6 - Divide and Conquer

1. Innerhalb eines zufällig mit Ganzzahlen gefüllten Arrays soll das minimale und das maximale Element gefunden werden. Schreibe einen Algorithmus *minMax*, welcher auf dem Divide and Conquer Ansatz basiert. Am Ende soll die größte und die kleinste Zahl zurückgegeben werden. Achte darauf, dass die Anzahl der Vergleiche möglichst gering ist. Als Orientierung kannst du die fiktive Zahl $2(n-1)$ nehmen. Versuche unter dieser Anzahl an Vergleichen zu sein. Zur Kontrolle nutze eine Variable *call*, welche die Anzahl der Vergleiche speichert. (Für die Größe das Arrays kannst du $2^k, k \in N$ annehmen. In diesem Fall wird die 0 zu den natürlichen Zahlen hinzu gezählt.)

2. Schreibe einen Algorithmus, welcher aus einer Menge von Punkten die beiden Punkte findet, welche sich am Nächsten sind. Der Grundansatz soll natürlich wieder *Divide and Conquer* sein. Zusätzlich kannst du die Matplotlib in Python verwenden um das Ergebnis zu visualisieren. (Das ist aber nicht erforderlich!)

Hilfestellung zu Zweitens

- Sortiere die Punkte nach X-Achse
- Ziehe eine Mittellinie L in der Mitte zwischen den mittleren Punkten der sortierten Liste. (Bei einer ungeraden Anzahl an Punkten muss entschieden werden, auf welcher Seite mehr Punkte sind.)
- Teile die Punktmenge an der Linie L in zwei Hälften Q und R
- Die Mittelline L musst du dir merken!
- Führe dies rekursiv fort, bis nur noch Hälften zwei oder drei Punkten bestehen.
- Die Punkte für die kürzeste Distanz bei zwei Punkten ist trivial. Bei drei Punkten muss man sich schon was einfallen lassen. Die beiden Punkte, welche das nächste Paar bilden, müssen zurückgegeben werden.
- Dann musst du die Hälften noch *mergen*. Dabei ist zu beachten, dass natürlich neue nächste Paare entstehen könnten. Zuerst muss die kürzere Distanz als δ gespeichert werden.
- Beim *mergen* können nur die Punkte eine kürzere Distanz haben, welche maximal die Distanz δ von der Mittellinie L entfernt sind. Alle Punkte, für die das zutrifft, werden in eine temporäre Liste gespeichert und nach Y-Achse sortiert.

nearestPair(P)

1. MergeSort(P)
2. **return** nearestPairRec(P)

nearestPairRec(P)

1. **if** P.length() == 2 **return** P[0], P[1]
2. **if** P.length() == 3 **return** nearestPairOutOfThreePairs((P[0],P[1]),(P[0],P[2]),(P[1],P[2]))
3. divide points to Q and R
4. calculate line L
5. **return** mergePoints(Q, R, nearestPairRec(Q), nearestPairRec(R), L)

mergePoints(Q, R, leftPairNearest, rightPairNearest, L)

1. P' = []
2. δ = minDistance(leftPairNearest, rightPairNearest)
3. **for** i to Q.length() **do**
   if calcDistance(Q[i], (L, Q[i][1])) $< \delta$ **then** P'.append(Q[i])

4. **for** i to R.length() **do**
   if calcDistance(R[i], (L, R[i][1])) $< \delta$ **then** P'.append(Q[i])

5. **if** P'.length() == 0 **then**
   if distance of leftPairNearest < distance of rightPairNearest **return** left-PairNearest **else return** rightPairNearest

6. candidatePair = (P'[0],P'[1])

7. test the distance between all pointrs in P'

8. store the best in candidatePair

9. **return** nearestPairOutOfThreePairs(candidatePair, leftPairNearest, right-PairNearest)