```
#1.ASSEMBLYLINE SCHEDULING
Def assemblyLine(a, t, e, x, n):
    T1 = [0] * n
    T2 = [0] * n

    T1[0] = e[0] + a[0][0]
    T2[0] = e[1] + a[1][0]

    For i in range(1, n):
        T1[i] = min(T1[i-1] + a[0][i], T2[i-1] + t[1][i] + a[0][i])
        T2[i] = min(T2[i-1] + a[1][i], T1[i-1] + t[0][i] + a[1][i])

    Return min(T1[n-1] + x[0], T2[n-1] + x[1])
A = [[4, 5, 3, 2], [2, 10, 1, 4]]
T = [[0, 7, 4, 5], [0, 9, 2, 8]]
E = [10, 12]
X = [18, 7]
N = 4

Print(assemblyLine(a, t, e, x, n))

#2.knapsack
Def knaps(wt, val, cap):
    N = len(val)
    Dp = [0] * (cap + 1)
    For i in range(n):
        For w in range(cap, wt[i] – 1, -1):
            Dp[w] = max(dp[w], dp[w – wt[i]] + val[i])
```

```
  Return dp[cap]


Wt = list(map(int, input(“Enter weights separated by spaces: “).split()))

Val = list(map(int, input(“Enter values separated by spaces: “).split()))

Cap = int(input(“Enter capacity: “))

Print(knaps(wt, val, cap))


#3.bellman and ford

Class Graph:


  Def __init__(self, vertices):

    Self.V = vertices

    Self.graph = []

  Def addEdge(self, u, v, w):

    Self.graph.append([u, v, w])

  Def printArr(self, dist):

    Print(“Vertex Distance from Source”)

    For i in range(self.V):

      Print(“{0}\t\t{1}”.format(i, dist[i]))

  Def BellmanFord(self, src):

    Dist = [float(“Inf”)] * self.V

    Dist[src] = 0

    For _ in range(self.V – 1):

      For u, v, w in self.graph:

        If dist[u] != float(“Inf”) and dist[u] + w < dist[v]:

          Dist[v] = dist[u] + w

    For u, v, w in self.graph:

      If dist[u] != float(“Inf”) and dist[u] + w < dist[v]:
        Print(“Graph with negative wt cycle”)
        Return
    Self.printArr(dist)
If __name__ == ‘__main__’:
  G = Graph(5)
  g.addEdge(0, 1, -1)
  g.addEdge(0, 2, 4)
  g.addEdge(1, 2, 3)
  g.addEdge(1, 3, 2)
  g.addEdge(1, 4, 2)
  g.addEdge(3, 2, 5)
  g.addEdge(3, 1, 1)
  g.addEdge(4, 3, -3)
  g.BellmanFord(0)
```

```
#WARSHALL AND FLOYD
V = 4
INF = 99999
Def floydWarshall(graph):
    Dist = list(map(lambda i: list(map(lambda j: j, i)), graph))
    For k in range(V):
        For i in range(V):
            For j in range(V):
                Dist[i][j] = min(dist[i][j],dist[i][k] + dist[k][j])
    printSolution(dist)
def printSolution(dist):
    print(“matrix shows shortest dis btw evry pair of vertices”)
    for i in range(V):
```

```
    for j in range(V):
      if(dist[i][j] == INF):
        print(“%7s” % (“INF”), end=” “)
      else:
        print(“%7d\t” % (dist[i][j]), end=’ ‘)
      if j == V-1:
        print()
if __name__ == “__main__”:
  graph = [[0, 5, INF, 10],[INF, 0, 3, INF],[INF, INF, 0, 1],[INF, INF, INF, 0]]
floydWarshall(graph)


#output:35
“””Enter weights separated by spaces: 7 6 13
Enter values separated by spaces: 70 60 130
Enter capacity: 200
260
Vertex Distance from Source
0              0
1              -1
2              2
3              -2
4              1
Matrix shows shortest dis btw evry pair of vertices
   0       5       8       9
  INF     0        3       4
  INF    INF     0         1
  INF    INF    INF     0        “””
```