Data Structures used:

1. all_info: A 2-D vector which stores all the inputs in the form of 1-D vectors [time, origin floor, destination floor]
2. ready: a queue which stores 1- D vectors in the form of [time, origin floor, destination floor]. This queue stores people who can enter the lift while it is going in a particular direction.
3. completed: a set which stores 1-D vectors as [time, origin floor, destination floor]. This set stores people who have already entered the lift.
4. isDest: a 1-D vector which stores binary values for all the floors. This vector informs us if a particular floor is a destination floor of any person in the lift or not.

Variables:

1. current_time: tells what the current time is since starting
2. current_floor: tells which floor the elevator is on currently.

Assumptions:

1. If a floor ‘x’ is the origin floor of a person ‘p’, it cannot be the origin floor for any other person, till ‘p’ reaches its destination.
2. If a floor ‘x’ is the destination floor of a person ‘p’, it cannot be the destination floor for any other person, till ‘p’ reaches its destination.
3. However, a floor ‘x’ can be the origin for a person ‘p1’ and the destination for a person ‘p2’ simultaneously and vice versa.
4. When the door of the elevator is open, only 1 person can come in or go out at a single time. For 2 people to come in and go out, the elevator has to open and close 2 times.
   (NOTE: 2 is specified with respect to point number 3).
5. Origin floor and destination floor are different.

Algorithm:

Overview: The easiest way to implement this is using the FCFS approach. I first used the FCFS approach and the results were not promising. The response and turnaround time taken was high. I then modified the FCFS approach. The new approach determines the elevator to go in a particular direction(upward or downward) and serve all the requests of that direction. Once all the requests in this direction are served, reverse the direction and again serve all the requests in this direction.

Detailed approach:

1. Standard input for the number of queries is taken, and then standard input for each query is taken.
2. The algorithm runs till all the queries are not processed, that is, till all the people are not served by elevator.
3. Sort the all_info vector in increasing order according to time difference between queries and current time. While the current time is less than the nearest query(nearest by time and not by distance), stay at the current position and increase current time by 1.

4. When current time becomes more than or equal to the time of the nearest query, move to the origin floor of the nearest query. Determine whether this direction is upwards or downwards.
5. While the elevator is moving from every floor, check if the floor it is on currently, has a person to be served. If there is not any such person, move to the next floor. If there is such a person, check if the person's destination is in the direction of the elevator, or in the opposite direction of the elevator. If it is in the opposite direction, move to the next floor. If it is in the direction of the elevator, and the person has pressed the elevator button before current time, stop the elevator and pick up this person. This particular index of all_info moves into the 'ready' queue and 'completed' set. isDest vector is updated at the destination floor of this person, and set to 1.
6. Also, while going from every floor, check if this particular floor is the destination of any person in the elevator. This is checked by isDest vector. If it is one, then open the elevator and close it, then change its isDest value to 0. If it was already 0, keep moving.
7. Process all the queries in a particular direction using steps 5 and 6 and the 'ready' queue. When the ready queue is empty, then start from step 3.