Problem 1: The Launch Day puzzle → project scheduling

Example I/P -:

Number of projects: N = 3

projects: (Deadline, profit) = (2,100), (1,19), (2,27)

Schedule to

Maximize total profit, respecting the deadlines

Expected O/P: 127

⇒ Approach -: 1. we will sort the projets by profit in desc. order. (bcoz we need to have projets with high profit)

2. Starting with most profit try to schedule it.
   - Try to schedule on latest available day on or before its deadline.

So we have day 1 - (2, 27) } Total profit = 127
day 2 - (2, 100)

with high profit (2,100) (2,27) (1,19)

pseudo code -:

```
max_profit ( list<project > projects) {
    projects.sort ((a,b) -> b.profit - a.profit);
    int MaxDeadline = 0;
for( projects p: projets) {   if(p.deadline > maxDeadline){
        maxdeadline = p.deadline;
    }
// To track day which is used
    boolean [] schedule = new boolean[Maxdeadline + 1];
int totalprofit = 0;
for (projects p: projets) {
        for( int day = p.deadline; day >= 1; day - -){
if (! schedule[day]) {
            schedule[day] = true;
        totalprofit += p.profit;
            break;
    }
    return total profit
```

dry run :-

Sorts in descending order: (2,100) (2,27) (1,19)

* day = p.deadline
  // check if condition.
  if(!schedule[day]) ✓ = it will become true
  schedule[day] = true // Means now it is occupied
  total profit += 100;
* Now, for day[1] = project(2,27)
  Schedule[2] == true → already taken
  Try day[1] → schedule[1] == false (if cond. become false)
  schedule[1] = true (Now occupied)
  (Total profit += p.profit)
  100 += 27
  = 127

day--

* Project (1,19)
  schedule[2] → occupied
  " [1] → occupied

  So no days left - skip

Total profit = 127 Ans

Problem 2:- The VIP commute crisis - Toll Gate optimisation

⇒ Example I/P:- T = [0, 1, 2, 3]

VIP = [0, 1, 0, 1]

W = 2

Approach:- 1) Add arriving cars in queue

2) If any VIP is waiting more than W, process that VIP.

3) otherwise, process the front car in queue

4) Add car's waiting time to running total.

5) Move to next minute (recursive call)

pseudo code:-

```
int totalwait = 0;
Static queue<car> queue = new LinkedList<>();
processCarRec(List<car> cars, int index, int time, int w){
  if (index >= car.size() && queue.isEmpty()) return;

while(index < car.size() && car.get(index).arrivalTime <=time)
  {
queue.add(car.get(index));
}                    index++;
 car selectCar = null;
for(car car: queue){
  if(car.isVIP && (time - car.arrivalTime > w)){
    selectedCar = car;
}          break;
 if(selectedCar == null && !queue.isEmpty()){
     selectedCar = queue.peek();
}
if(selectedCar != null){
        queue.remove(selectedCar);
    totalwait += time - selectedCar.arrivalTime;
    }

    processCar(cars, index, index+1, w);
}
```

Dry Run:-

T = [0, 1, 2, 3] → arrival time

VIP = [0, 1, 0, 1] → VIP flags (1 = VIP, 0 = regular)

W = 2

Minute 0 :-

car 0

queue[0]

car is not VIP

Process :-

waittime = 0 − 0 = 0 (arrival time - current time)

Totalwaittime = 0 ;

Queue [ ] - become empty

Recurse to time = 1

Minute 1 :-

car 1

queue[1]

It is a VIP car (true)

(time - car.arrival > w) ⇒ 1 − 1 > 2 (not need to select)

- totalwait = 0;
- Queue [ ]
- Recurse to time = 2

Minute 2; car 2 , queue[2] , VIP → no.

wait time = 2 − 2 = 0 , totalwaittime = 0 , Queue [ ]

Recurse to time = 3

Same for Minute 3 :-

VIP yes but (3 − 3 > 2) false

wait time = 3 − 3 = 0

totalwait = 0;

Queue [ ]

Recurse to time = 4

⇒ 0 + 0 + 0 + 0 = 0 Ans