

Analysis: This is the least impressive Success as the MinMax lift's way of managing its capacity is only du e to it being more efficient than the old one. If it had a better management system, such as getting peopl e whose destinations are closer first then the success margin would have been much greater



Finally I modified the code of the minimal scenario so that every iteration will add a floor to the building. I also modified it so that it would print suitable data so that it can be spliced into an excel file and made in

to the chart above (log data and excel file are in the submission folder.) This shows that MinMax Algorithm is increasingly more performant than the Mechanical algorithm the more floors there are to the building. Interestingly it also shows that this algorithm is either the same or more efficient in the old algorithm, since at worst case, It does the same movements as the mechanical algorithm, thus performing identically.

Because of bad management of project time from my part, I cannot improve my code in order to increase the algorithms efficiency, I will therefore suggest some improvements ideas instead of implementing them:

-the lift should pick up in priority people who are getting off nearby. This should leave the people who have a long distance to travel near the end which means they will not clog up the elevator

-the lift can pick up people getting off in a minimum amount of stops even though it is not facing the going in the right direction. This minimum should be relative to the total amount of floors and amount of people still in the building (this should be tweaked and tested in order to find the optimal solution.)

IV/ Weekly log

Approx. week 4: creation of the framework using bad java code (as I had not been taught it yet)

-implementation of the Mechanical Elevator

Approx. week 5: -bug fixing and creation of a function that would not allow the people to have the same start and destination

- Added feature: people will only get in if the elevator is going in the right direction.

Week 9 :- new control algorithm research and Brainstorming,

- rewriting the entire framework into better written code

Week 10: -implementation of first attempt of a new algorithm Democratic Elevator (source code is still in file)

-realising that the Democratic elevator does not work unless it has many people in.

-Does not work because it only picks up people going in one direction

Week 11:- Democratic elevator scrapped

-renaming the functions.

Week 12: -Implemented Junit texting instead of main.java

Week 13:-writing down the introduction and outline of my submission

Week 14: Implementation of ServantElevator algorithm (Note: source code is no longer in files as MinMaxElevator was built on top of it,).

-implementation of MinMaxElevator as a successor to ServantElevator.

Week 15:-fix of MinMaxElevator because the new ceiling and floor collided and the elevator could therefore bypass them.

-write up of Submission (creation of graphs, testing, scenarios.s)