Question 1

(a) Silver Meal Heuristic

To begin, we plot a table listing down the demand for each month. Since current inventory is 4 units and ending inventory must be 8, we adjust the demand to account for these requirements. Given a setup cost

Month	1	2	3	4	5	6	7	8	9	10	11	12
Demand	6	12	4	8	15	25	20	5	10	20	5	12
Adjusted Demand	2	12	4	8	15	25	20	5	10	20	5	20

of $K = \$40$ and a holding cost of $h = \$1/\text{unit}\cdot\text{period}$, we can begin our silver meal calculation starting from month 1.

$$\begin{aligned}
C_1(1) &= K && = 40\\
C_1(2) &= (K + hd_2)/2 = (40 + 12)/2 && = 26\\
C_1(3) &= (K + hd_2 + 2hd_3)/3 = (40 + 12 + 2 \cdot 4)/3 && = 20\\
C_1(4) &= (K + hd_2 + 2hd_3 + 3hd_4)/4 = (40 + 12 + 2 \cdot 4 + 3 \cdot 8)/4 && = 21
\end{aligned}$$

Here notice that $C_1(4) > C_1(3)$. Thus, we have reason to stop the current order and start a new order at month 4. Thus, with month 4 as our new starting period, we perform the iteration again.

$$\begin{aligned}
C_2(1) &= K && = 40\\
C_2(2) &= (K + hd_2)/2 = (40 + 15)/2 && = 27.5\\
C_2(3) &= (K + hd_2 + 2hd_3)/3 = (40 + 15 + 2 \cdot 25)/3 && = 35
\end{aligned}$$

Here, $C_2(3) > C_2(2)$ and so we stop the current order. With month 6 as our new starting period, we perform the iteration again.

$$\begin{aligned}
C_3(1) &= K && = 40\\
C_3(2) &= (K + hd_2)/2 = (40 + 20)/2 && = 30\\
C_3(3) &= (K + hd_2 + 2hd_3)/3 = (40 + 20 + 2 \cdot 5)/3 && = 23.\bar{3}\\
C_3(4) &= (K + hd_2 + 2hd_3 + 3hd_4)/4 = (40 + 20 + 2 \cdot 5 + 3 \cdot 10)/4 && = 25
\end{aligned}$$

Here, $C_3(4) > C_3(3)$ and so we stop the current order. With month 9 as our new starting period, we perform the iteration again.

$$\begin{aligned}
C_4(1) &= K && = 40\\
C_4(2) &= (K + hd_2)/2 = (40 + 20)/2 && = 30\\
C_4(3) &= (K + hd_2 + 2hd_3)/3 = (40 + 20 + 2 \cdot 5)/3 && = 23.\bar{3}\\
C_4(4) &= (K + hd_2 + 2hd_3 + 3hd_4)/4 = (40 + 20 + 2 \cdot 5 + 3 \cdot 20)/4 && = 32.5
\end{aligned}$$

Here $C_4(4) > C_4(3)$ and so we stop the current order. With month 12 as our new starting period, we perform the iteration again.

$$C_5(1) = K = 40$$

Month	1	2	3	4	5	6	7	8	9	10	11	12
Demand	6	12	4	8	15	25	20	5	10	20	5	12
Adjusted Demand	2	12	4	8	15	25	20	5	10	20	5	20
Production	18	0	0	23	0	50	0	0	35	0	0	20

Thus, in summary, we have a production schedule which incurs a total cost of:

$$C = \sum_{i=1}^{5} C_i(d_i) \cdot d_i$$

where $C_i(d_i)$ refers to the cost per period of the i-th order for a planning horizon of d_i months.

$$\begin{aligned} C &= C_1(3) \cdot 3 + C_2(2) \cdot 2 + C_3(3) \cdot 3 + C_4(3) \cdot 3 + C_5(1) \cdot 1 \\ &= 20 \cdot 3 + 27.5 \cdot 2 + 23.\bar{3} \cdot 3 + 23.\bar{3} \cdot 3 + 40 \cdot 1 \\ &\boxed{C = 295} \end{aligned}$$

(b) Least Unit Cost Heuristic

Now we perform a very similar method to Silver-Meal. We begin at month 1.

$$\begin{aligned} C_1(1) &= \frac{K}{d_1} &&= \frac{40}{2} &&= 20 \\ C_1(2) &= \frac{K + hd_2}{d_1 + d_2} &&= \frac{40 + 12}{2 + 12} &&= 3.714285714 \\ C_1(3) &= \frac{K + hd_2 + 2hd_3}{d_1 + d_2 + d_3} &&= \frac{40 + 12 + 2 \cdot 4}{2 + 12 + 4} &&= 3.\bar{3} \\ C_1(4) &= \frac{K + hd_2 + 2hd_3 + 3hd_4}{d_1 + d_2 + d_3 + d_4} &&= \frac{40 + 12 + 2 \cdot 4 + 3 \cdot 8}{2 + 12 + 4 + 8} &&= 3.230769231 \\ C_1(5) &= \frac{K + hd_2 + 2hd_3 + 3hd_4 + 4hd_5}{d_1 + d_2 + d_3 + d_4 + d_5} &&= \frac{40 + 12 + 2 \cdot 4 + 3 \cdot 8 + 4 \cdot 15}{2 + 12 + 4 + 8 + 15} &&= 3.512195122 \end{aligned}$$

Since $C_1(5) > C_1(4)$, we stop the current order and start a new order at month 5.

$$\begin{aligned} C_2(1) &= \frac{K}{d_1} &&= \frac{40}{15} &&= 2.\bar{6} \\ C_2(2) &= \frac{K + hd_2}{d_1 + d_2} &&= \frac{40 + 25}{15 + 25} &&= 1.625 \\ C_2(3) &= \frac{K + hd_2 + 2hd_3}{d_1 + d_2 + d_3} &&= \frac{40 + 25 + 3 \cdot 20}{15 + 25 + 20} &&= 1.75 \end{aligned}$$

Since $C_2(3) > C_2(2)$, we stop the current order and start a new order at month 7.

$$\begin{aligned} C_3(1) &= \frac{K}{d_1} &&= \frac{40}{20} &&= 2 \\ C_3(2) &= \frac{K + hd_2}{d_1 + d_2} &&= \frac{40 + 5}{20 + 5} &&= 1.8 \\ C_3(3) &= \frac{K + hd_2 + 2hd_3}{d_1 + d_2 + d_3} &&= \frac{40 + 5 + 2 \cdot 10}{20 + 5 + 10} &&= 1.857142857 \end{aligned}$$

Since $C_3(3) > C_3(2)$, we stop the current order and start a new order at month 9.

$$\begin{aligned}
C_4(1) &= \frac{K}{d_1} &&= \frac{40}{10} &&= 4\\
C_4(2) &= \frac{K + hd_2}{d_1 + d_2} &&= \frac{40+20}{10+20} &&= 2\\
C_4(3) &= \frac{K + hd_2 + 2hd_3}{d_1 + d_2 + d_3} &&= \frac{40+20+2\cdot 5}{10+20+5} &&= 2\\
C_4(4) &= \frac{K + hd_2 + 2hd_3 + 3hd_4}{d_1 + d_2 + d_3 + d_4} &&= \frac{40+20+2\cdot 5+3\cdot 20}{10+20+5+20} &&= 2.363636364
\end{aligned}$$

Since $C_4(3) > C_4(2)$, we stop the current order and start a new order at month 12.

$$C_5(1) = \frac{K}{d_1} = \frac{40}{20} = 2$$

Month	1	2	3	4	5	6	7	8	9	10	11	12
Demand	6	12	4	8	15	25	20	5	10	20	5	12
Adjusted Demand	2	12	4	8	15	25	20	5	10	20	5	20
Production	26	0	0	0	40	0	25	0	35	0	0	20

Thus, we have a production schedule as shown above, incurring a total cost of:

$$C = \sum_{i=1}^{5} C_i(d_i) \cdot u_i$$

where $C_i(d_i)$ refers to the cost per unit of the i-th order for a planning horizon of d_i months; and u_i refers to the number of units made in the i-th order.

$$\begin{aligned}
C &= C_1(4)\cdot 26 + C_2(2)\cdot 40 + C_3(2)\cdot 25 + C_4(3)\cdot 35 + C_5(1)\cdot 20\\
&= 3.230769231\cdot 26 + 1.625\cdot 40 + 1.8\cdot 25 + 2\cdot 35 + 2\cdot 20\\
&\boxed{C = 304}
\end{aligned}$$

(c) Part Period Balancing Heuristic

Here, we implement the part period balancing heuristic. We begin with month 1 as our starting period.

Month	1	2	3	4
Order Horizon	1	2	3	4
Total Holding Cost	0	12	20	44

Since $44 > 40$ and 40 is closer to 44 than 20, we end our current order and start a new order in month 5.

Month	5	6	7
Order Horizon	1	2	3
Total Holding Cost	0	25	65

Since $65 > 40$ and 40 is closer to 25 than 65, we end our current order and start a new order in month 7.

Month	7	8	9	10
Order Horizon	1	2	3	4
Total Holding Cost	0	5	25	85

Since $85 > 40$ and 40 is closer to 25 than 85, we end our current order and start a new order in month 10.

Month	10	11	12
Order Horizon	0	1	2
Total Holding Cost	0	5	45

Here, our planning horizon ends and thus our production schedule is complete. Now, we compute the total

Month	1	2	3	4	5	6	7	8	9	10	11	12
Demand	6	12	4	8	15	25	20	5	10	20	5	12
Adjusted Demand	2	12	4	8	15	25	20	5	10	20	5	20
Production	26	0	0	0	40	0	35	0	0	45	0	0

cost associated with our production schedule by the following equation:

$$C = \sum_{i=1}^{4} \Big(H_i + 40\Big)$$

where H_i refers to the total holding cost the i-th order and 40 arises from the setup cost in each order.

$$C = 44 + 25 + 25 + 45 + 40 \cdot 4$$
$$\boxed{C = 299}$$

(d)

Unsurprisingly, the Silver Meal Heuristic yielded the best results with a total cost of \$295.

Question 2

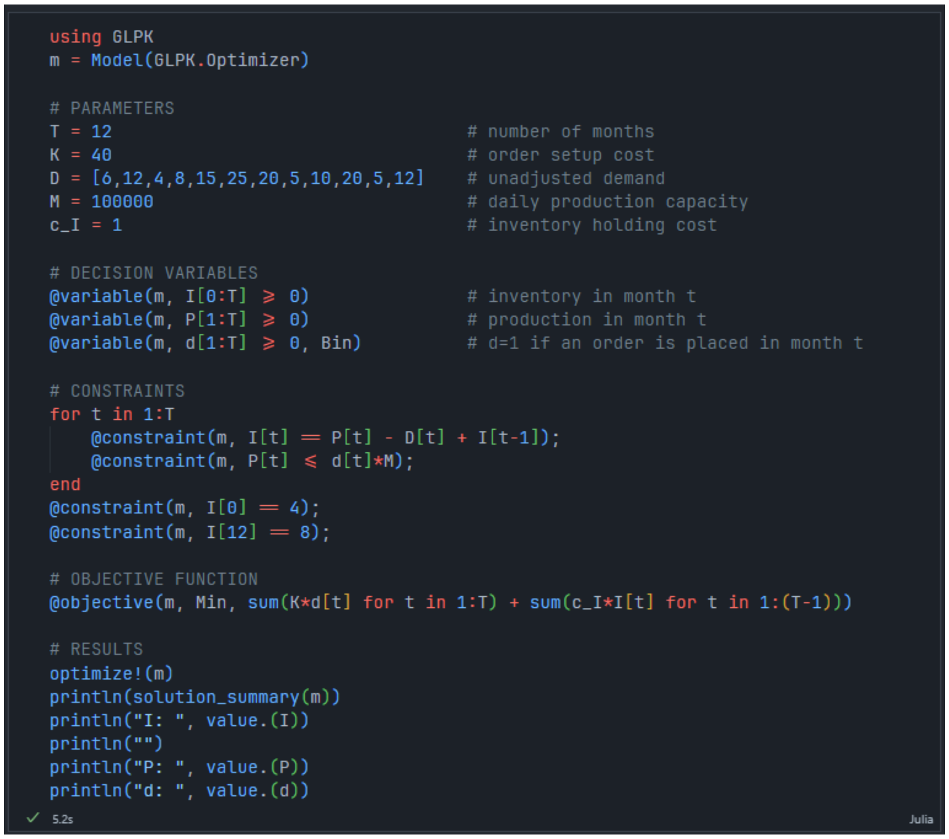
MILP Formulation

(a)

Considering the rather low dimensions of the problem, we can solve this problem through a Mixed-Integer Linear Program.

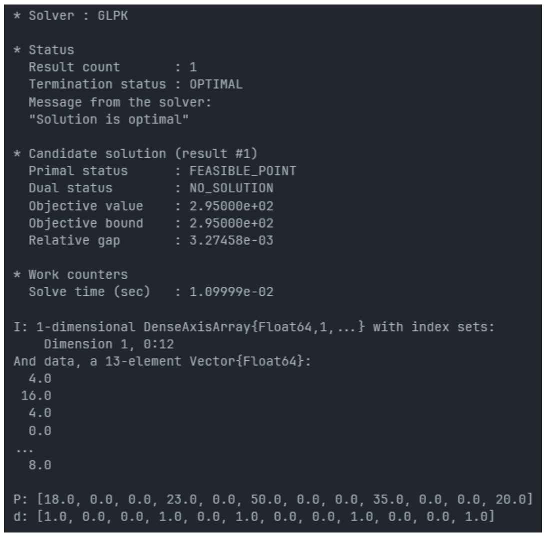
$$\begin{aligned}
\min \quad & \sum_{t=1}^{11} c_I \cdot I_t + \sum_{t=1}^{12} K \cdot \delta_t \\
\text{s.t.} \quad & I_t = P_t - D_t + I_{t-1} && \forall i = 1, \dots, 12 \\
& P \leq \delta_t \cdot M && \forall i = 1, \dots, 12 \\
& I_0 = 4 \\
& I_{12} = 8 \\
& I_t, P_t \geq 0 && \forall i = 1, \dots, 12 \\
& \delta_t \in \{0, 1\} && \forall i = 1, \dots, 12
\end{aligned}$$

where I_t refers to inventory levels in month t; P_t refers to units produced in month t; D_t is the original demand (unadjusted), δ_t is a binary variable equal to 1 if a production order is made in month t, and 0 otherwise; and M is the company's monthly production capacity (in this case, since there the company's production capacity is unmentioned, M is assumed to be a very large number). Solving this problem using Julia, we get:



```julia
using GLPK
m = Model(GLPK.Optimizer)

# PARAMETERS
T = 12                                      # number of months
K = 40                                      # order setup cost
D = [6,12,4,8,15,25,20,5,10,20,5,12]        # unadjusted demand
M = 100000                                  # daily production capacity
c_I = 1                                     # inventory holding cost

# DECISION VARIABLES
@variable(m, I[0:T] ≥ 0)                    # inventory in month t
@variable(m, P[1:T] ≥ 0)                    # production in month t
@variable(m, d[1:T] ≥ 0, Bin)               # d=1 if an order is placed in month t

# CONSTRAINTS
for t in 1:T
    @constraint(m, I[t] == P[t] - D[t] + I[t-1]);
    @constraint(m, P[t] ≤ d[t]*M);
end
@constraint(m, I[0] == 4);
@constraint(m, I[12] == 8);

# OBJECTIVE FUNCTION
@objective(m, Min, sum(K*d[t] for t in 1:T) + sum(c_I*I[t] for t in 1:(T-1)))

# RESULTS
optimize!(m)
println(solution_summary(m))
println("I: ", value.(I))
println("")
println("P: ", value.(P))
println("d: ", value.(d))
```

Figure 1: Code in Julia



```
* Solver : GLPK

* Status
  Result count       : 1
  Termination status : OPTIMAL
  Message from the solver:
  "Solution is optimal"

* Candidate solution (result #1)
  Primal status      : FEASIBLE_POINT
  Dual status        : NO_SOLUTION
  Objective value    : 2.95000e+02
  Objective bound    : 2.95000e+02
  Relative gap       : 3.27458e-03

* Work counters
  Solve time (sec)   : 1.09999e-02

I: 1-dimensional DenseAxisArray{Float64,1,...} with index sets:
    Dimension 1, 0:12
And data, a 13-element Vector{Float64}:
  4.0
 16.0
  4.0
  0.0
...
  8.0

P: [18.0, 0.0, 0.0, 23.0, 0.0, 50.0, 0.0, 0.0, 35.0, 0.0, 0.0, 20.0]
d: [1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0]
```

Figure 2: Results

With this, we get our optimal policy:

$$\boxed{\mathbf{y}^* = (18, 0, 0, 23, 0, 50, 0, 0, 35, 0, 0, 20)}$$

The optimal policy incurs a total cost of:

$$\boxed{C = \$295}$$

which confirms that the silver meal heuristic (which produced an optimal value of \$295) reached an optimal solution.

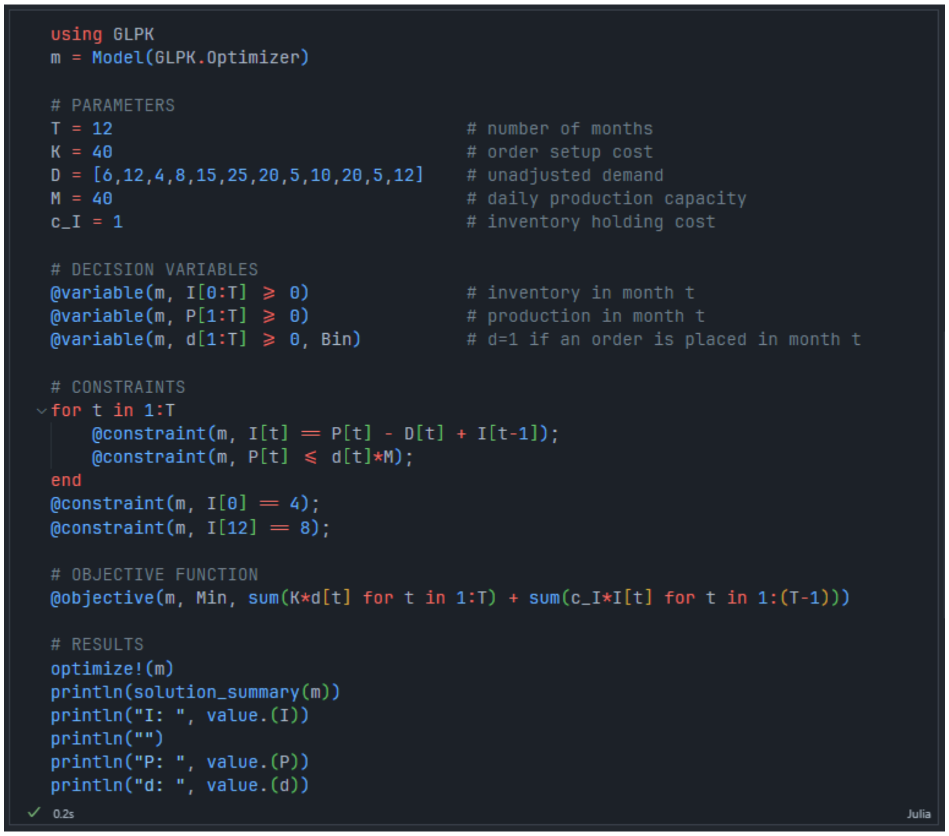
(b)

The MILP formulation presents an elegant solution to the new requirement. We can simply set $M = 40$, effectively limiting the company's ability to send a maximum of 40 TMEs in each month. With this, we simply run our code again and solve the problem.
With this, we get our optimal policy:

$$\boxed{\mathbf{y}^* = (26, 0, 0, 0, 40, 0, 35, 0, 0, 25, 0, 20)}$$

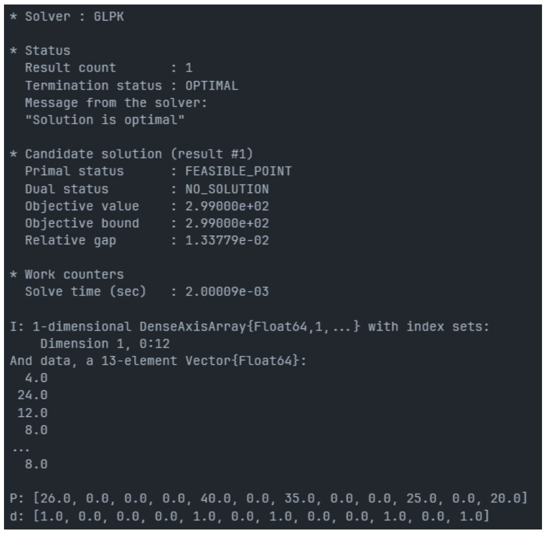
The optimal policy incurs a total cost of:

$$\boxed{C = \$299}$$



```julia
using GLPK
m = Model(GLPK.Optimizer)

# PARAMETERS
T = 12                                  # number of months
K = 40                                  # order setup cost
D = [6,12,4,8,15,25,20,5,10,20,5,12]    # unadjusted demand
M = 40                                  # daily production capacity
c_I = 1                                 # inventory holding cost

# DECISION VARIABLES
@variable(m, I[0:T] ≥ 0)                # inventory in month t
@variable(m, P[1:T] ≥ 0)                # production in month t
@variable(m, d[1:T] ≥ 0, Bin)           # d=1 if an order is placed in month t

# CONSTRAINTS
for t in 1:T
    @constraint(m, I[t] == P[t] - D[t] + I[t-1]);
    @constraint(m, P[t] ≤ d[t]*M);
end
@constraint(m, I[0] == 4);
@constraint(m, I[12] == 8);

# OBJECTIVE FUNCTION
@objective(m, Min, sum(K*d[t] for t in 1:T) + sum(c_I*I[t] for t in 1:(T-1)))

# RESULTS
optimize!(m)
println(solution_summary(m))
println("I: ", value.(I))
println("")
println("P: ", value.(P))
println("d: ", value.(d))
```
✓ 0.2s Julia

Figure 3: Code in Julia



```
* Solver : GLPK

* Status
  Result count       : 1
  Termination status : OPTIMAL
  Message from the solver:
  "Solution is optimal"

* Candidate solution (result #1)
  Primal status      : FEASIBLE_POINT
  Dual status        : NO_SOLUTION
  Objective value    : 2.99000e+02
  Objective bound    : 2.99000e+02
  Relative gap       : 1.33779e-02

* Work counters
  Solve time (sec)   : 2.00009e-03

I: 1-dimensional DenseAxisArray{Float64,1,...} with index sets:
    Dimension 1, 0:12
And data, a 13-element Vector{Float64}:
  4.0
 24.0
 12.0
  8.0
 ...
  8.0

P: [26.0, 0.0, 0.0, 0.0, 40.0, 0.0, 35.0, 0.0, 0.0, 25.0, 0.0, 20.0]
d: [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0]
```

Figure 4: Results

Backwards DP Formulation

(a)

For problems with larger dimensions, a backwards Dynamic Programming (DP) approach may be more suitable. We set up a graph with $n+1$ nodes ($n = 12$ as there are 12 months). The dynamic programming can be implemented by the following recursive structure:

$$f_k = \min_{j>k}(c_{k,j} + f_j) \quad \text{for } k = 1, \dots, n$$

where f_k is the minimum cost starting at node k, assuming that an order is placed in period k; and $c_{k,j}$ is the setup and holding cost of ordering in period k to meet requirements through period $j-1$. Note that the base case for this formulation is $f_{n+1} = f_{13} = 0$.

$$c_{k,j} = K + \sum_{i=1}^{j-1} (i-1) \cdot h \cdot d_i$$

To demonstrate how the backward DP formulation works, consider the following demonstration. We firstly draw the complete network.

$$\begin{tikzcd}
1 \arrow[r] & 2 \arrow[r] & \cdots \arrow[r] & \cdots \arrow[r] & \cdots \arrow[r] & 12 \arrow[r] & 13
\end{tikzcd}$$

Now, we consider the last arc (or the last two nodes) of the network.

$$\begin{tikzcd}
\cdots \arrow[r] & 12 \arrow[r, "c_{12,13}"] \arrow[r, bend left, "f_{12}"] & 13
\end{tikzcd}$$

To travel through the subpath 12 to 13, we incur a cost of:

$$\begin{aligned}
f_{12} &= \min(c_{12,13} + f_{13}) \\
&= K + 0 \\
\boxed{f_{12} = 40} &\text{ at } j = 13
\end{aligned}$$

We now move backwards through the network (hence the name backward DP), considering the nodes 11, 12, and 13.

$$\begin{tikzcd}
\cdots \arrow[r] & 11 \arrow[r, "c_{11,12}"] \arrow[rr, bend right, "c_{11,13}"'] & 12 \arrow[r, "f_{12}"] & 13
\end{tikzcd}$$

$$f_{11} = \min \left\{ \begin{array}{c} c_{11,12} + f_{12} \\ c_{11,13} + f_{13} \end{array} \right\} = \min \left\{ \begin{array}{c} (K+0) + 40 \\ (K+20) + 0 \end{array} \right\} = \min \left\{ \begin{array}{c} 80 \\ 60 \end{array} \right\}$$

$$\boxed{f_{11} = 60} \text{ at } j = 13$$

Moving backwards once again, we get:

$$\begin{tikzcd}
\cdots \arrow[r] & 10 \arrow[r, "c_{10,11}"] \arrow[rr, bend left, "c_{10,12}"] \arrow[rrr, bend right, "c_{10,13}"'] & 11 \arrow[rr, bend left, "f_{11}"] & 12 \arrow[r, "f_{12}"] & 13
\end{tikzcd}$$

$$f_{10} = \min \left\{ \begin{array}{l} c_{10,11} + f_{11} \\ c_{10,12} + f_{12} \\ c_{10,13} + f_{13} \end{array} \right\} = \min \left\{ \begin{array}{l} (K+0)+60 \\ (K+5)+40 \\ (K+5+2(20))+0 \end{array} \right\} = \min \left\{ \begin{array}{l} 100 \\ 85 \\ 85 \end{array} \right\}$$

$\boxed{f_{10} = 85}$ at $j = 13$

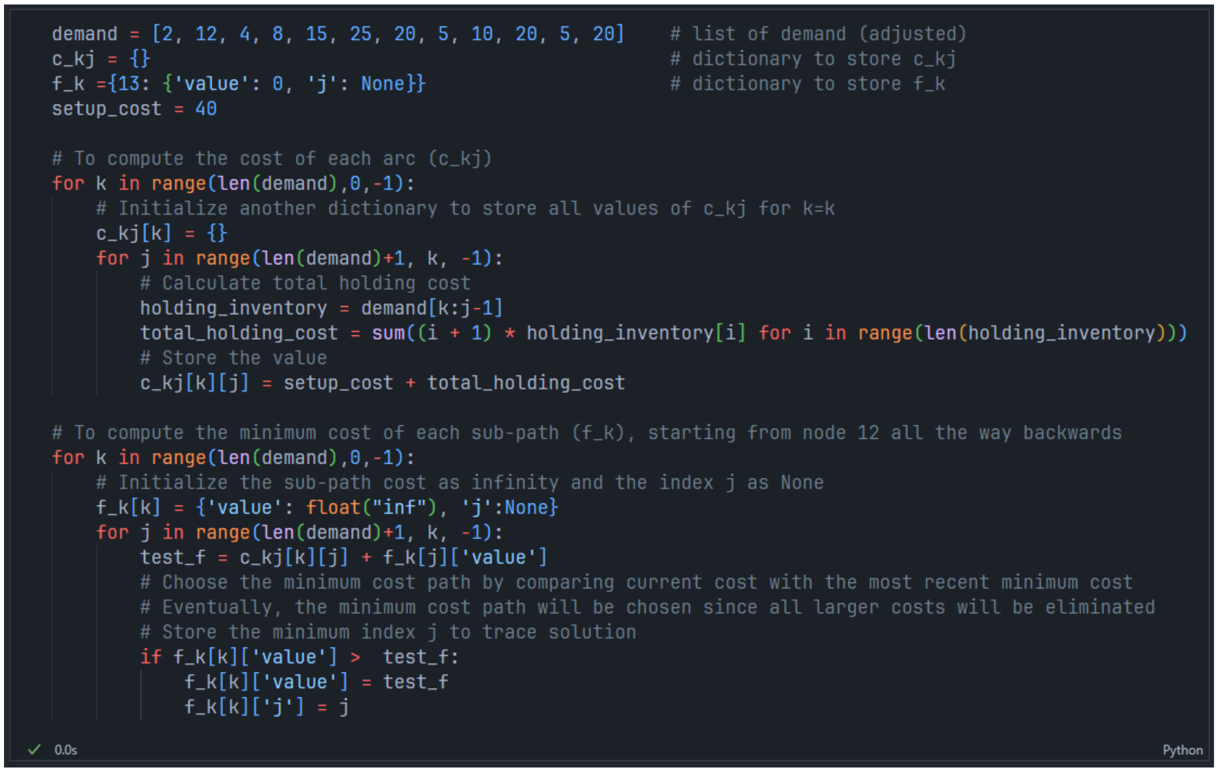
Moving backwards another time, we get:

$$\begin{tikzcd}
\cdots \arrow[r] & 9 \arrow[r, "c_{9,10}"] \arrow[rr, bend left, "c_{9,11}"] \arrow[rrr, bend right, "c_{9,12}"'] \arrow[rrrr, bend right=50, "c_{9,13}"'] & 10 \arrow[rrr, bend right, "f_{10}"'] & 11 \arrow[rr, bend left, "f_{11}"] & 12 \arrow[r, "f_{12}"] & 13
\end{tikzcd}$$

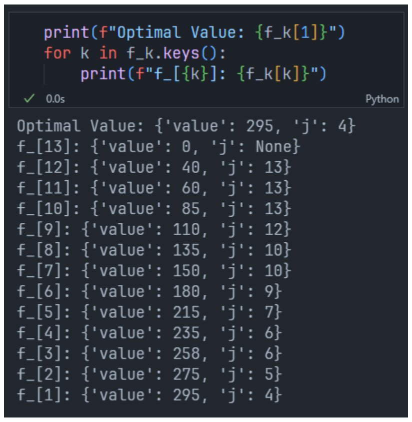
$$f_9 = \min \left\{ \begin{array}{l} c_{9,10} + f_{10} \\ c_{9,11} + f_{11} \\ c_{9,12} + f_{12} \\ c_{9,13} + f_{13} \end{array} \right\} = \min \left\{ \begin{array}{l} (K+0)+85 \\ (K+20)+60 \\ (K+20+2(5))+40 \\ (K+20+2(5)+3(20))+0 \end{array} \right\} = \min \left\{ \begin{array}{l} 125 \\ 120 \\ 110 \\ 130 \end{array} \right\}$$

$\boxed{f_9 = 110}$ at $j = 12$

We can keep going but we can also let Python do the work for us. By using the following code, we complete the DP.



```python
demand = [2, 12, 4, 8, 15, 25, 20, 5, 10, 20, 5, 20]    # list of demand (adjusted)
c_kj = {}                                               # dictionary to store c_kj
f_k ={13: {'value': 0, 'j': None}}                      # dictionary to store f_k
setup_cost = 40

# To compute the cost of each arc (c_kj)
for k in range(len(demand),0,-1):
    # Initialize another dictionary to store all values of c_kj for k=k
    c_kj[k] = {}
    for j in range(len(demand)+1, k, -1):
        # Calculate total holding cost
        holding_inventory = demand[k:j-1]
        total_holding_cost = sum((i + 1) * holding_inventory[i] for i in range(len(holding_inventory)))
        # Store the value
        c_kj[k][j] = setup_cost + total_holding_cost

# To compute the minimum cost of each sub-path (f_k), starting from node 12 all the way backwards
for k in range(len(demand),0,-1):
    # Initialize the sub-path cost as infinity and the index j as None
    f_k[k] = {'value': float("inf"), 'j':None}
    for j in range(len(demand)+1, k, -1):
        test_f = c_kj[k][j] + f_k[j]['value']
        # Choose the minimum cost path by comparing current cost with the most recent minimum cost
        # Eventually, the minimum cost path will be chosen since all larger costs will be eliminated
        # Store the minimum index j to trace solution
        if f_k[k]['value'] >  test_f:
            f_k[k]['value'] = test_f
            f_k[k]['j'] = j
```



```
print(f"Optimal Value: {f_k[1]}")
for k in f_k.keys():
    print(f"f_[{k}]: {f_k[k]}")
```

```
Optimal Value: {'value': 295, 'j': 4}
f_[13]: {'value': 0, 'j': None}
f_[12]: {'value': 40, 'j': 13}
f_[11]: {'value': 60, 'j': 13}
f_[10]: {'value': 85, 'j': 13}
f_[9]: {'value': 110, 'j': 12}
f_[8]: {'value': 135, 'j': 10}
f_[7]: {'value': 150, 'j': 10}
f_[6]: {'value': 180, 'j': 9}
f_[5]: {'value': 215, 'j': 7}
f_[4]: {'value': 235, 'j': 6}
f_[3]: {'value': 258, 'j': 6}
f_[2]: {'value': 275, 'j': 5}
f_[1]: {'value': 295, 'j': 4}
```

To analyze the solutions, we use the j indices saved throughout the iterations. Starting from $k = 1$, we utilize the Theorem of Optimality to find y_k, which is the production units in period k.

$$\because (k, j) = (1, 4)$$
$$y_1 = d_1 + d_2 + d_3 = 2 + 12 + 4$$
$$\boxed{y_1 = 18, y_2 = y_3 = 0}$$

We repeat this process for $k = 4$, $k = 6$, $k = 9$, and $k = 12$.

$$\because (k, j) = (4, 6)$$
$$y_4 = d_4 + d_5 = 8 + 15$$
$$\boxed{y_4 = 23, y_5 = 0}$$

$$\because (k, j) = (9, 12)$$
$$y_9 = d_9 + d_{10} + d_{11} = 10 + 20 + 5$$
$$\boxed{y_9 = 35, y_{10} = y_{11} = 0}$$

$$\because (k, j) = (6, 9)$$
$$y_6 = d_6 + d_7 + d_8 = 25 + 20 + 5$$
$$\boxed{y_6 = 50, y_7 = y_8 = 0}$$

$$\because (k, j) = (12, 13)$$
$$y_{12} = d_{12} = 20$$
$$\boxed{y_{12} = 20}$$

With this, we have our optimal policy:

$$\boxed{\mathbf{y}^* = (18, 0, 0, 23, 0, 50, 0, 0, 35, 0, 0, 20)}$$

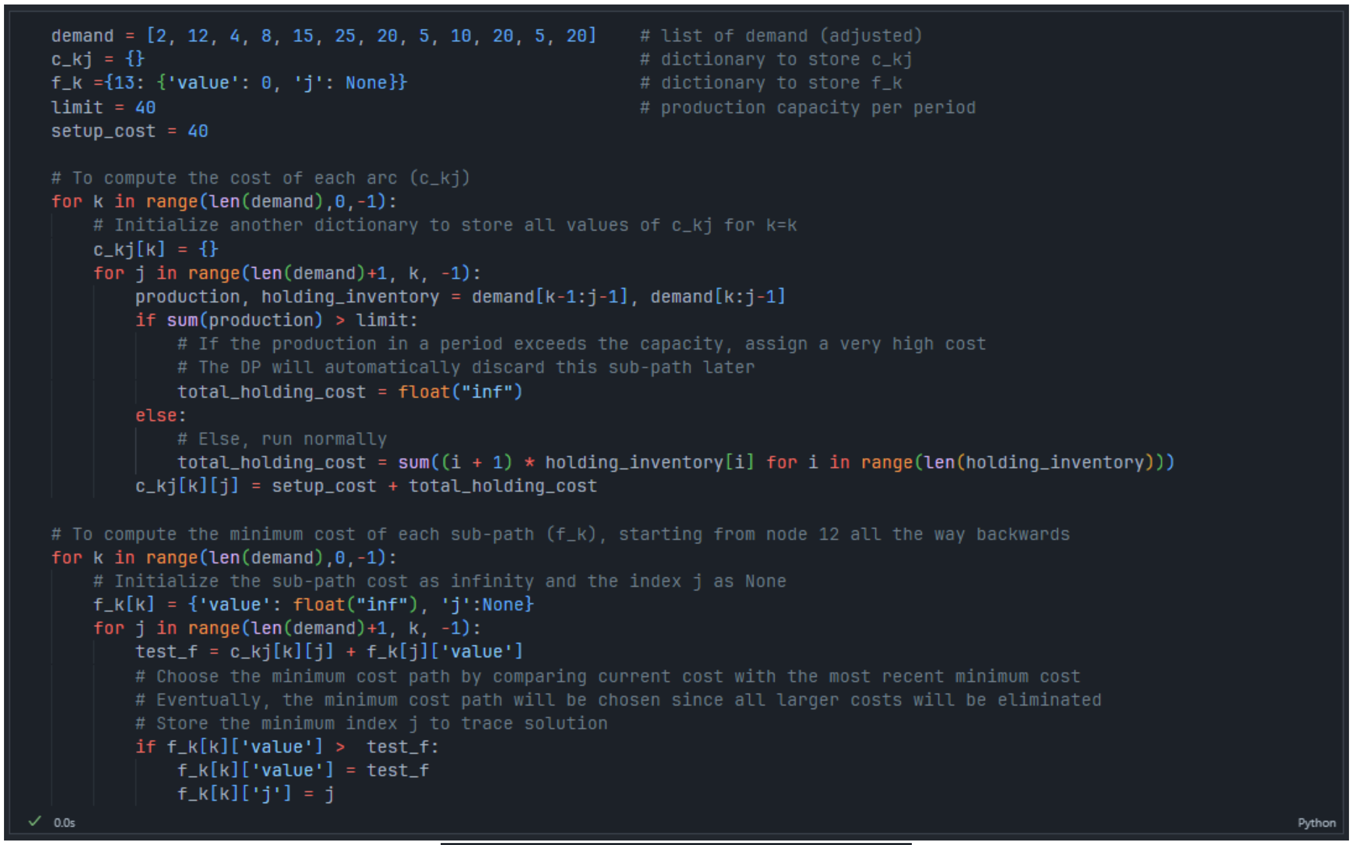
This policy incurs a total cost of:

$$\boxed{C = f_1 = \$295}$$

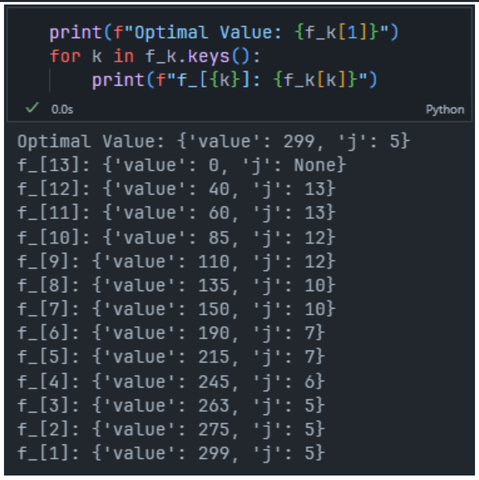
This conclusion essentially confirms that the silver meal heuristic (which produced an optimal value of \$295) reached an optimal solution.

(b)

Now, to implement the extra requirement, we add another procedure to the algorithm. As shown in the code below, we add an extra step which computes the production value required at the start of a sub-path. If this production value is higher than the given production capacity, then we assign a very high cost to the sub-path. Doing so will prompt the DP algorithm to automatically eliminate any sub-paths which involve a production schedule that exceeds production capacity at any point in time. The DP algorithm will take any other feasible alternatives with a smaller cost, ensuring that the extra requirement is implemented.



```python
demand = [2, 12, 4, 8, 15, 25, 20, 5, 10, 20, 5, 20]    # list of demand (adjusted)
c_kj = {}                                               # dictionary to store c_kj
f_k ={13: {'value': 0, 'j': None}}                      # dictionary to store f_k
limit = 40                                              # production capacity per period
setup_cost = 40

# To compute the cost of each arc (c_kj)
for k in range(len(demand),0,-1):
    # Initialize another dictionary to store all values of c_kj for k=k
    c_kj[k] = {}
    for j in range(len(demand)+1, k, -1):
        production, holding_inventory = demand[k-1:j-1], demand[k:j-1]
        if sum(production) > limit:
            # If the production in a period exceeds the capacity, assign a very high cost
            # The DP will automatically discard this sub-path later
            total_holding_cost = float("inf")
        else:
            # Else, run normally
            total_holding_cost = sum((i + 1) * holding_inventory[i] for i in range(len(holding_inventory)))
        c_kj[k][j] = setup_cost + total_holding_cost

# To compute the minimum cost of each sub-path (f_k), starting from node 12 all the way backwards
for k in range(len(demand),0,-1):
    # Initialize the sub-path cost as infinity and the index j as None
    f_k[k] = {'value': float("inf"), 'j':None}
    for j in range(len(demand)+1, k, -1):
        test_f = c_kj[k][j] + f_k[j]['value']
        # Choose the minimum cost path by comparing current cost with the most recent minimum cost
        # Eventually, the minimum cost path will be chosen since all larger costs will be eliminated
        # Store the minimum index j to trace solution
        if f_k[k]['value'] >  test_f:
            f_k[k]['value'] = test_f
            f_k[k]['j'] = j
```



```python
print(f"Optimal Value: {f_k[1]}")
for k in f_k.keys():
    print(f"f_[{k}]: {f_k[k]}")
```

```
Optimal Value: {'value': 299, 'j': 5}
f_[13]: {'value': 0, 'j': None}
f_[12]: {'value': 40, 'j': 13}
f_[11]: {'value': 60, 'j': 13}
f_[10]: {'value': 85, 'j': 12}
f_[9]: {'value': 110, 'j': 12}
f_[8]: {'value': 135, 'j': 10}
f_[7]: {'value': 150, 'j': 10}
f_[6]: {'value': 190, 'j': 7}
f_[5]: {'value': 215, 'j': 7}
f_[4]: {'value': 245, 'j': 6}
f_[3]: {'value': 263, 'j': 5}
f_[2]: {'value': 275, 'j': 5}
f_[1]: {'value': 299, 'j': 5}
```

By undertaking steps similar to part (a), we retrieve the optimal solution to the problem. Note that the results match with the results from the MILP formulation, verifying our approach. Note too, that in this particular policy, there are no production values which exceed the given production capacity of 40 units.

$$\boxed{\mathbf{y}^* = (26, 0, 0, 0, 40, 0, 35, 0, 0, 25, 0, 20)}$$

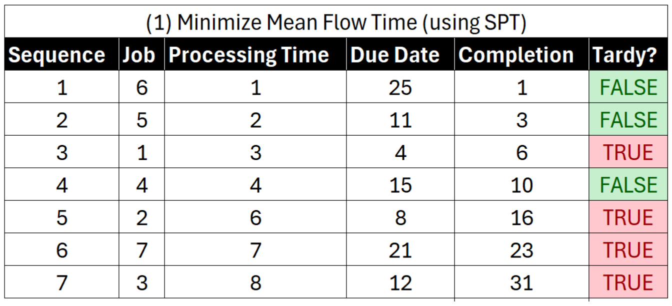
In addition, this policy incurs a total cost of:

$$\boxed{C = f_1 = \$299}$$

Question 3

Minimizing Mean Flow Time

To minimize mean flow time, we implement the Shortest-Processing-Time rule whereby every job with the shortest processing time gets scheduled ahead. By creating the following table in Excel and sorting by the column 'Processing Time' we get a sequence with minimum mean flow time.



(1) Minimize Mean Flow Time (using SPT)

Sequence	Job	Processing Time	Due Date	Completion	Tardy?
1	6	1	25	1	FALSE
2	5	2	11	3	FALSE
3	1	3	4	6	TRUE
4	4	4	15	10	FALSE
5	2	6	8	16	TRUE
6	7	7	21	23	TRUE
7	3	8	12	31	TRUE

Here and in the following parts, we assume that there are no delays involved between different jobs (different sequencing results do not affect inter-job processing time) and there is only a single machine running which is capable of processing all the jobs but only one at a time. The sequence we obtain has the following mean flow time:

$$\begin{aligned} F' &= \frac{1}{n}\sum_{k=1}^{n} F_{[k]} \\ &= (1+3+6+10+16+23+31)/7 = 12.8571428571 \\ \boxed{F' = 12.857} \end{aligned}$$

In the above, $F_{[k]}$ is calculated by computing the completion time of job k, which is the sum of the processing times of job 1 to job k.

Minimizing Number of Tardy Jobs

To minimize the number of tardy jobs, we implement Moore's Algorithm. Firstly, we sequence the jobs based on the earliest due date (EDD).

(2) Minimize Number of Tardy Jobs (use EDD first)

Sequence	Job	Processing Time	Due Date	Completion	Tardy?
1	1	3	4	3	FALSE
2	2	6	8	9	TRUE
3	5	2	11	11	FALSE
4	3	8	12	19	TRUE
5	4	4	15	23	TRUE
6	7	7	21	30	TRUE
7	6	1	25	31	TRUE

Now, we scan for the first tardy job, which in this case is job 2. We discard job 2 since it has the largest processing time from jobs 1 to 2 and rearrange the sequence. All other jobs are shifted up the schedule, resulting in the following seqeuence.

Sequence	Job	Processing Time	Due Date	Completion	Tardy?
1	1	3	4	3	FALSE
2	5	2	11	5	FALSE
3	3	8	12	13	TRUE
4	4	4	15	17	TRUE
5	7	7	21	24	TRUE
6	6	1	25	25	FALSE
7					

Again, we scan for the first tardy job and find that it is job 3. Between jobs 1 to 3, job 3 has the highest processing time, and thus we discard this and rearrange the sequence.

Sequence	Job	Processing Time	Due Date	Completion	Tardy?
1	1	3	4	3	FALSE
2	5	2	11	5	FALSE
3	4	4	15	9	FALSE
4	7	7	21	16	FALSE
5	6	1	25	17	FALSE
6					
7					

Since there are no more tardy jobs, we can append the initially rejected jobs in any order to the end of the sequence. With this, we get the optimal sequence to minimize the number of tardy jobs.

Sequence	Job	Processing Time	Due Date	Completion	Tardy?
1	1	3	4	3	FALSE
2	5	2	11	5	FALSE
3	4	4	15	9	FALSE
4	7	7	21	16	FALSE
5	6	1	25	17	FALSE
6	2	6	8	23	TRUE
7	3	8	12	31	TRUE

Minimizing Maximum Lateness

To minimize maximum lateness, we simply use the EDD rule where every job with the earliest deadline gets scheduled first. We simply sort the table based on the column 'Due Date' and obtain the optimal sequence to minimize maximum lateness.

(3) Minimize Maximum Lateness (using EDD)

Sequence	Job	Processing Time	Due Date	Completion	Tardy?
1	1	3	4	3	FALSE
2	2	6	8	9	TRUE
3	5	2	11	11	FALSE
4	3	8	12	19	TRUE
5	4	4	15	23	TRUE
6	7	7	21	30	TRUE
7	6	1	25	31	TRUE

Minimizing Makespan

We have assumed that there is only a single machine processing each job (and one at a time), and that the sequencing of jobs does not affect the time to transition between different jobs. As such, in this particular context, the makespan has been optimized. No matter the arrangement of jobs, and provided that there are no unexpected delays between jobs, then the makespan will constantly be the same as the completion time of the job.

Question 4

First Come First Serve

We only have 1 dock to unload the trucks' contents, and here we proceed with the First Come First Serve (FCFS) approach. The trucks that arrive the earliest are scheduled first, and since A, B, C, D are labels given according to the trucks' order of arrival, we simply sort the sequence in alphabetical order to implement this approach. Hence our FCFS schedule.

(1) First Come First Serve

Sequence	Delivery Vehicle	Unloading Time	Deadline	Completion Time	Flow Time	Tardy?	Tardiness
1	A	20	1:25:00 PM	1:20:00 PM	20	FALSE	0
2	B	14	1:45:00 PM	1:34:00 PM	34	FALSE	0
3	C	35	1:50:00 PM	2:09:00 PM	69	TRUE	19
4	D	10	1:30:00 PM	2:19:00 PM	79	TRUE	49

In this context, the flow time of the k-th truck is the sum of the first to k-th unloading times. With this, and with the help of Excel, we compute the mean flow time, number of tardy jobs, and average tardiness.

$$F' = (20 + 34 + 69 + 79)/4 = 50.5 \text{ minutes}$$
$$\text{Number of Tardy Jobs} = 2$$
$$\text{Average Tardiness} = (0 + 0 + 19 + 49)/4 = 17 \text{ minutes}$$

Shortest Processing Time

In this approach, we sort the trucks by order of shortest processing time. Trucks with smallest unloading times go first, resulting in the following schedule.

(2) Shortest Processing Time

Sequence	Delivery Vehicle	Unloading Time	Deadline	Completion Time	Flow Time	Tardy?	Tardiness
1	D	10	1:30:00 PM	1:10:00 PM	10	FALSE	0
2	B	14	1:45:00 PM	1:24:00 PM	24	FALSE	0
3	A	20	1:25:00 PM	1:44:00 PM	44	TRUE	19
4	C	35	1:50:00 PM	2:19:00 PM	79	TRUE	29

The flow time, number of tardy jobs, and average tardiness is computed using the same method as before.

$$\begin{aligned} F' &= (10 + 24 + 44 + 79)/4 = 39.25 \text{ minutes} \\ \text{Number of Tardy Jobs} &= 2 \\ \text{Average Tardiness} &= (0 + 0 + 19 + 29)/4 = 12 \text{ minutes} \end{aligned}$$

Earliest Due Date

Here, we sort the trucks such that those with the earliest deadline unload first.

(3) Earliest Due Date

Sequence	Delivery Vehicle	Unloading Time	Deadline	Completion Time	Flow Time	Tardy?	Tardiness
1	A	20	1:25:00 PM	1:20:00 PM	20	FALSE	0
2	D	10	1:30:00 PM	1:30:00 PM	30	FALSE	0
3	B	14	1:45:00 PM	1:44:00 PM	44	FALSE	0
4	C	35	1:50:00 PM	2:19:00 PM	79	TRUE	29

The flow time, number of tardy jobs, and average tardiness is computed using the same method as before.

$$\begin{aligned} F' &= (20 + 30 + 44 + 79)/4 = 43.25 \text{ minutes} \\ \text{Number of Tardy Jobs} &= 1 \\ \text{Average Tardiness} &= (0 + 0 + 0 + 29)/4 = 7.25 \text{ minutes} \end{aligned}$$

Critical Ratio

Here, we implement the Critical Ratio approach. The critical ratio is defined as:

$$\text{Critical Ratio} = \frac{\text{Remaining Time}}{\text{Processing Time}}$$

We implement the method by calculating the critical ratio iteratively and prioritizing operations with the lowest critical ratio in each iteration. In our first iteration, truck A has the lowest critical ratio and as such is scheduled first.

(4) Critical Ratio				Current Time=	1:00:00 PM		
Sequence	**Delivery Vehicle**	**Unloading Time**	**Deadline**	**Remaining Time**	**Critical Ratio**	**Completion Time**	**Tardy?**
1	A	20	1:25:00 PM	25	1.250	1:20:00 PM	FALSE
	B	14	1:45:00 PM	45	3.214		
	C	35	1:50:00 PM	50	1.429		
	D	10	1:30:00 PM	30	3.000		

We calculate the critical ratio for another iteration, and this time C has the lowest critical ratio and is scheduled next.

(4) Critical Ratio				Current Time=	1:20:00 PM		
Sequence	**Delivery Vehicle**	**Unloading Time**	**Deadline**	**Remaining Time**	**Critical Ratio**	**Completion Time**	**Tardy?**
1	A	20	1:25:00 PM	-	-	1:20:00 PM	FALSE
2	C	35	1:50:00 PM	30	0.857	1:55:00 PM	TRUE
	D	10	1:30:00 PM	10	1.000		
	B	14	1:45:00 PM	25	1.786		

Another iteration, D has the lowest critical ratio and is scheduled next. B is the lat remaining truck and thus is scheduled last.

(4) Critical Ratio				Current Time=	1:55:00 PM		
Sequence	**Delivery Vehicle**	**Unloading Time**	**Deadline**	**Remaining Time**	**Critical Ratio**	**Completion Time**	**Tardy?**
1	A	20	1:25:00 PM	-	-	1:20:00 PM	FALSE
2	C	35	1:50:00 PM	-	-	1:55:00 PM	TRUE
3	D	10	1:30:00 PM	-25.00	-2.500	2:05:00 PM	TRUE
	B	14	1:45:00 PM	-10.00	-0.714		

We calculate the completion times, flow time, and tardiness below.

(4) Critical Ratio							
Sequence	**Delivery Vehicle**	**Unloading Time**	**Deadline**	**Completion Time**	**Flow Time**	**Tardy?**	**Tardiness**
1	A	20	1:25:00 PM	1:20:00 PM	20	FALSE	0
2	C	35	1:50:00 PM	1:55:00 PM	55	TRUE	5
3	D	10	1:30:00 PM	2:05:00 PM	65	TRUE	35
4	B	14	1:45:00 PM	2:19:00 PM	79	TRUE	34

The flow time, number of tardy jobs, and average tardiness is computed using the same method as before.

$$F' = (20 + 55 + 65 + 79)/4 = 54.75 \text{ minutes}$$
$$\text{Number of Tardy Jobs} = 3$$
$$\text{Average Tardiness} = (0 + 5 + 35 + 34)/4 = 18.5 \text{ minutes}$$

In summary is a comparison of the mean flow times, number of tardy jobs, and average tardiness across the four methods explored.

Method	**Mean Flow Time**	**Tardy Jobs**	**Average Tardiness**
FCFS	50.5	2	17
SPT	39.25	2	12
EDD	43.25	1	7.25
CR	54.75	3	18.5
Best	SPT	EDD	EDD
Worst	CR	CR	CR

The results cannot be generalized (e.g. we cannot conclude that EDD is the best for every scenario just because it performed best in terms of minimizing tardy jobs and average tardiness in this particular scenario). Each method has its strengths and weaknesses and must be implemented according to the needs and situation.

Question 5

To solve this problem, we implement Johnson's algorithm for solving two machine problems. We see Laurel and Hardy as two machines (following Johnson's notation, labelled A and B respectively) and we implement the following rule:

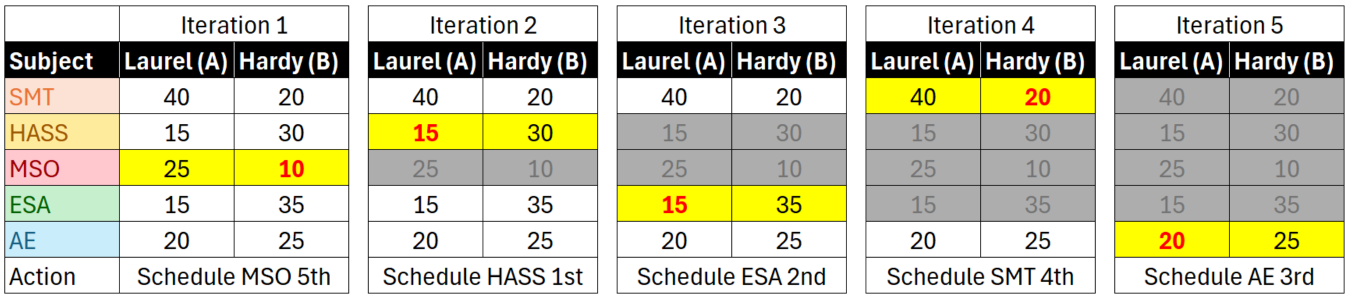
$$\text{Job } i \text{ precedes } j \text{ if } \min(A_i, B_j) < \min(A_j, B_i)$$

To solve this problem, we follow these steps:

- List values of A_i and B_i in 2 columns
- Find smallest remaining value in the 2 columns. If it belongs to Col A, schedule job next. If it belongs to Col B, schedule job last.
- Cross off scheduled job
- Stop when done

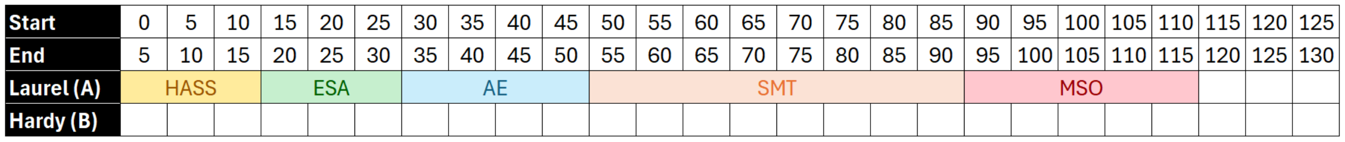
Following these rules (shown in the screenshot below), we arrive at the optimal scheduling sequence:

$$[\text{HASS, ESA, AE, SMT, MSO}]$$



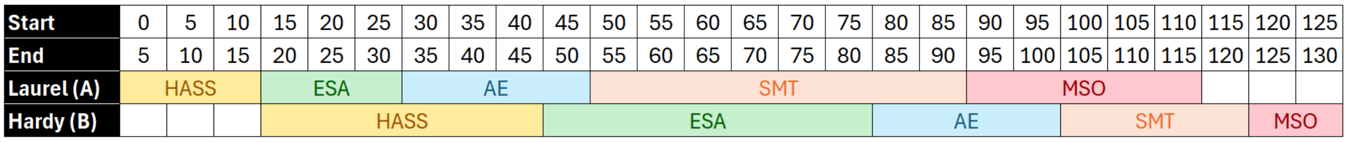
	Iteration 1		Iteration 2		Iteration 3		Iteration 4		Iteration 5	
Subject	**Laurel (A)**	**Hardy (B)**	**Laurel (A)**	**Hardy (B)**	**Laurel (A)**	**Hardy (B)**	**Laurel (A)**	**Hardy (B)**	**Laurel (A)**	**Hardy (B)**
SMT	40	20	40	20	40	20	40	20	40	20
HASS	15	30	15	30	15	30	15	30	15	30
MSO	25	10	25	10	25	10	25	10	25	10
ESA	15	35	15	35	15	35	15	35	15	35
AE	20	25	20	25	20	25	20	25	20	25
Action	Schedule MSO 5th		Schedule HASS 1st		Schedule ESA 2nd		Schedule SMT 4th		Schedule AE 3rd	

Thus, we can now proceed to schedule the jobs, beginning with machine A for Laurel. Evidently, there is no lead time between different jobs. This is the nature of machine A in any optimal situations produced by Johnson's algorithm.



Start	0	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100	105	110	115	120	125
End	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100	105	110	115	120	125	130
Laurel (A)	HASS	HASS	HASS	ESA	ESA	ESA	AE	AE	AE	AE	SMT	SMT	SMT	SMT	SMT	SMT	SMT	SMT	MSO	MSO	MSO	MSO	MSO			
Hardy (B)																										

Next, following the same sequence, we schedule the jobs for machine B, A.K.A. Hardy. We try to fit the sequence as best as we can (by minimizing lead times as much as possible) and arrive at the following schedule.



Start	0	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100	105	110	115	120	125
End	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100	105	110	115	120	125	130
Laurel (A)	HASS	HASS	HASS	ESA	ESA	ESA	AE	AE	AE	AE	SMT	SMT	SMT	SMT	SMT	SMT	SMT	SMT	MSO	MSO	MSO	MSO	MSO			
Hardy (B)				HASS	HASS	HASS	HASS	HASS	HASS	ESA	ESA	ESA	ESA	ESA	ESA	ESA	AE	AE	AE	AE	SMT	SMT	SMT	SMT	MSO	MSO

While Hardy does have to start and end at a later timing than Laurel, the schedule is optimal as there is absolutely no lead time between any of the jobs, for both Laurel and Hardy. Thus, in terms of the 'makespan' (i.e., the total time spent to complete all operations, including any transition times between operations if any), both Laurel and Hardy are following an optimal schedule.