Q1(b)

In [2]:

```
from gurobipy import *

m = Model("dual")

# Creat variables
p1 = m.addVar(name = "p1")
p2 = m.addVar(name = "p2")

# Set objective
m.setObjective(2*p1 + 3*p2, GRB.MAXIMIZE)

# Add constraint:
m.addConstr(p1+2*p2 <= 2)

m.addConstr(6*p1-5*p2 <= 15)

m.addConstr(3*p1+p2 <= 5)

m.addConstr(p1-3*p2 <= 6)

m.addConstr(p1 >= 0)
m.addConstr(p2 >= 0)

# Solving the model
m.optimize()

#  Print optimal solutions and optimal value
for v in m.getVars():
    print(v.VarName,":", v.x)

print('Obj:', m.objVal)
```

```
Optimize a model with 6 rows, 2 columns and 10 nonzeros
Coefficient statistics:
  Matrix range     [1e+00, 6e+00]
  Objective range  [2e+00, 3e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+00, 2e+01]
Presolve removed 4 rows and 0 columns
Presolve time: 0.01s
Presolved: 2 rows, 2 columns, 4 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    4.0000000e+00   5.806667e-01   0.000000e+00      0s
       2    3.8000000e+00   0.000000e+00   0.000000e+00      0s

Solved in 2 iterations and 0.04 seconds
Optimal objective  3.800000000e+00
p1 : 1.6
p2 : 0.19999999999999996
Obj: 3.8
```

In [3]:

```
m = Model("primal")

# Creat variables
x1 = m.addVar(name = "x1")
x2 = m.addVar(name = "x2")
x3 = m.addVar(name = "x3")
x4 = m.addVar(name = "x4")

# Set objective
m.setObjective(2*x1 + 15*x2 + 5*x3 + 6*x4, GRB.MINIMIZE)

# Add constraint:
m.addConstr(x1 + 6*x2 + 3*x3 + x4 >= 2)

m.addConstr(2*x1-5*x2 + x3-3*x4 >= 3)

m.addConstr(x1>=0)
m.addConstr(x2>=0)
m.addConstr(x3>=0)
m.addConstr(x4>=0)

# Solving the model
m.optimize()

#  Print optimal solutions and optimal value
for v in m.getVars():
    print(v.VarName,":", v.x)

print('Obj:', m.objVal)
```

```
Optimize a model with 6 rows, 4 columns and 12 nonzeros
Coefficient statistics:
  Matrix range     [1e+00, 6e+00]
  Objective range  [2e+00, 2e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+00, 3e+00]
Presolve removed 4 rows and 2 columns
Presolve time: 0.03s
Presolved: 2 rows, 2 columns, 4 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   2.000000e+00   0.000000e+00      0s
       2    3.8000000e+00   0.000000e+00   0.000000e+00      0s

Solved in 2 iterations and 0.05 seconds
Optimal objective  3.800000000e+00
x1 : 1.4000000000000001
x2 : 0.0
x3 : 0.19999999999999996
x4 : 0.0
Obj: 3.8
```

Q4

In [5]:

```
m = Model("diet")

# Creat variables
# addVar(lb=0.0, ub=GRB.INFINITY, obj=0.0, vtype=GRB.CONTINUOUS (variable type is CONTINUOUS), name
# lb: lower bound, ub: upper bound
# vtype: continuous, binary or integer
# name: name for the variable
x1 = m.addVar(name = "Apple")
x2 = m.addVar(name = "Banana")
x3 = m.addVar(name = "Blueberries")
x4 = m.addVar(name = "Durian")
x5 = m.addVar(name = "Tangerine")
# Set objective
# setObjective ( expr, sense=None )
# expr: linear or quadratic expression
# sense: GRB.MINIMIZE or GRB.MAXIMIZE
m.setObjective(0.5*x1 + 0.3*x2 + 2.5*x3 + 10*x4 + 0.5*x5, GRB.MINIMIZE)

# Add constraint:
m.addConstr(52*x1 + 89*x2 + 57*x3 + 147*x4 + 53*x5 >= 500)
m.addConstr(52*x1 + 89*x2 + 57*x3 + 147*x4 + 53*x5 <= 3000)

m.addConstr(14*x1 + 23*x2 + 14*x3 + 27*x4 + 13*x5 >= 50)
m.addConstr(14*x1 + 23*x2 + 14*x3 + 27*x4 + 13*x5 <= 400)

m.addConstr(2.4*x1 + 2.6*x2 + 2.4*x3 + 3.8*x4 + 1.8*x5 >= 20)
m.addConstr(2.4*x1 + 2.6*x2 + 2.4*x3 + 3.8*x4 + 1.8*x5 <= 30)

m.addConstr(54*x1 + 64*x2 + 54*x3 + 44*x4 + 681*x5 >= 2000)
m.addConstr(54*x1 + 64*x2 + 54*x3 + 44*x4 + 681*x5 <= 3500)

m.addConstr(4.6*x1 + 8.7*x2 + 9.7*x3 + 19.7*x4 + 26.7*x5 >= 75)
m.addConstr(4.6*x1 + 8.7*x2 + 9.7*x3 + 19.7*x4 + 26.7*x5 <= 150)

m.addConstr(x1>=0)
m.addConstr(x2>=0)
m.addConstr(x3>=0)
m.addConstr(x4>=0)
m.addConstr(x5>=0)

# Solving the model
m.optimize()

#  Print optimal solutions and optimal value
for v in m.getVars():
    print(v.VarName,":", v.x)

print('Obj:', m.objVal)
```

```
Optimize a model with 15 rows, 5 columns and 55 nonzeros
Coefficient statistics:
  Matrix range     [1e+00, 7e+02]
  Objective range  [3e-01, 1e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+01, 4e+03]
Presolve removed 10 rows and 0 columns
```

```
Presolve time: 0.02s
Presolved: 5 rows, 10 columns, 30 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   9.562500e+01   0.000000e+00      0s
       3    2.9998792e+00   0.000000e+00   0.000000e+00      0s

Solved in 3 iterations and 0.03 seconds
Optimal objective  2.999879183e+00
Apple : 0.0
Banana : 6.052917723812976
Blueberries : 0.0
Durian : 0.0
Tangerine : 2.3680077322701463
Obj: 2.9998791832789657
```

In [7]:

```
m = Model("diet")

# Creat variables
# addVar(lb=0.0, ub=GRB.INFINITY, obj=0.0, vtype=GRB.CONTINUOUS (variable type is CONTINUOUS), name
# lb: lower bound, ub: upper bound
# vtype: continuous, binary or integer
# name: name for the variable
x1 = m.addVar(name = "Apple")
x2 = m.addVar(name = "Banana")
x3 = m.addVar(name = "Blueberries")
x4 = m.addVar(name = "Durian")
x5 = m.addVar(name = "Tangerine")
# Set objective
# setObjective ( expr, sense=None )
# expr: linear or quadratic expression
# sense: GRB.MINIMIZE or GRB.MAXIMIZE
m.setObjective(0.5*x1 + 0.3*x2 + 2.5*x3 + 10*x4 + 0.5*x5, GRB.MINIMIZE)

# Add constraint:
m.addConstr(52*x1 + 89*x2 + 57*x3 + 147*x4 + 53*x5 >= 500)
m.addConstr(52*x1 + 89*x2 + 57*x3 + 147*x4 + 53*x5 <= 3000)

m.addConstr(14*x1 + 23*x2 + 14*x3 + 27*x4 + 13*x5 >= 50)
m.addConstr(14*x1 + 23*x2 + 14*x3 + 27*x4 + 13*x5 <= 400)

m.addConstr(2.4*x1 + 2.6*x2 + 2.4*x3 + 3.8*x4 + 1.8*x5 >= 20)
m.addConstr(2.4*x1 + 2.6*x2 + 2.4*x3 + 3.8*x4 + 1.8*x5 <= 30)

m.addConstr(54*x1 + 64*x2 + 54*x3 + 44*x4 + 681*x5 >= 2000)
m.addConstr(54*x1 + 64*x2 + 54*x3 + 44*x4 + 681*x5 <= 3500)

m.addConstr(4.6*x1 + 8.7*x2 + 9.7*x3 + 19.7*x4 + 26.7*x5 >= 75)
m.addConstr(4.6*x1 + 8.7*x2 + 9.7*x3 + 19.7*x4 + 26.7*x5 <= 150)

m.addConstr(0.2*x1 + 0.3*x2 + 0.3*x3 + 5*x4 + 0.3*x5 >= 0)
m.addConstr(0.2*x1 + 0.3*x2 + 0.3*x3 + 5*x4 + 0.3*x5 <= 10)

m.addConstr(x1>=0)
m.addConstr(x2>=0)
m.addConstr(x3>=0)
m.addConstr(x4>=0)
m.addConstr(x5>=0)

# Solving the model
m.optimize()

#  Print optimal solutions and optimal value
for v in m.getVars():
    print(v.VarName,":", v.x)

print('Obj:', m.objVal)
```

```
Optimize a model with 17 rows, 5 columns and 65 nonzeros
Coefficient statistics:
  Matrix range     [2e-01, 7e+02]
  Objective range  [3e-01, 1e+01]
```

```
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+01, 4e+03]
Presolve removed 11 rows and 0 columns
Presolve time: 0.02s
Presolved: 6 rows, 9 columns, 34 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   9.562500e+01   0.000000e+00      0s
       3    2.9998792e+00   0.000000e+00   0.000000e+00      0s

Solved in 3 iterations and 0.03 seconds
Optimal objective  2.999879183e+00
Apple : 0.0
Banana : 6.052917723812976
Blueberries : 0.0
Durian : 0.0
Tangerine : 2.3680077322701463
Obj: 2.9998791832789657
```

Q5(b)

In [1]:

```
from gurobipy import *
import numpy as np

#########Parameters Set-up############

#the vector of prices
price = np.array([60, 54, 48, 36])
#the vector of demands
demand = np.array([125, 162.5, 217.5, 348.8])
#salvage value
s = 25
#total number of inventory
I = 1975
#Time horizon
T = 14
#full price week
#full_price_week = 1

#number of price levels
N = len(price)
```

In [2]:

```
#########Model Set-up###############

m = Model("Retail")

# number of weeks to offer price level i
x = m.addVars(N, name = "x")

# set objective
m.setObjective( quicksum(price[i]*demand[i]*x[i] for i in range(N)) + s*(I - quicksum(demand[i]*x[i]

# capcity constraint:
m.addConstr( quicksum(demand[i]*x[i] for i in range(N)) <= I , "capacity")

# time constraint:
m.addConstr( quicksum(x[i] for i in range(N)) <= T , "time")

# full price constraint:
#m.addConstr( x[0] >= full_price_week , "full_price")

# Solving the model
m.optimize()

#  Print optimal solutions and optimal value
print("\n Optimal solution:")
for v in m.getVars():
    print(v.VarName, v.x)

print("\n Optimal profit:")
print('Obj:', m.objVal)

#  Print optimal dual solutions
print("\n Dual solutions:")
for d in m.getConstrs():
    print('%s %g' % (d.ConstrName, d.Pi))
```

```
Academic license - for non-commercial use only
Optimize a model with 2 rows, 4 columns and 8 nonzeros
Coefficient statistics:
  Matrix range     [1e+00, 3e+02]
  Objective range  [4e+03, 5e+03]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+01, 2e+03]
Presolve time: 0.03s
Presolved: 2 rows, 4 columns, 8 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.0005250e+33   6.153906e+30   1.000525e+03      0s
       3    1.1265000e+05   0.000000e+00   0.000000e+00      0s

Solved in 3 iterations and 0.04 seconds
Optimal objective  1.126500000e+05

 Optimal solution:
x[0] 8.0
x[1] 6.0
x[2] 0.0
x[3] 0.0
```

```
 Optimal profit:
Obj: 112650.0

 Dual solutions:
capacity 9
time 3250
```

In [4]:

```python
# if stick to the old strategy, the revenue will be
r = 25*60+10.67*125*60+3.33*162.5*54+25*(2000-25-10.67*125-3.33*162.5)
print("the revenue if stick to the old strategy is:", r)
```

```
the revenue if stick to the old strategy is: 113248.875
```

Q5(c)

In [5]:

```python
from gurobipy import *
import numpy as np

#########Parameters Set-up############

#the vector of prices
price = np.array([60, 58, 56, 54, 52, 50, 48, 46, 44, 42, 40, 38, 36])
#the vector of demands
demand = np.array([125, 137.5, 150, 162.5, 180.8, 199.1, 217.5, 239.4, 261.3, 283.2, 305.1, 327, 348

#salvage value
s = 25
#total number of inventory
I = 2000
#Time horizon
T = 15
#full price week
full_price_week = 1

#number of price levels
N = len(price)
```

In [6]:

```
#########Model Set-up###############

m = Model("Retail")

# number of weeks to offer price level i
x = m.addVars(N, name = "x")

# set objective
m.setObjective( quicksum(price[i]*demand[i]*x[i] for i in range(N)) + s*(I - quicksum(demand[i]*x[i]

# capcity constraint:
m.addConstr( quicksum(demand[i]*x[i] for i in range(N)) <= I , "capacity")

# time constraint:
m.addConstr( quicksum(x[i] for i in range(N)) <= T , "time")

# full price constraint:
m.addConstr( x[0] >= full_price_week , "full_price")

# Solving the model
m.optimize()

#  Print optimal solutions and optimal value
print("\n Optimal solution:")
for v in m.getVars():
    print(v.VarName, v.x)

print("\n Optimal profit:")
print('Obj:', m.objVal)

#  Print optimal dual solutions
print("\n Dual solutions:")
for d in m.getConstrs():
    print('%s %g' % (d.ConstrName, d.Pi))
```

```
Optimize a model with 3 rows, 13 columns and 27 nonzeros
Coefficient statistics:
  Matrix range     [1e+00, 3e+02]
  Objective range  [4e+03, 5e+03]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 2e+03]
Presolve removed 1 rows and 0 columns
Presolve time: 0.02s
Presolved: 2 rows, 13 columns, 26 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.2000000e+05   1.587200e+01   0.000000e+00      0s
       2    1.1725000e+05   0.000000e+00   0.000000e+00      0s

Solved in 2 iterations and 0.04 seconds
Optimal objective  1.172500000e+05

 Optimal solution:
x[0] 5.0
x[1] 10.0
x[2] 0.0
x[3] 0.0
x[4] 0.0
```

```
x[5] 0.0
x[6] 0.0
x[7] 0.0
x[8] 0.0
x[9] 0.0
x[10] 0.0
x[11] 0.0
x[12] 0.0

 Optimal profit:
Obj: 117250.0

 Dual solutions:
capacity 13
time 2750
full_price 0
```

In [ ]: