**Dictionary Countries MergeSort QuickSort**

# Define an empty dictionary to store cost data for all countries,

costData = {}

#List of food groups ( as per Table 1)

foodGroups = [“FRUITS”,”VEGETABLES”,”ANIMAL\_SOURCED\_FOODS”, “LEGUMES\_NUTS\_AND\_SEEDS”, “OILS\_AND\_FATS”, “STARCHY\_STAPLES”]

#List of countries (as per Table 1)

countries =[“UK”, “FRANCE”,”GERMANY”, “ITALY”, “SPAIN”]

# countryMap : map country name / index to country

# foodGroupMap : map food group name / index to foodGroup

**Task 1**

STORE-USER-INPUT-IN-A-LIST(countries[1..nc], foodGroup[1..nf], countryMap, foodGroupMap)

1 Input[1..nc] <- initialise a list of String arrays

2 print(“Enter cost data for each country…”)

3 for i = 1 to nc

4 costs[nf] <- initialise a String array of size foodGroup.length

5 for j = 1 to nf

6 print(“Enter cost of “ + foodGroupMap(j).name + “ in “ + countryMap(i).name + “: “

7 costs[j] = input()

8 add costs to Input

VALIDATE-AND-REPLACE-INVALID-DATA(Input[1..nc], countryMap, foodGroupMap)

1 InvalidIndexes <- initialise a list of Integer arrays to store index-pair of invalid data

2 for i = 1 to nc

3 dataArray = Input[i] <- array in size nf

4 for j = 1 to nf

5 value = dataArray[j]

6 try

7 parseValue = PARSE-DOUBLE(value)

8 if parsedValue < 0

9 dataArray[j] = “0”

10 print(countryMap(i).name + “-“ + foodGroupMap(j).name + “ value changed from “ + value + “ to 0”)

11 add {i,j} to InvalidIndexes

12 catch NumberFormationException

13 dataArray[j] = “0”

14 print(countryMap(i).name + “-“ + foodGroupMap(j).name + “ value changed from “ + value + “ to 0”)

15 add {i,j} to InvalidIndexes

16 if !InvalidIndexes.isEmpty()

17 print(“Some data points have been replaced by zero. Do you want to correct values? Y/N”)

18 choice = input()

19 if choice == ‘Y’ or choice == ‘y’

20 CORRECT-INVALID-VALUES(Input, InvalidIndexes)

CORRECT-INVALID-VALUES(Input[1..nc] , InvalidIndexes, countryMap, foodGroupMap)

1 for i=1 to InvalidIndexes.size

2 cIndex = InvalidIndexes[i][0]

3 fIndex = InvalidIndexes[i][1]

4 print(“Enter the correct value for “ + countryMap(cIndex).name + “ – “ + foodGroupMap(fIndex).name)

5 newValue = input()

6 while !IS-VALID-DOUBLE(newValue) or PARSE-DOUBLE(newValue) <= 0

7 if !IS-VALID-DOUBLE(newValue)

8 print(“Invalid input. Please enter a valid number”)

9 else if PARESE-DOUBLE(newValue) <= 0

10 print(“Value cannot be negative or zero. Please enter a positive value: “)

11 newValue = input()

12 oldValue = Input[cIndex][fIndex]

13 Input[cIndex][fIndex] = newValue

14 print(“Value changed from “ + oldValue + “ to “ + newValue)

PARSE-INPUT-TO-COST-FOR-EACH-COUNTRY’S-FOOD-GROUP(Input[1..nc],countries[1..nc]], countryMap, foodGroupMap)

1 VALIDATE-AND-REPLACE-INVALID-DATA(Input, countryMap, foodGroupMap)

2 for i = 1 to nc

3 costDataString[1..nf] = Input[i]

4 costData[i] = {} 🡨 Create an empty dictionary for the current country

4 for j = 1 to nf

5 cost = PARSE-DOUBLE(costDataString[j])

6 costData[i][j] = cost

IS-VALID-DOUBLE(value)

1 try

2 doubleValue = PARSE-DOUBLE(value)

3 return TRUE

4 catch NumberFormationException

5 return FALSE

PARSE-DOUBLE(value)

1 if value can be converted to a double without error

2 return the value in double

3 else error “InvalidInputError”

INPUT-COST-FOR-EACH-COUNTRY’S-FOOD-GROUP(countries[], foodGroup[])

1 for each c ∈ countries:

2 costData[c] = {} 🡨 Create an empty dictionary for the current country

3 for each f ∈ foodGroups:

4 print “Enter the cost for “ + f.name + “ in “ + c.name + “: “

5 try: cost = input()

6 except InvalidInputError: error “Invalid input. Please enter a positive numeric value.”

7 costData[c][f] = cost

# Define an empty dictionary to store cost data for all countries,

shareData = {}

# foodGroupMap : map foodGroup name to foodGroup

**Task 2**

INPUT-SHARE-FOR-EACH-COUNTRY’S-FOOD-GROUP(countries[], foodGroups[])

1 for each c ∈ countries:

2 shareData[c] = {} 🡨 Create an empty dictionary for the current country

3 for each f ∈ foodGroups:

4 print “Enter the share for “ + f + “ in “ + c + “: “

5 try: share = input()

6 except InvalidInputError: error( “Invalid input. Please enter a positive numeric value.”

7 shareData[c][f] = share/100

**Task 3**

DISPLAY-TOTAL-COST-OF-ANIMAL-SOURCED-FOODS(countries[],costData[][])

1 totalCost = 0

2 for each c ∈ countries

3 totalCost = totalCost + costData[c][“ANIMAL\_SOURCED\_FOODS”]

4 print “Total cost of animal-sourced food of all countries: “ + totalCost

**Task 4**

DISPLAY-TOTAL-DIET-COST-AND-FIND-MINIMUM(countries[], foodGroups[], costData[][])

1 minCountry = NIL

2 min = +∞

3 for each c ∈ countries

4 totalCost = 0

5 for all f ∈ foodGroups

6 totalCost = totalCost + costData[c][f]

7 if totalCost < min

8 min = totalCost

9 minCountry = c

10 print “Total cost of diet for “ + c.name + “: £” + totalCost”

11 print “Minimum cost: £” + min +”, in “ + minCountry.name

**Task 5**

DISPLAY-SORTED-COUNTIES-BY-FRUIT-SHARE(countrie[], shareData[][])

1 sortedCountries[] = MERGE-SORT-COUNTRIES-BY-FRUIT-SHARE (countries,shareData)

2 for each c ∈ sortedCountries

3 print c.name + “: “+ shareData[c][“FRUIT”] \*100 + “%”

MERGE-SORT-COUNTRIES-BY-FRUIT-SHARE(countries[], shareData[][])

1 if countries.length <= 1

2 return countries

3 mid =

4 left[] = MERGE-SORT-COUNTRIES-BY-FRUIT-SHARE(countries[0…mid-1], shareData)

5 right[] = MERGE-SORT-COUNTIRES-BY-FRUIT-SHARE(countries[mid…countries.length-1], shareDate)

6 return MERGE(left,right,shareData)

MERGE(left[], right[], shareData[])

1 i = 0

2 j = 0

3 k = 0

4 result[left.length + right.length]

5 while i < left.length and j < right.length

6 if shareData[left[i]][FRUIT] >= shareData[right[j]][FRUIT]

7 result[k] = left[i]

8 k = k + 1

9 i = i + 1

10 else result[k] = right[j]

11 k = k +1

12 j = j + 1

13 while i < left.length

14 result[k] = left[i]

15 k = k + 1

16 i = i + 1

17 while j < right.length

18 result[k] = right[j]

19 k = k + 1

20 j = j + 1

**Task 6**

DISPLAY-SORTED-DIET-DATA-IN-GERMANY-BY-FOOD-SHARE(countrie[], foodGroups[] costData[][],shareData[][])

1 gIndex = getCountryIdByName (“Germany”,countries) 🡨from map

2 foodGroupSharePairs[] = GET-FOODGROUP-SHARE-PAIRS(gIndex, foodGroups,shareData) 🡨 from map

3 QUICK-SORT-FOODGROUP-BY-SHARE(foodGroupSharePairs, 0, foodGroupSharePairs.length-1)

4 for each f ∈ foodGroupSharePairs

5 foodGroup = f.foodGroup

6 print foodGroup.name + “ with cost, £” + costData[gIndex][foodGroup] + “ and share, “ + (f.share\*100) +”%”

GET-FOODGROUP-SHARE-PAIRS(gIndex, foodGroups[], shareData[][])

1 pairs[] = {}

2 for each foodGroup ∈ foodGroups 🡨 foodGroup could be int index

3 share = shareData[gIndex][foodGroup]

4 add (foodGroup, share) to pairs[]

5 return pairs

QUICK-SORT-FOODGROUP-BY-SHARE(pairs[], left, right)

1 if left < right

2 pivotIndex = PARTITION-FOODGROUP-BY-SHARE(pairs, left, right)

3 QUICK-SORT-FOODGROUP-BY-SHARE(pairs, left, pivotIndex-1)

4 QUICK-SORT-FOODGROUP-BY-SHARE(pairs, pivotIndex + 1, right)

PARTITION-FOODGROUP-BY-SHARE(pairs[], left, right)

1 pivotIndex = CHOOSE-RANDOM-PIVOT-INDEX(left, right)

2 SWAP-PAIRS(pairs, pivotIndex, right)

3 pivot = pairs[right].share

4 i = left – 1

5 for j = left to right – 1

6 if pairs[j].share >= pivot

7 i = i +1

8 SWAP-PAIRS(pairs, i, j)

9 SWAP-PAIRS(pairs, i+1, right)

10 return i+1

SWAP-PAIRS(pairs[], i, j)

1 tempPair = pairs[i]

2 pairs[i] = pairs[j]

3 pairs[j] = tempPair

CHOOSE-RANDOM-PIVOT-INDEX(left, right)

1 return RANDOM(right – left) + left

**Task 7**

SEARCH-COUNTRY-FOR-ITS-COST-OF-OILS-AND-FATS(countries[], foodGroups[], costData[][], shareData[][])

1 print(“Enter country name: “)

2 countryName = input()

3 countryIndex = LOOKUP-COUNTRY-ID(countryName, countries)

4 if countryIndex == -1

5 print “Country: “ + countryName + “ not found”

6 else

7 oilAndFatIndex = LOOKUP-FOOD-GROUP-ID(“OILS\_AND\_FATS”, foodGroups)

8 if oilAndFatIndex == -1

9 print “Food group : ‘OILS\_AND\_FATS’ not found”

10 else

11 cost = costData[countryIndex][oilAndFatIndex]

12 share = shareData[countryIndex][oilAndFatIndex]

13 print “The cost of oils and fats in “ + countryName + “ is £” + cost

14 print “The share of oils and fats in “ + countryName + “ is “ + (share\*100) + “%”

lookupCountryID (countryName)

1 return countryIDMap.get(countryName)

lookupFoodGroupID (foodGroupName)

1 return foodGroupIDMap.get(foodGroupName)

**Task 8**

SEARCH-COUNTRY-FOR-ITS-COST-OF-FOOD-GROUPS-AND-TOTAL-COST(countries[], foodGroups[], costData[][], shareData[][], lookupCountryID, lookupFoodGroupID)

1 print “Enter country name: ”

2 countryName = input()

3 countryIndex = lookupCountryID(countryName)

4 if countryIndex == -1

5 print “Country: “ + countryName + “ not found”

6 else

7 totalCost = 0

8 for each f ∈ foodGroups

9 foodGroupIndex = lookupFoodGroupID(f)

10 if foodGroupIndex == -1

11 print(“Food group: “ + f.name + “ not found.”

12 else

13 cost = costData[countryIndex][foodGroupIndex]

14 share = shareData[countryIndex][foodGroupIndex]

15 costShare = cost \* share

16 totalCost = totalCost + costShare

17 print “Least cost of healthy diet for “ + f.name + “: £” + cost + “ \* “ + (share\*100) + “% = £” + costShare

18 print “Total cost of a healthy diet in “ + countryName + “: £” + totalCost

lookupCountryID (countryName) 🡨 direct mapping

1 return countryIDMap.get(countryName)

lookupFoodGroupID (foodGroup) 🡨 direct mapping

1 return foodGroupIDMap.get(foodGroup)

**Dictionary Countries HeapSort QuickSort with three partitions**

**Task 5**

DISPLAY-SORTED-COUNTIES-BY-FRUIT-SHARE(countrie[], shareData[][], fruitGroupName)

1 BUILD-MAX-HEAP(countries, shareData, fruitGroupName)

2 for i = 0 to countries.length – 1

3 maxCountry = HEAP-EXTRACT-MAX(countries, shareData, fruitGroupName)

4 print maxCountry.name + “: “ + (shareData[maxCountry][fruitGroupName] \* 100) + “%”

BUILD-MAX-HEAP(countries[], shareData[][], fruitGroupName]

1 heapSize = countries.length

2 for i = down to 1

3 MAX-HEAPIFY(countries, i, shareData, fruitGroupName)

MAX-HEAPIFY(counties[], i, shareData[][], fruitGRoupName)

1 l = left(i)

2 r = right(i)

3 if l <= heapSize and shareData[counties[l]][fruitGroupName] > shareData[countries[r]][fruitGroupName]

4 largest = l

5 else

6 largest = r

7 if r <= heapSize and shareData[countries[r]][fruitGroupName] > shareData[countries[largest]][fruitGroupName]

8 largest = r

9 if largest != i

10 SWAP(countries, i, largest)

11 MAX-HEAPIFY(countries, largest, shareData, fruitGroupName)

SWAP(countries[], i, j)

1 tempCountry = countries[i]

2 countries[i] = countries[j]

3 countries[j] = tempCountry

HEAP-EXTRACT-MAX(countries[], shareData[][], fruitGroupName)

1 if heapSize < 1

2 print “heap underflow”

3 maxCountry = countries[0]

4 countries[0] = countries[heapSize-1]

5 heapSize = heapSize – 1

6 MAX-HEAPIFY(countrie, 1, shareData, fruitGroupName)

7 return maxCountry

**Task 6**

DISPLAY-SORTED-DIET-DATA-IN-GERMANY-BY-FOOD-SHARE(countrie[], foodGroups[] costData[][],shareData[][])

1 gIndex = getCountryIdByName (“Germany”,countries) 🡨from map

2 foodGroupSharePairs[] = GET-FOODGROUP-SHARE-PAIRS(gIndex, foodGroups,shareData) 🡨 from map

3 QUICK-SORT-FOODGROUP-BY-SHARE(foodGroupSharePairs, 0, foodGroupSharePairs.length-1)

4 for each f ∈ foodGroupSharePairs

5 foodGroup = f.foodGroup

6 print foodGroup.name + “ with cost, £” + costData[gIndex][foodGroup] + “ and share, “ + (f.share\*100) +”%”

GET-FOODGROUP-SHARE-PAIRS(gIndex, foodGroups[], shareData[][])

1 pairs[] = {}

2 for each foodGroup ∈ foodGroups 🡨 foodGroup could be int index

3 share = shareData[gIndex][foodGroup]

4 add (foodGroup, share) to pairs[]

5 return pairs

QUICK-SORT-FOODGROUP-BY-SHARE(pairs[], left, right)

1 if left < right

2 equalIndexPair = PARTITION-FOODGROUP-BY-SHARE(pairs, left, right)

3 QUICK-SORT-FOODGROUP-BY-SHARE(pairs, left, equalIndexPair[0])

4 QUICK-SORT-FOODGROUP-BY-SHARE(pairs, equalIndexPair[1] + 1, right)

PARTITION-FOODGROUP-BY-SHARE(pairs[], left, right)

1 equalIndexPair[2] 🡨 Intialise empty array of size 2

2 pivotIndex = CHOOSE-RANDOM-PIVOT-INDEX(left, right)

3 SWAP-PAIRS(pairs, pivotIndex, right)

4 pivot = pairs[right].share

5 i = left – 1

6 e = left -1

7 for j = left to right – 1

8 if pairs[j].share > pivot

9 i = i +1

10 e = e + 1

11 SWAP-PAIRS(pairs, i, j)

12 if i != e

13 SWAP-PAIRS(pairs, e, j)

14 elseif pair[j].share == pivot

15 e = e + 1

16 SWAP-PAIRS(pairs, e, j)

17 SWAP-PAIRS(pairs, e+1, right)

18 equalIndexPair[0] = i

19 equalIndexPair[1] = e+1

20 return equalIndexPair

SWAP-PAIRS(pairs[], i, j)

1 tempPair = pairs[i]

2 pairs[i] = pairs[j]

3 pairs[j] = tempPair

CHOOSE-RANDOM-PIVOT-INDEX(left, right)

1 return RANDOM(right – left) + left

**Dictionary Countries Binary Search Tree QuickSort with three partitions using While loop**

**Task 5**

DISPLAY-SORTED-COUNTIES-BY-FRUIT-SHARE(countrie[], shareData[][], fruitGroupName)

1 tree = BUILD -BINARY-SEARCH-TREE-BY-FRUIT-SHARE(countries, shareData, fruitGroupName)

2 REVERSE-INORDER-TREE-WALK-PRINT-FRUIT-SHARE(tree.root)

BUILD-BINARY=SEARCH-TREE-BY=FRUIT=SHARE(countries[], shareData[][], fruitGroupName)

1 tree 🡨 intialise a new tree

2 for each c ∈ countries

3 node 🡨 create new Node(c, shareData[c][fruitGroupName]

4 TREE-INSERT(tree, node)

5 return tree

TREE-INSERT(tree, node)

1 y = NIL

2 x = tree.root

3 while x != NIL

4 y = x

5 if node.share < x.share

6 x = x.left

7 else

8 x = x.right

9 node.parent = y

10 if y == NIL

11 tree.root = node

12 elseif node.share < y.share

13 y.left = node

14 else

15 y.right = node

REVERSE-INORDER-TREE-WALK-PRINT-FRUIT-SHARE(x)

1 if x != NIL

2 REVERSE-INORDER-TREE-WALK-PRINT-FRUIT-SHARE(x.right)

3 print x.country.name + “: “ + (x.share \*100) + “%”

4 REVERSE-INORDER-TREE-WALK-PRINT-FRUIT-SHARE(x.left)

**Task 6**

DISPLAY-SORTED-DIET-DATA-IN-GERMANY-BY-FOOD-SHARE(countrie[], foodGroups[] costData[][],shareData[][])

1 gIndex = getCountryIdByName (“Germany”,countries) 🡨from map

2 foodGroupSharePairs[] = GET-FOODGROUP-SHARE-PAIRS(gIndex, foodGroups,shareData) 🡨 from map

3 QUICK-SORT-FOODGROUP-BY-SHARE(foodGroupSharePairs, 0, foodGroupSharePairs.length-1)

4 for each f ∈ foodGroupSharePairs

5 foodGroup = f.foodGroup

6 print foodGroup.name + “ with cost, £” + costData[gIndex][foodGroup] + “ and share, “ + (f.share\*100) +”%”

GET-FOODGROUP-SHARE-PAIRS(gIndex, foodGroups[], shareData[][])

1 pairs[] = {}

2 for each foodGroup ∈ foodGroups 🡨 foodGroup could be int index

3 share = shareData[gIndex][foodGroup]

4 add (foodGroup, share) to pairs[]

5 return pairs

QUICK-SORT-FOODGROUP-BY-SHARE(pairs[], left, right)

1 if left < right

2 equalIndexPair = PARTITION-FOODGROUP-BY-SHARE(pairs, left, right)

3 QUICK-SORT-FOODGROUP-BY-SHARE(pairs, left, equalIndexPair[0])

4 QUICK-SORT-FOODGROUP-BY-SHARE(pairs, equalIndexPair[1] + 1, right)

PARTITION-FOODGROUP-BY-SHARE(pairs[], left, right)

1 equalIndexPair[2] 🡨 Intialise empty array of size 2

2 pivotIndex = CHOOSE-RANDOM-PIVOT-INDEX(left, right)

3 SWAP-PAIRS(pairs, pivotIndex, right)

4 pivot = pairs[right].share

5 i = left – 1 🡨track elements less than pivot

6 j = left 🡨iterate through the list

7 k = right 🡨tracks elements greater than pivot

8 while j < k

9 if pairs[j].share > pivot

10 i = i +1

11 SWAP-PAIRS(pairs, i, j)

12 j = j +1

13 elseif pairs[j].share < pivot

14 k = k -1

15 SWAP-PAIRS(pairs, j, k)

16 else

17 j = j + 1

18 SWAP-PAIRS(pairs, k, right) 🡨move the pivot to the correct

18 equalIndexPair[0] = i

19 equalIndexPair[1] = k

20 return equalIndexPair

SWAP-PAIRS(pairs[], i, j)

1 tempPair = pairs[i]

2 pairs[i] = pairs[j]

3 pairs[j] = tempPair

CHOOSE-RANDOM-PIVOT-INDEX(left, right)

1 return RANDOM(right – left) + left

Testing validateAndReplaceInvalidData

Input: {"0.53", "0.40", "0.49", "0.20", "0.05", "0.15"}

Expected output: {"0.53", "0.40", "0.49", "0.20", "0.05", "0.15"}

Sending an array with all acceptable values should return an array with the same values

Input: {" ", "1B", "-23", "21\*", "0.05", "0.15"}

Expected output: {“0”, “0”, “0”, “0”, “0.05”, “0.15”}

Sending an array with unacceptable vlaues, “ “, “1B”, “-23”, “21\*”, should return an array with these values replaced by “0”

Testing validateAndReplaceInvalidSales with user answering Yes/No:

Input: “y” or “Y”

Expected Output: “Enter the correct value for COUNTRY NAME – FOOD GROUP NAME : “

Answering Y or y means Yes for correcting invalid data.

Input: n or N or asdqd

Expected Output: do nothing

Answering with any values other than y or Y will be treated as a No , so do nothing

Testing correctInvalidValues with user input after answering Yes to validateAndReplaceInvalidSales:

Intput: 0.53

Expected Output: Value changed from 0 to 0.53

Answering with a valid value, will change from 0 to this value

Input: -23

Expected output: Value cannot be negative. Please enter a positive value :

Answering with a negative value, will force user to re enter a new a positive value

Input: ()

Expected output: Invalid input. Please enter a valid number

Answering with an invalid value, will force user to re enter a valid value

Testing validateAndCorrectShareData

Input: {"30", "30", "9.4", "21.8", "2.2", "8.4"}

Expected output: The sum of all share element in COUNTRY NAME is not 100%. Please re-enter the value.

Enter value for FOOD GROUP NAME in COUNTRY NAME

Answering with an array with sum/100 not equal to 100% will force user to re-enter all values in this array

Testing validateAndCorrectShareData when reentering share value.

Input: -20

Expected output: Share value cannot be negative. Please enter a positive value :

Answering with a negative value, will force user to re enter a new a positive value

Input: \*\*

Expected output: Invalid input. Please enter a valid number

Answering with an invalid value, will force user to re enter a valid value

Input: 20

Expected output: change the value to “20”

Answering with an acceptable value, will change value to “20”