Data structure course – assignment #2
Theoretical questions section

6.1 Describing my implementation

For the task, I chose to implement the data structure with two Bi-directional Linked Lists, each one sorted by a different axis (X and Y), and make every corresponding item to point to its "twin" from the other list as well. In addition, head and tail pointers are included for each list.

```
//O(1)
void DataStructure();
```

- Create two empty bi-directional linked lists, one for X and one for Y.

```
//O(n)
void addPoint(Point point);
```

- Create two links, one for the X-list and one for the Y-list, make them point at each other.
- Then insert each one to its corresponding list its correct sorted place.
- The two links must be two different object, because otherwise, they will share the same previous and next pointers and it will mess up the sorting.

```
//O(n)
Point[] getPointsInRangeRegAxis(int min, int max, Boolean axis);
```

- Determine in which axis we are working on, and make sure min is less than max.
- Start from the head of the list and move to the next item until we have reached a link with the axis value not less than mim and count how many iteration we've done. If we've reached null (no items left) return empty array.
- Start from the tail of the list and move to the previous item until we found a link with axis value not greater than max and count how many iteration we've done.
- Starting from where the lower bound stopped, add the N next items of the list to an array and return it (N – the size of the linked list minus the counter from the head and tail traversals)

```
//O(n)
Point[] getPointsInRangeOppAxis(int min, int max, Boolean axis);
```

- Determine in which axis we are working on, and make sure min is less than max.
- Create a new bi-directional linked list to temporarily store the relevant points
- Start from the head of the list and for each link check if its relevant value is in range min <= value =< max, if so – add it to the temporary list
- Move all the points from the temporary list to a new points array and return it

```
//O(1)
double getDensity();
```

- Get the size of our data structure
- Calculate the distance between the two most far away points in respect to each axis
- Using the provided formula to calculate the density:
  D = n / sqrt(X-distance * Y-distance)

```
//O(|A|) - |A| is the number of points that will be deleted from the data structure
void narrowRange(int min, int max, Boolean axis);
```

- Determine in which axis we are working on.
- Start from the head of the list and continue until the current link's axis value is no less than min: remove the current link and move to its next.
- Start from the tail of the list and continue until the current link's axis value is not greater than max: remove the current link and move to its previous.

```
//O(1)
Boolean getLargestAxis();
```

- Calculate each axis' length by comparing the axis value of its head and tail
- Check which length is larger and return it

```
//O(n)
Container getMedian(Boolean axis);
```

- Check if the data structure is empty, if so return null
- Get the size of the list and divide it by 2 (the definition of a median)
- Start from the head of the corresponding axis and move size/2 times
- Return the link we've landed on

```
//O(min(n, |B|log|B|)) - B is the number of points in the strip.
//You may use Arrays.sort to implement this function, but you may not change the class Point for that sake
Point[] nearestPairInStrip(Container container, double width, Boolean axis);
```

- Get an array of all the points inside a strip centered at container's axis value with width **width** (inclusive) sorted by the other axis
- Check for edge cases (2 or less points)
- Split the array of point into right half and left half, and for each one find the closest pair of points in it
- Calculate the distance of the minimum pair: delta
- Find all the points in the strip centered at container with width delta
- For each point check the following 7 points in the array (geometric reason) and if you found a close pair among them set it as a new closest pair
- Return the closest pair

```
//O(min(n, |B|log|B|))
Point[] nearestPair();
```

- Determine which axis is larger
- Create container with the median of this axis
- Call the `nearestPairInStrip` method to find the closest pair of points with parameter **width** equals to the larger axis' length
- Return the provided pair

6.2 Analyzing the run time

```
//O(1)
void DataStructure();
```

- Creating an empty linked lists is a constant time, so 2*O(1) = O(1)

```
//O(n)
void addPoint(Point point);
```

- Create two links and making them point to each other is constant time O(1)
- Adding each link to its correct place in the list can take up to n step (if it's a new tail) so each insertion to sorted array is O(n)
- Therefore all the method is O(n)

```
//O(n)
Point[] getPointsInRangeRegAxis(int min, int max, Boolean axis);
```

- Going from the head of the list until the current link's axis value is not less than mim can be going over all the list in the worst case, if all the links' axis values is less than min, so the worst case scenario is O(n)
- All the other operation in the method are at most O(n) so the method is O(n) complexity

```
//O(n)
Point[] getPointsInRangeOppAxis(int min, int max, Boolean axis);
```

- Going over all the list is O(n) and adding the relevant points to an array can be O(n) in the worst case, where all the points are in the range.

```
//O(1)
double getDensity();
```

- Getting the length of each axis is O(1) because we have head and tail pointers
- Getting the size of the data structure is O(1) because we have dedicated size variable we maintain throughout

```
//O(|A|) - |A| is the number of points that will be deleted from the data structure
void narrowRange(int min, int max, Boolean axis);
```

- Starting from the head and deleting each link with axis value less than min O(|A1|)
- Starting from the tail and deleting each link with axis value greater than max O(|A2|)
- The deletion itself is O(1) because we only change the linking in the list.
- So the total is O(|A1 + A2|) = O(|A|)

```
//O(1)
Boolean getLargestAxis();
```

- Getting the length of each axis is O(1) because we have head and tail pointers
- Comparing the lengths is O(1) because there is a only two axis

```
//O(n)
Container getMedian(Boolean axis);
```

- Getting the middle element of a linked list is T(n/2) so the method is O(n)
- Creating a container is O(1), so the entire method is O(n)

```
//O(min(n, |B|log|B|)) - B is the number of points in the strip.
//You may use Arrays.sort to implement this function, but you may not change the class
Point for that sake
Point[] nearestPairInStrip(Container container, double width, Boolean axis);
```

- Get an array of all the points inside a strip centered at container's axis value with width **width** (inclusive) sorted by the other axis is O(n), this happens only in the driver function once (not recursive)
- Check for edge cases (2 or less points) O(1) because it is a constant time
- Split the array of point into right half and left half in O(|B|), and for each one find the closest pair of points in it in 2*T(|B|/2)
- Calculate the distance of the minimum pair is O(1)
- Find all the points in the strip centered at container with width delta is O(|B|)
- For each point check the following 7 points in the array is O(|B|)
- Return the closest pair is O(1)
- Using the master's theorem T(n) = a*T(n/b) + f(n) with a=2 b=2, f(n) = O(|B) we get that the function have O(|B|log(|B|)) complexity

```
//O(min(n, |B|log|B|))
Point[] nearestPair();
```

- Determine which axis is larger is O(1)
- Getting the smallest pair with nearestPairInStrip is O(|B|log|B|) complexity
- So the entire complexity is O(|B|log|B|)
- Note: I'm aware that the intended way for us to go about this problem is to do 4.1.9 and 4.1.10 separately (thus the task to describe the complexity of 4.1.10) but conceptually to me merging them together is more logical

6.3 Pseudo-code for the function split (Question 4.2)

BiDirectionalLinkedList [] Split (int value, Boolean axis)

- The function will return an array of two BiDirectionalLinkedList
- Pseudo code for the function:

1. curr_axis ← the list of **axis** axis
2. lower ← curr_axis.head
3. upper ← curr_axis.tail
4. while lower.axisValue < value OR upper.axisValue > value
    a. delete the twin of lower and upper
    b. lower = lower.next
    c. upper = upper.prev
5. array ← new array of BiDirectionalLinkedList with size 2
6. if lower.axisValue > value

a. temp ← lower.getPrev
b. temp.setNext(null)
c. lower.setPrev(null)
d. array[0] ← temp
e. array[1] ← lower

7. else (if upper.axisValue < value)
   a. temp ← upper.getNext
   b. temp.setPrev(null)
   c. upper.setNext(null)
   d. array[0] ← temp
   e. array[1] ← lower
8. return array

- In the worst case scenario we will go continue the loop until we've reached the median, and then the runtime will be T(|c|) = 2*|c|
- So the complexity of the method is O(|c|)

6.4 Pseudo-code for the function split

Done above in section 6.2

6.5 Bonus

Describing with words how to build from an existing DataStructure instance, two DataStructure instance, one containing all the points with X value larger than the median (included) and one containing all the points with X value smaller than the median. In complexity O(n)

Solution:

1. Create a new empty DataStructure instance
2. Go to the linked list of the points sorted by by X of the existing DS and start from the head of the list and until the list is over or the current point's X value equals the **X**, perform the following:
   2.1. Create 2 new Links and copy all the data from the current Link and its twin, set that newly created Links as the twins of each other
   2.2 add the 2 newly created links to the new DS as twins into their respected place (in the LinkedList)
   2.3 delete the current Link and its twin from this DS
3. At this point the remaining links in the DS are the ones with X value larger than **X**
4. return the newly created DS and this current DS

This function's complexity is O(n) because the worst case is copying the entire DS into a new one, which is only proportional to the number of links in the DS

7. GUI -

I’ve submitted the 2 sets of GU (txt and image files are uploaded to the VPL):