The algorithm will check if any elevator is on this floor, if so, it assigns the elevator.
If not, it checks whether the user wants to reach an upper or lower floor from his current floor.
He then assigns the elevator that ascends and approaches the same floor
Each elevator maintains its data structure, the element tax is the maximum stop tax of the elevator
- It is possible to assign end cases and set that during rush hours such as morning in the office building the available elevators will wait on the 0th floor

And in the afternoon in the residential building the available elevators will wait on the 0th floor
the idea of the algoritem:
We will take the first call in the array and check which elevator can reach it in the minimum time:
The calculation of the minimum time for the arrival of the elevator will be made according to the calculation of the time of arrival to the reading floor + the time of stopping on the floor + the time of travel to the destination floor + the time of stopping on the floors on the way.
If the elevator is already set for the task we will only refer to it if the task is in the same direction of the reading and the source floor is in the path of the elevator.
Once the elevator has reached the maximum floor in the direction it is traveling we will redefine it as open to readings and refer to it any relevant reading
We will run the test on all the elevators and select the elevator that returns the lowest output.
We will go through all the readings in the list and add readings to the elevators that are set to travel in the same direction. For each reading we will check which elevator is most suitable according to the algorithm described in 2 and to which we will add the reading.
After practicing a lift call we deleted it from the original array.
We will continue until the end of the reading list
The algorithm will return a list of elevators with task assignment

- Check the floor is bigger then exist
- Check if the name of building/min floor/max floor/number of elevator not null

Test my elevator-
- Check if the time for open/close/stop is not null

Test for calls-
- Check which elevator this call was assigned to
- Check if the value of the variable not null

Test for system
- Check if it's match the correct elevator to given call
- That the calls not go out of bound of the arry

- the scenario that we are facing in the smart elevator is mostly the time destination that

  the elevator most needs to go through it

- the solution that we are giving here is improving the elevator algorithm

  The senario that we can face :
  Scenario 1 There are 2 people ordering an elevator at that moment and they are on different floors. Both are routed to the same elevator, one person has ordered the elevator and he is holding it and at this time the other person is waiting for the elevator. Should he be allowed to wait for the elevator? Should I give the elevator an order that after a certain time another elevator will be routed to it?
  • Another scenario if there is congestion in the elevators and there is not enough allocation for people let’s say congestion time on a known day such as company elevators that usually work from 9-17. How to optimize people's waiting time
  • How many people order an elevator in different places? How will the elevator know when there is no more space and assign a certain tax of people (are there other people with them like family and only one ordered, carrying shopping, etc.).

T**<u>he Algorithm in pseodo-cod</u>**

```
match_elevator(B : Building.json,C : Calls.csv, O : output.csv)
     read C to call_list
     read B[“elevators”] to building.elevator_list
     for c in call_list
          for b in building.elevator_list
               calculate the current position of b
               calculate the time will take this call
               b* = the elevator with the minimal time
          b* = c.allocted_elevaotr
     write call_list to output.csv
```