Algorithms for the reduction of the number of points required to represent a digitized line or its caricature

- naive algorithm: only consider every n-th point for fixed n
- Algorithm LINEREDUCTION

  INPUT: List of points, minimum distance ε

  start_anchor, end_anchor ← first and last point

  furthest, distance ← point and distance of point furthest away from line defined by anchors

  IF distance < ε:

    RETURN [start_anchor, end_anchor]

  ELSE:

    RETURN concat(LINEREDUCTION(points[:furthest], ε), LINEREDUCTION(points[furthest:], ε))

  ENDIF

- distance measure: { perpendicular distance if start_point ≠ endpoint; maximum distance to point if start_point = end point

- ALGORITHM Method1

  INPUT: list of points, ε

  anchor, fp ← points[0], points[1]

  result = [anchor]

  DO:

    furthest, distance ← find furthest point to line defined by anchor and fp

    IF distance < ε:

      result.append(fp)

      IF fp = points[-1]: BREAK

      ELSE: anchor ← fp, fp ← points[-1]

    ELSE:

      fp = furthest

    ENDIF

  ENDDO

  RETURN result

- ALGORITHM Method2

```
INPUT: list of points, ε
anchor, fp ← points[0], stack([points[-1]])
result ← [anchor]
DO:
  furthest, distance ← find furthest point to line defined by anchor and fp
  IF distance < ε:
    current_fp ← fp.pop()
    result.append(current_fp)
    IF fp.empty: BREAK
    ELSE: anchor ← current_fp
  ELSE: fp.stack(furthest)
ENDDO
return result
```

- ALGORITHM Lang

```
INPUT: list of points, ε
anchor, fp, result ← points[0], points[2], []
FOR 2 < i < len(points)
  FOR p between anchor and fp:
    IF p in tolerance distance of segment defined by anchor and fp
      fp ← points[i]
    ELSE:
      result.append(anchor)
      anchor ← points[index(fp)-1]   | Alternative: move anchor to furthest point from segment
      BREAK
    ENDIF
  ENDFOR
ENDFOR
```

- ALGORITHM Modified-Lang

```
INPUT: list of points, ε
anchor, fp, result ← points[0], points[2], []
distances = []
FOR 1 ≤ i < len(points):
    distance ← distance between points[i] and segment defined by anchor and fp
    distances.append(distance)
    IF sum(distances) < ε:
        fp ← points[i+2]
    ELSE:
        furthest, distance, index ← find furthest point to line defined by anchor and fp
        IF distance ≥ ε:
            distances = distances[index - index(anchor):]
            anchor ← furthest
        ENDIF
    ENDIF
ENDFOR
RETURN result
```