**Public**

Optimal Sort

enum type

* This is used to define what type the user wants the data sorted in. The possible values are INCREASING AND DECREASING.

Public static void sort(T[] comparable)

* Input: an array that has a type, which implements the comparable interface.
* Sorts the array in increasing order.

Output:

* Returns nothing

Public static void sort(T[] comparable, type sortType)

input:

* comparable = an array of a type that extends the comparable interface,
* sortType = to tell the function how to sort the array. type.INCREASING will sort it in an increasing manor. Type.DECREASING will sort in a decreasing manor.

Output:

* returns nothing

public static void sort (T[] comparables, int min, int max)

Input

* comparables = is an array of a type that extends the comparable interface,
* min = is the minimum index in the range of things that need to be sorted
* max = is the maximum index in the range of things that need to be sorted

Output:

* returns nothing

public static void sort (T[] comparables, int min, int max, type sortType)

Input

* comparables = is an array of a type that extends the comparable interface,
* min = is the minimum index in the range of things that need to be sorted
* max = is the maximum index in the range of things that need to be sorted
* sortType = to tell the function how to sort the array. type.INCREASING will sort it in an increasing manor. Type.DECREASING will sort in a Output:
* returns nothing

public static Boolean sorted(T[] comparables)

input:

* comparables = is an array of a type that extends the comparable interface,

output:

* returns a boolean value that represents whether the array is sorted or not.

public static Boolean sorted(T[] comparables, boolean sortType)

Input:

* comparables = is an array of a type that extends the comparable interface
* sortType = to tell the function how to sort the array. type.INCREASING will sort it in an increasing manor. Type.DECREASING will sort in a

output:

* returns a boolean value that represents whether the array is sorted or not.

boolean sorted(T[] commparables, int min, int max, type OptimalSort.type sortType)

Input:

* comparables = is an array of a type that extends the comparable interface,
* min = is the minimum index in the range of things that need to be sorted
* max = is the maximum index in the range of things that need to be sorted
* sortType = to tell the function how to sort the array. type.INCREASING will sort it in an increasing manor. Type.DECREASING will sort in a

Output:

* returns a boolean value that represents weather the array is sorted or not.

Void printArray(Object[] words)

Input:

* words = an array

Output:

- returns nothing,

This function prints the array in the console. It is used for debugging purposes.

Insertion

Void show(Object[] words)

Input:

* words = an array

Output:

- returns nothing,

This function prints the array in the console. It is used for debugging purposes.

Public static void sort(T[] comparable)

* Input: an array that has a type, which implements the comparable interface.
* Sorts the array in increasing order.

Output:

* Returns nothing

Public static void sort(T[] comparable, type sortType)

input:

* comparable = an array of a type that extends the comparable interface,
* sortType = to tell the function how to sort the array. type.INCREASING will sort it in an increasing manor. Type.DECREASING will sort in a decreasing manor.

Output:

* returns nothing

public static void sort (T[] comparables, int min, int max)

Input

* comparables = is an array of a type that extends the comparable interface,
* min = is the minimum index in the range of things that need to be sorted
* max = is the maximum index in the range of things that need to be sorted

Output:

* returns nothing

public static void sort (T[] comparables, int min, int max, type sortType)

Input

* comparables = is an array of a type that extends the comparable interface,
* min = is the minimum index in the range of things that need to be sorted
* max = is the maximum index in the range of things that need to be sorted
* sortType = to tell the function how to sort the array. type.INCREASING will sort it in an increasing manor. Type.DECREASING will sort in a Output:
* returns nothing

UploadFile

String filePath

* Stores the path to the file that has been uploaded
* Is used by the parser to parse the contents of the file into a data object which then goes to the view to draw

String filename

* Stores the name of the file
* This variable is part of the path, It is used to get the csv file to produce a graph

Data

Name

* This represents the name of the graph
* We do not currently use this variable but its existence has value because now adding more functionality will be easier

xaxis

* This represents name of the xaxis
* We do not currently use this variable but its existence has value because now adding more functionality will be easier

yaxis

* This represents the y axis
* We do not currently use this variable but its existence has value because now adding more functionality will be easier

data

* This represents the data the program will graph
* It has the type of an arraylist.
* The elements of the arraylist are of type Point

VIEW(JAVASCRIPT )

PieChart

width = represents the width of the area that will display the graph

height = represents the height of the area that will display the graph

radius = Represents the radius for the pie chart

x(var input) = this variable is a function that maps a given set of inputs into a given outputs. Currently it transforms the points with a square root applied to them. This simulates how in a circle large sets of values appear bigger then small sets of values

for example: If you have a set of input from 0 to 100 and an output of 0 to 500. It will map a given x value to an output.

For example: x(0) 🡪 0

x(100) 🡪 500

x(50) 🡪 353

y(var input) = This variable is a function that maps a given set of inputs into a given output. Currently it transforms the points with a square root applied to them. This simulates how in a circle large sets of values appear bigger then small sets of values. This is used position certain inputs to a given y value.

for example: If you have a set of input from 0 to 100 and an output of 0 to 750. It will map a given x value to an output.

For example: y(0) 🡪 0

y(100) 🡪 750

y(50) 🡪 530

b = the bound box, This represents the attributes of the bread crumbs which is the place that tells you the path and its percentage. It tells the width, the height, the spacing, and tip/tail

colors = This represents all the data objects and their respective colors. Currently the program generates a random color which will be displayed on the screen. That random color will get stored in this variable with its name and color respectively. At initialization there is only the root because the root will always exist in all cases

totalSize = total size of all the segments.

vis = This represents the chart itself. It is has the width and the height of the public variables called width and height. It is placed inside the chart div. and it is transformed width/2 to the left and height/2 to the right. This allows the entire graph to be displayed.

Partition = this creates a partition object.

arc = This can be called on a selection. It will configure the selection in a manor that will display it in its correct position using the x and y transformation functions. It will also make sure the values are not outside the bounds of the stage

node = holds the current rote of the program. This is used when the user chooses a certain category and that becomes the current node.

json = represents the json object that the graph needs to draw.

createVisualization

* This function takes in a json object and creates a graph of that json object.

Stash

* This stores the old x and dx variables. This is used when the graph gets transformed, this allows it to transform back.

arcTweenZoom

* This function allows the pie chart to grow and shrink. When the user clicks on one of its categories it will make that the center and show the rest of the data with respect to it

mouseover

* When the user’s mouse goes over the graph this function gets called. This allows that current path to be highlighted and everything else to become faded. It also displays the path on the top left and the percentage this piece is taking up. This can be called on an array of collections which will call this on whatever node goes over a mouse.

mouseleave

* Removes all the effects that were done by the mouseover function

getAncestors

* gets the path of the current node. Meaning all the objects that are over this one towards the center until it reaches the root node.

initializeBreadcrumbTrail

* This function displays the trail at the top that shows that path of the current node and its percentage

getRandomColor

* Returns a random color.

drawLegend

* draws the legend on the screen.

toggleLegend

- adds or removes the legend depending on if it was previously there.

buildHierchy2

- takes in a json object and returns a new json object in the correct format needed to display it on the screen