2)MIN-MAX :

findMinMax(array)

minValue = array[0]

maxValue = array[0]

for i = 1 to array.length - 1

if array[i] < minValue

minValue = array[i]

if array[i] > maxValue

maxValue = array[i]

return (minValue, maxValue)

3)QUICK SORT :

quickSort(array, low, high)

if low < high

pivotIndex = partition(array, low, high)

quickSort(array, low, pivotIndex - 1)

quickSort(array, pivotIndex + 1, high)

partition(array, low, high)

pivot = array[high]

i = low - 1

for j = low to high - 1

if array[j] <= pivot

i = i + 1

swap array[i] and array[j]

swap array[i + 1] and array[high]

return i + 1

6)GREEDY KNAPSACK ALG:

fractionalKnapsack(items, maxWeight)

sort items in descending order based on value-to-weight ratio

knapsackWeight = 0

totalValue = 0

for i = 0 to items.length - 1

if knapsackWeight + items[i].weight <= maxWeight

knapsackWeight += items[i].weight

totalValue += items[i].value

else

fraction = (maxWeight - knapsackWeight) / items[i].weight

knapsackWeight = maxWeight

totalValue += items[i].value \* fraction

break

return totalValue

8)DIJKSTRA’S :

function Dijkstra(Graph, source):

dist[source] = 0

for vertex in Graph:

if vertex != source:

dist[vertex] = infinity

prev[vertex] = None

queue = PriorityQueue()

queue.put(source, 0)

while not queue.empty():

u = queue.get()

for v, weight in Graph[u].neighbors:

alt = dist[u] + weight

if alt < dist[v]:

dist[v] = alt

prev[v] = u

queue.put(v, alt)

return dist, prev

8)BELLMANN FORD :  
function BellmanFord(Graph, source):

dist[source] = 0

for i in range(len(Graph) - 1):

for u, v, weight in Graph.edges:

if dist[u] != infinity and dist[u] + weight < dist[v]:

dist[v] = dist[u] + weight

for u, v, weight in Graph.edges:

if dist[u] != infinity and dist[u] + weight < dist[v]:

return False, dist, prev

return True, dist, prev

9)OPTIMAL PATH :

function TapeStorage(files, tape\_capacity):

files.sort(key=lambda x: -x[1]) # sort files in descending order by size

tapes = []

for file in files:

placed = False

for tape in tapes:

if tape[0] + file[1] <= tape\_capacity: # if file fits on tape

tape[0] += file[1] # update tape usage

tape[1].append(file) # add file to tape

placed = True

break

if not placed: # if file could not be placed on existing tapes

tapes.append([file[1], [file]]) # create a new tape and add file to it

return tapes