Amit Ner-Gaon & Oren Vaknin

6.3 Split pseudo-code

```
Split (value, axis)
    Small_col= new stack
    Bigger_side = biggerside(value, axis) // full function is down below
    If (bigger_side) \\ left side
        P= small_col.head
        While(p!= null&p.value<value)
            Small_col.push(p)
            Temp=p.next
            Original.delete_(p)
            P<-temp
        If (original.head.value = value)
            Original.delete(head)
    Else \\ right side
        P=list.tail
        While(p!= null& p.value>value)
            Small_col.push(p)
            Temp=p.prev
            Original.delete(p)
        If (original.tail.value = value)
            Original.delete(tail)
Return small_col
Biggerside (value, axis)
    Container left_p = list.head
        Right_p = list.tail
    While (left_p.value<val)
        If (right_p.value <= val)
            Return false
        Left_p = left_p.next
        Right_p = right_p.prev
```

Return true

6.1 & 6.2
DataStructure class is a linked list implementation of a 2D coordinate system.
Each node in the list contains a Point object, which represents a point in the 2D coordinate system.
The linked list is sorted in non-decreasing order by both the X-axis and the Y-axis values of the Point objects.
It is one by creating 2 "chains" so that one chain is sorted by X-axis and the other sorted by Y-axis.
The chains are made from the same nodes.
Every node has 2 sets of next and prev pointer. One of the set is regarding the X-axis and the other the Y-axis.

addPoint

The function first creates a new Container object to wrap the Point object,

If the new Container is the head of some chain, the function insert it to the first place and update the head.

If not, it iterates over each chain of the list to determine the appropriate insertion point for the new container and insert it there.

At the end it updates the minimum and maximum X and Y.

Run Time: O(n),n is the number of points already in the list. The function iterate through the list to find the insertion point.

getPointsInRangeRegAxis

The function returns an array of Point in range of values of axis's values. The array is sorted in non-decreasing order by the axis value. It creates an array with appropriate size. Then it goes thought the list and find the first node on the range. Then it adds the Point in range to the array.

Run Time O(n), where n is the number of nodes in the linked list. This is because the function must traverse through the list twice. First to count the number of nodes in range and second to add the appropriate Point objects to the output array.

getPointsInRangeOppAxis

The function returns an array of Point in range of values of axis's values. The array is sorted in non-decreasing order by the other axis value. It then creates an array of Point objects with a length equal to the number of nodes within the range. Next, it sets a pointer to the head of the list and iterates through the list until it finds the first node in range. Note that the function literates on the chain of the other axis value. Then it adds each node's data to the array until it has reached the end of the range.

Run Time O(n), where n is the number of nodes in the linked list. This is because the function must traverse through the list twice. First to count the number of nodes in range and second to add the appropriate Point objects to the output array.

getDensity

First the function calculates the length of the x-axis and y-axis. Then, if either the length of the x-axis or y-axis is zero, it returns -1 to indicate an error. Finally, it returns the density of the points in the space by dividing the number of points in the linked list by the product of the length of the x-axis and y-axis.

The runtime complexity is O(1). Because the function only performs a fixed number of operations. It achieve it by using the length value instead of calculate it.

narrowRange

First the function go over from the head of the list to the first node in range, deleting the nodes as it go throw. Second it go over backward from the tail of the list to the first node in range, deleting the nodes as it go throw.

Time complexity: O(|A|) where |A| is the number of points to be deleted. This is because it goes from the head of the list up to the first container the range and from the tail of the list backward up to the first container the range.

getLargestAxis

The function use axisLength function in order to calculate the axis' length. Then it compare them and return true if X axis is bigger and false otherwise.

Run time is O(1) since the axisLength function's run time is O(1). It archived it by using the head and tail of an axis chain, which hold the minimum and maximum axis' value.

getMedian

The function returns the median node of the linked list regard an axis chain. It first calculates the index of the median node by dividing the total number of nodes in the linked list by 2. Then, it iterates over the axis chain and returns the node at the calculated median index.

The runtime complexity is O(n), where n is the number in the list. This because its literates through half of the list without going back.