Colton Parham, CDP210001

CS3345.505

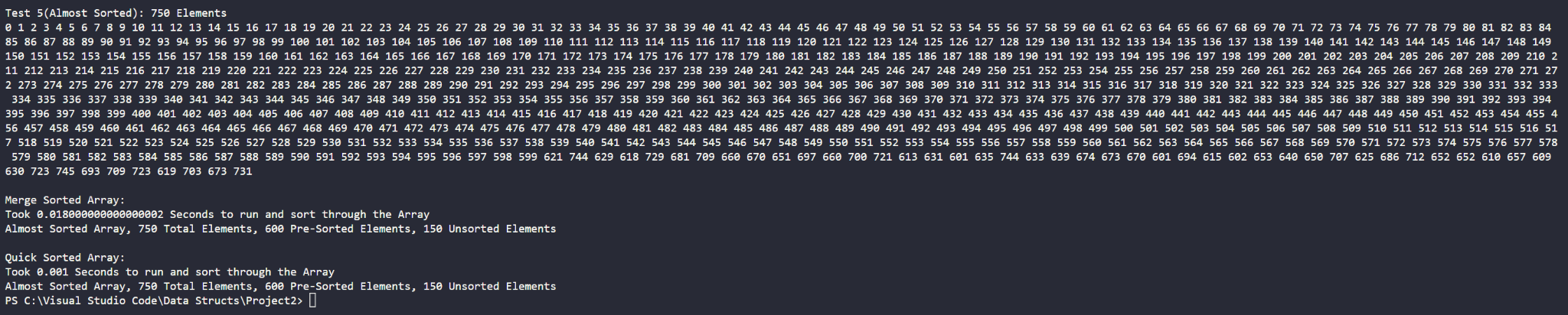
Dr. Zhao

\*Note that the full image files of the outputs, and program files will be inside of the Zip File.

Conclusion:

In conclusion, it was found that the Median of 3, with a Cutoff of 15 QuickSort Algorithm is faster than the Recursive MergeSort. Between the two, they were handed the exact same data, with the QuickSort remaining around twice as fast in most cases compared to the QuickSort. In the almost sorted test, it was shown that the QuickSort was exponentially faster, by more than a factor of 10. Both of the algorithms used the same interface, with the comparable to keep the margin of difference only at the algorithm design level, and no dependencies between data types.





BONUS:

import point library/package

public class coLinear

{

public main

{

int nV // n value

int xArr[] = [0,1,5, 19, 6, 20, 3, -5, -11, -3]

int yArr[] = [0, -1, 10,5, -19, 3, 5, 11, -4]

nV =xArr.length

Point []points = new Point[nV]

Point []CoLinearPoints = new Point[100]

double slope[] = new double[n] // slope of the colinear points

for (int i = 0, i < CoLinearPoints.length, i++)

{

CoLinearPoints[i] = new Point(-9999,-9999) // setting the point coords

}

print("Points")

for (int i = 0, i < nV, i++)

{

points[i] = new Point(xArr[i],yArr[i])

print(getPoint([points[i]) + ", ")

}

// each point, will repeat the given N amount of times

for (int i = 0, i < nV, i++)

{

Point []TempPoint = points

int k = 1

Point tempV = new Point()

tempV = TempPoint[0]

TempPoint[0] = TempPoint[i]

TempPoint[0] = tempV

slope[0] = 0

for (int j = 1, j < nV, j++)

{

if (TempPoint[0].getX() - TempPoint[j].getX() != 0) then (slope[k] = (double)(TempPoint[0].getY()-TempPoint[j].getY())/ (TempPoint[0].getX()-TempPoint[j].getX()))

k++

}

// sorting by the slopes (N^2)

for (int q = 1, q < nV, q++)

{

for (int m = 1 + 1, m < nV, m++)

{

Point temp1 = new Point(-9999,-1111)

double TempSlope

// this will swap slopes

if (slope[q] <= slope[m]) then (temp1 = TempPoint[1], TempPoint[q] - TempPoint[m], TempPoint[m] = temp1, TempSlope = slope[1], slope[q] = slope[m], slope[m] = TempSlope )

}

}

// signifier that not colinear

int notColinear = 0

ColinearPoints[notColinear] = TempPoint[0]

notColinear++

for (int t = 1, t < nV, t++)

{

if (slope[t] == slope[t + 1]) then (if (notReal(TempPoint[t],ColinearPoints)) then (ColinearPoints[notColinear] = TempPoint[t], t++)), if(notReal(TempPoint[t+1, ColinearPoints])) then (ColinearPoints[notColinear] = TempPoint[t+1])

notColinear++

else

{

if (notColinear >= 4) then (Print("Colinear Points: "), for (int j = 0, j < notColinear, j++)){Print(getPoints(ColinearPoints[j]))}

}

// getting rid of the prior colinear points

for (int j = 1, j < notColinear, j++)

{

ColinearPoints[j] = new Point(-9999,-9999)

}

notColinear = 1

}

}

}

public stat string getPoints(Point poin)

{

String point = " "

points = points + "("poin.getXArr() + ", " + poin.getYArr + ") "

return points

// this will return the points values

}

// true false function

public stat bool notReal (Point Poin, Point[] pts)

{

for (int i = 0, i < points.length, i++)

{

if (poin.getXArr() == pts[i].getXArr and poin.getYArr() == pts[i].getYArr) then (false)

else (true)

}

}

}