The three algorithms are designed in such a way that each one is an improvement on the other.

**Algorithm 1**

* classes:

elevatorFast

* inputs:

unsigned int Totalfloors, numberofPersons;

unsigned int timeBtnfloor; unsigned int startLoc, unsigned int nofElevators;

* functions:

void randomNumberGenSlow (int numberOfrequests[])

🡪 generates random number of requests

processRequestSlow(numberOfrequests, timeBtnfloor, startLoc)

🡪 **processes all the requests on first come first serve basis**

* Problems Faced:

🡪passing arrays between functions

🡪elevator not considering weather start location greater than the number of floors.

🡪elevator too slow in processing requests.

**Algorithm 2**

* classes:

elevator

* inputs:

unsigned int Totalfloors, numberofPersons;

unsigned int timeBtnfloor; unsigned int startLoc, unsigned int nofElevators;

functions:

void randomNumberGenSort(int numberOfrequests[])

🡪 generates random number of requests.

-🡪 sorts the requests in an order to process them accordingly

processRequestSort(numberOfrequests, timeBtnfloor, startLoc)

🡪 **processes all the requests on a sorted basis where the lower floor requests are solved first.**

* Problems Faced :

🡪The elevator was slow but a sort algorithm did the trick.

🡪elevator solving lower floor requests first even if elevator starts on top floor.

**Algorithm 3**

* classes:

elevatorfaster

* inputs:

unsigned int Totalfloors, numberofPersons;

unsigned int timeBtnfloor; unsigned int startLoc, unsigned int nofElevators;

* functions:

void randomNumberGen(int numberOfrequests[])

🡪 generates random number of requests.

-🡪 sorts the requests in an order to process them accordingly

processRequests(numberOfrequests, timeBtnfloor, startLoc)

🡪 **processes all the requests on a sorted basis where the priority is given to the floors nearer to the elevator start location.**

Problems Faced:

🡪 elevator was not considering the start location and working accordingly, but a small classification based on its position did the trick and elevator working in consideration with the start location.

**Main Function:**

* Here, the objects are created dynamically basing on the count of the elevators.
* The dynamically created objects are used to access and call the members of the **elevatorfaster** class.
* Checks if the entered inputs are valid.
* Each elevator or object chooses an algorithm to execute the requests randomly and displays the desired results.
* Logs all the results to a file.

**To Execute and run the code please type the following commands in the terminal**

* **make**
* **./alg3**
* **Enter the required values that are asked.**
* **The output is displayed in the terminal as well as logged into an output file.**