**Data Structures**

**Spring 2020**

**Lab #4\_Problem 2**

**Date: 2020. 05. 21**

**ID/Name: 201533661 이승수**

**[Lab4\_Problem2(pseudocode)] (30 points)**

**Write an algorithm in pseudocode form for each of the following functions: (include the function header)**

Here are some functions and variables used at pseudocodes at below.

**[My Pseudocode]**

Struct Point

int x; int y;

Point Points; //array of Point

Point findMedian(struct Point \*A, int dimValue)

return &A[median by dimValue]

Boolean ifLeafNode(int index)

if Points[index] is leaf node, return true

else, return false

Float distance(Point A, Point B)

return sqrt( pow((A->x) – (B->x)) + pow((A->y) –(B->y)) )

**(1) point\_search for a user-specified point**

**Point\_search(findMedian(Points), Point Target): //initial Root is findeMedian(Points), Target is point to search**

searchChild(Point \*Root, Point Target, int dimValue, Point \*leftChild, Point \*rightChild)

struct Point Root, leftChild, rightChild=&Root, &leftChild, &rightChild

if dimValue==0: //set median by x

if Root==Target: //if Target node is searched at Root node, return Root

return Root

else if !ifLeafNode(Root) and (Root->x) < (Target->x): //if Target node isn’t searched and Target’s x>Root’s x, goto rightChild of Root

dimValue=(dimValue+1)%2

return searchChild(rightChild, Target, dimValue, rightChild->leftChild,

rightChild->rightChild)

else if !ifLeafNode(Root) and (Root->x) > (Target->x): //if Target node isn’t searched and Target’s x<Root’s x, goto leftChild of Root

dimValue=(dimValue+1)%2

return searchChild(leftChild, Target, dimValue, leftChild->leftChild,

rightChild->rightChild)

if dimValue==1: //set median by y

if Root==Target: //if Target node is searched at Root node, return Root

return Root

else if !ifLeafNode(Root) and (Root->x) < (Target->x): //if Target node isn’t searched and Target’s y>Root’s y, goto rightChild of Root

dimValue=(dimValue+1)%2

return searchChild(rightChild, Target, dimValue, rightChild->leftChild,

rightChild->rightChild)

else if !ifLeafNode(Root) and (Root->x) > (Target->x): //if Target node isn’t searched and Target’s y<Root’s y, goto leftChild of Root

dimValue=(dimValue+1)%2

return searchChild(leftChild, Target, dimValue, leftChild->leftChild,

rightChild->rightChild)

return Null; //if Target node cannot be found until leaf node, return Null

**(2) range\_search(find all points contained within a specified bounding rectangle)**

**range\_search(Point rangeLT, Point rangeRB): //rangeLT(x,y value of left-top point of range), ranger(x,y value of right-bottom point of range)**

Point[] pointsInRange=[]; //array to store all points in specified bounding rectangle

for i in range(Points):

Boolean X,Y=False

if (rangeRB->x <= Points[i]->x) and (Points[i]->x <= rangeLT->x):

X=True

If (rangeRB->y <= Points[i]->y) and (Points[i]->y <= rangeLT->y):

Y=True

If X==True and Y==True:

pointsInRange.append(Points[i])

return pointsInRange;

**(3) nearest\_neighbor\_search(given a point, find one or more nearest neighbor points)**

**[Result]**

**Nearest\_neighbor\_search(Point Target) //search for one or more nearest neighbors in Points from Target**

Float minDist=0

Int neighborNum=0

Point[] nearestNeighber=[]

For i in range(Points):

If Target!=Points[i] and minDist>distance(Points[i],Target):

minDist= distance(Points[i],Points[i+1])

neighborNum=0; nearestNeighbor=[]

nearestNeighbor[neighborNum]=Points[i]

else if Target!=Points[i] and minDist==distance(Points[i],Target):

neighborNum++

nearestNeighbor[neighborNum]=Points[i]

return nearestNeighbor;