6 Theoretical questions

6.1+6.2

The data structure is two sorted LinkedLists and an int value "number of planes":

planesX- A sorted Linked List of Points, this field sorts the points (planes) by their X axis value.

planesY- A sorted Linked List of Points, this field sorts the points (planes) by their Y axis value.

NumOfPlanes- represent the current amount of Points(planes) int the data structure.

- Each list is a two way LinkedList.
- Each Link has 4 four fields, data-the point, prev, next, twin- a pointer to the corresponding Link in the second list.

Methods

addPoint:

the function receives a point, creates two Links with the point as data. $\Theta(2)$

Set each link as the others twin, $\theta(2)$.

Adding the links to the both lists, using the add function of Sorted LinkList $\theta(2n)$.

Updates the value of "NumOfPlanes"$\theta(1)$.

Overall: $\theta(n)$

getPointInRangeRegAxis:

the function creates an array $\theta(1)$.

We check each point in the List sorted by the given axis, and add to the array each point with the axis value between min and max given value $\theta(n)$.

Overall $\theta(n)$.

getPointInRangeOppAxis:

the function creates an array $\theta(1)$.

We check each point in the List sorted by the opposite axis, and add to the array each point with the given axis value between min and max $\theta(n)$.

Overall $\theta(n)$.

getDensity:

The function uses the equation for density using four basic arithmetic operations $\theta(\alpha)$.

When α is a constant.

Overall $\theta(1)$.

<u>**narrowRange:**</u>

The function start from the first link of the list sorted by the given axis. The function then uses removeFirst method of LinkedList on each link up until the first link with a larger or equal to the min value. removeFirst of LinkedList $\theta(1)$. Because the list is sorted by the axis we only visit the points that are removed. Total $\theta(\mathrm{a1})$, $\mathrm{a1}=$ number of points with axis value smaller than min.

After that, the function start from the last link of the list sorted by the given axis. The function then uses removeLast method of LinkedList on each link up until the first link with a smaller or equal to the max value. removeLast of LinkedList $\theta(1)$. Because the list is sorted by the axis we only visit the points that are removed. Total $\theta(\mathrm{a2})$, $\mathrm{a2}=$ number of points with axis value larger than max.

Each time we remove a link from one list we use the twin pointer to remove it from the second list. By that we can remove from the second list without searching the second list.

This time complexity is possible due to the LinkedLists of the data structure being a two way LinkedList.

Overall $\theta(\mathrm{a2}+\mathrm{a1}=$all points to be removed).

<u>**getLargestAxis**</u>:

each list is sorted by different axis where the first is the minimal value and the last is the maximal. the function uses a constant number of arithmetic operations on max and min which can be used in a constant number of commend because planesX and planesY are two way LinkedList.

Overall $\theta(1)$.

<u>**getMedian**</u>:

The function iterate over the list sorted by the given axis until "number of planes"/2 $\theta(n/2)$.

It then return the point found inside a container.

Overall $\theta(n)$.

<u>**nearestPairInStrip**</u>:

the function create a constant number of variables and a comparator of planes by the opposite axis given.

Create an array by calling the getPointsInRangeOppAxis $\theta(n)$.

By using two loops we compare every two Points with the possibility of distance smaller than the minimum distance, while saving the two points with the minimum distance.

As a result of the assumption that every two point in the strip has a minimum of width/2 distance apart, we can then see that for every point there are a maximum of 7 points meaning that the two loops has a maximum of 7n.

We use two auxiliary function that calculate distance and compare two points. Total of $\theta(1)$ each.

Overall $\theta(n)$.

nearestPair:

The function splits the array in two each time, and stop when there are less then three points in the array. It then uses the nearestPairInStrip function to find the closest pair in both sides.

Overall $\theta(n\log n)$.

6.3

Split(int value, boolean axis)

```
start ← planesOf(axis).First()
end ← planesOf(axis).Last()
while(start.data.value(axis)<value and end.data.value(axis)>value)
    start←start.Next
    end←end.Prev
new DS output1
new DS output2
if(start.data.value(axis)≥ value and end.data.value(axis)≤ value)
    output1←null
    output2← this
else if(start.data.value(axis)≥ value)
    while(planesOf(axis).first.data.value(axis)<value)
        output1.addPoint(planesOf(axis).removeFirst.data)
        output2= this.narrowRange(value, planesOf(axis).last.data.value(axis),axis)
else
    while(planesOf(axis).last.data.value(axis)>value)
        output1.addPoint(planesOf(axis).removeLast.data)
        output2= this.narrowRange(planesOf(axis).first.data.value(axis),value ,axis)
return (output1,output2)
```

*|C| is the number of points in the smaller collection

Line number	**times**	**cost θ**
1	1	Θ (1)
2	1	Θ(1)
3	2*C	Θ(c)
4	1	Θ(c)
5	1	Θ(c)
6	1	Θ(1)
7	1	Θ(1)
8	3	Θ(1)
9	1	Θ(1)
10	1	Θ(1)
11	2	Θ(1)
12	C	Θ(c)
13	C	Θ(c)
14	C	Θ(c)
15	1	Θ(1)
16	C	Θ(c)
17	C	Θ(c)
18	C	Θ(c)
19	1	Θ(1)
total		Θ(c)

<u>6.4 Time complexity of nearestPair()</u>

We divide the function into section for an easier orientation over the code.

<u>First section- stop condition:</u>

There are three stop conditions.

number of planes <2 - θ (2)

number of planes = 2- θ (4)

number of planes =3 - θ(5) + θ(bruteForce)

<u>BruteForce time complexity-</u> the function receives an array of point, compare 3 points exactly, return the two points with the smallest distance apart. Since the function compares exactly 3 points disregarding the length of the array, the function time complexity is θ (constant).

Overall θ (11 + constant) ⟹θ (1)

<u>Second section- instantiating variables:</u>

Med variable uses getMedian function which is θ (n).

Instantiating the dtLarger and dtSmaller variables using a copy constructor is a total of θ (2*n)

All other are θ (1)

Overall θ (n)

<u>Third section – splitting:</u>

Using the narrowRange function on dtLarger and dtSmaller, a total of θ (2*A)

Where A is the number of points deleted.

Fourth section- recursive calls:

Calling twice to this function, recursively, with data structures each with n/2 points.

Overall, 2*T(n/2).

<u>Fifth section – min distance from both sides:</u>

Comparing a constant number of points, 4 or less, to find the minimum distance from both.

In total it is a constant number of operations with time complexity θ (1).

Overall θ (1).

<u>Sixth section – nearestPairInStrip for middle part:</u>

Using the nearestPairInStrip with time complexity of θ (n). all other lines compare two pairs of points like the fifth section θ (1).

Overall θ (n).

Total : $T(n)=2*T(n/2)+\theta(n)+\theta(1)+\theta(2*A)+\theta(1)+\theta(n)= 2*T(n/2)+\theta(n)\Leftrightarrow T(n)=2*T(n/2)+\mathcal{F}(n)$

we will use the master theorem:

2= a , b=2.

$n^{\log_b(a)}=n^{\log_2(2)}=n=\mathcal{F}(n)$ which is case 2 of the master theorem,

therefore $T(n)=\theta(n*\log(n))$.

__**6.5**__

We implemented the exact same thing in our nearestPair function.