```
(*SortPoints Algorithm_____________
 ___________________________________________*)

FindMinMax[PointList_] :=
  Module[{imin, imax, jmin, jmax, iSplits, jSplits, iDistance, jDistance},
   Modul
   imin = 10000;
   imax = 0;
   jmin = 10000;
   jmax = 0;

   Print["Splits = ", splits];
   gib aus
   For[i = 1, i ≤ Length[PointList], i++,
                 Länge
    If[imin ≥ PointList[[i, 2]], imin = PointList[[i, 2]]];
    wenn
    If[jmin ≥ PointList[[i, 1]], jmin = PointList[[i, 1]]];
    wenn
    If[imax ≤  PointList[[i, 2]], imax = PointList[[i, 2]]];
    wenn
    If[jmax ≤  PointList[[i, 1]], jmax = PointList[[i, 1]]];
    wenn
   ];

   Print["imin = ", imin, " imax = ", imax , " jmin = ", jmin, " jmax = ", jmax];
   gib aus

   jSplits = ConstantArray[0, {1, splits + 1}];
             konstantes Array
   jSplits[[1, 1]] = jmin;

   jDistance = jmax - jmin;

   For[i = 2, i ≤ splits + 1, i++,
   For-Schleife
    jSplits[[1, i]] = jSplits[[1, i - 1]] + (jDistance / splits)
   ];

   Print["jSplits = ", jSplits];
   gib aus
   iSplits = ConstantArray[0, {1, splits + 1}];
             konstantes Array
   iSplits[[1, 1]] = imin;

   iDistance = imax - imin;

   For[i = 2, i ≤ splits + 1, i++,
   For-Schleife
    iSplits[[1, i]] = iSplits[[1, i - 1]] + (iDistance / splits)
   ];

   Print["iSplits = ", MatrixForm[iSplits]];
   gib aus                  Matritzenform
```

```
    GoThrougConvexHulls[iSplits, jSplits, PointList];
   ];

GoThrougConvexHulls[iSplits_, jSplits_, PointList_] := Module[{ConvexHull},
                                                        Modul
    (*ConvexHull = ConstantArray[0,{splits,1}];*)
                   konstantes Array
    ConvexHull = {};

    For[ii = 1, ii ≤ Length[Flatten[iSplits]] - 1, ii++,
    For-Schleife          Länge   ebne ein
     AppendTo[ConvexHull, FindPointsInConvexHull[
     hänge an bei
        iSplits[[1, ii]], iSplits[[1, ii + 1]], jSplits, PointList]];
    ];
    (*Print["ConvexHull =",MatrixForm[Flatten[ConvexHull]]];*)
      gib aus                 Matritzenform  ebne ein
    ConvexHull = Flatten[ConvexHull];
                 ebne ein

    FindStartVectors[ConvexHull];

   ];


FindPointsInConvexHull[iSplitsBottom_, iSplitesTop_, jSplits_, PointList_] :=
  Module[{ConvexHullCell, ConvexHullCellList, ConvexHullCellKeys},
  Modul
    ConvexHullCell = {};
    ConvexHullCellKeys = <||>;
    For[b = 1, b ≤ Length[Flatten[jSplits]] - 1, b++,
                  Länge   ebne ein
     For[i = 1, i ≤ Length[PointList], i++,
     For-Schleife        Länge
       If[PointList[[i, 2]] ≥ iSplitsBottom &&
       wenn
           PointList[[i, 2]] ≤ iSplitesTop && PointList[[i, 1]] ≥ jSplits[[1, b]] &&
           PointList[[i, 1]] ≤ jSplits[[1, b + 1]],

         AssociateTo[ConvexHullCellKeys, CoordJ → PointList[[i, 1]]];
         assoziiere mit
         AssociateTo[ConvexHullCellKeys, CoordI → PointList[[i, 2]]];
         assoziiere mit
         AssociateTo[ConvexHullCellKeys, CellJ → b];
         assoziiere mit
         AssociateTo[ConvexHullCellKeys, CellI → ii];
         assoziiere mit
         AppendTo[ConvexHullCell, ConvexHullCellKeys];
         hänge an bei
        ];
      ];
    ];
    Return[ConvexHullCell];
    gib zurück
```

```
  gib zurück
 ];


FindStartVectors[ConvexHull_] := Module[{StartPointCloud = {},
                                  Modul
   StartPointCloudKeys = <||>, VecI, VecJ, counti, countj, Start, nextI, nextJ},

  For[rr = 0, rr ≤ Length[ConvexHull] - 1, rr++,
  For-Schleife          Länge
   If[ConvexHull[[rr]][CellI] == 1 && ConvexHull[[rr]][CellJ] ≤ splits,
   wenn
    AssociateTo[StartPointCloudKeys,
    assoziiere mit
     {CoordJ -> ConvexHull[[rr]][CoordJ], CoordI -> ConvexHull[[rr]][CoordI],
      CellJ -> ConvexHull[[rr]][CellJ], CellI -> ConvexHull[[rr]][CellI]}];
    AppendTo[StartPointCloud, StartPointCloudKeys]];
    hänge an bei

   If[ConvexHull[[rr]][CellI] ≤ splits && ConvexHull[[rr]][CellJ] == 1 ,
   wenn
    AssociateTo[StartPointCloudKeys,
    assoziiere mit
     {CoordJ -> ConvexHull[[rr]][CoordJ], CoordI -> ConvexHull[[rr]][CoordI],
      CellJ -> ConvexHull[[rr]][CellJ], CellI -> ConvexHull[[rr]][CellI]}];

    If[MemberQ[StartPointCloud, StartPointCloudKeys] == False,
    ··· enthalten?                                    falsch
     AppendTo[StartPointCloud, StartPointCloudKeys]];
     hänge an bei
   ];
  ];

  counti = 100;
  countj = 100;
  VecI = 0;
  VecJ = 0;

  For[aa = 0, aa ≤ Length[StartPointCloud], aa++,
  For-Schleife          Länge
   If[StartPointCloud[[aa]][CoordI] ≤ counti &&
   wenn
     StartPointCloud[[aa]][CellJ] ≤ splits && StartPointCloud[[aa]][CellI] == 1,

    counti = StartPointCloud[[aa]][CoordI];
    VecI = StartPointCloud[[aa]];
   ];


   If[StartPointCloud[[aa]][CoordJ] ≤ countj &&
   wenn
     StartPointCloud[[aa]][CellI] ≤ splits && StartPointCloud[[aa]][CellJ] == 1,

    countj = StartPointCloud[[aa]][CoordJ];
    VecJ = StartPointCloud[[aa]];
```

```
 ];
];

Print["VecI = ", VecI];
gib aus
Print["VecJ = ", VecJ];
gib aus

For[yy = 0, yy ≤ Length[StartPointCloud], yy++,
For–Schleife        Länge
 If[ StartPointCloud[[yy]][CellI] == 1 && StartPointCloud[[yy]][CellJ] ≤ splits &&
 wenn
    StartPointCloud[[yy]][CoordJ] ≤ VecI[CoordJ] &&
    StartPointCloud[[yy]][CoordI] ≤ VecI[CoordI] + 0.8,
   (*statt plus 0.8 vllt einen Offset aus distanzen bestimmen...
                                  Versatz
    nur wie wenn andere Nachbarn noch unbekannt?*)

   VecI =  StartPointCloud[[yy]];
  ];
];

For[yy = 0, yy ≤ Length[StartPointCloud], yy++,
For–Schleife        Länge
 If[ StartPointCloud[[yy]][CellJ] == 1 && StartPointCloud[[yy]][CellI] ≤ splits &&
 wenn
    StartPointCloud[[yy]][CoordI] ≤ VecJ[CoordI] &&
    StartPointCloud[[yy]][CoordJ] ≤ VecJ[CoordJ] + 0.5,

   VecJ =  StartPointCloud[[yy]];
  ];
];

Print["VecI = ", VecI, " VecJ =", VecJ];
gib aus

If[VecI == VecJ, Start = VecI, Start = VecJ];
wenn
Print["Start = ", Start];
gib aus
(*Find NextI and NextJ *)
  finde
nextI = <|CoordJ → 100 000, CoordI → 100 000|>;
nextJ = <|CoordJ → 100 000, CoordI → 100 000|>;

Print["StartPointCloud = ", StartPointCloud];
gib aus
For[bb = 0, bb ≤ Length[StartPointCloud], bb++,
For–Schleife        Länge

 If[StartPointCloud[[bb]][CellI] == Start[CellI] ||
 wenn
   StartPointCloud[[bb]][CellI] == Start[CellI] + 1 ||
   StartPointCloud[[bb]][CellI] == Start[CellI] - 1 ,
```

```
 If[StartPointCloud[[bb]][CoordI] ≥ Start[CoordI] && StartPointCloud[[bb]][
 wenn
       CoordJ] ≤ nextI[CoordJ] && StartPointCloud[[bb]] ≠ Start,

    nextI = StartPointCloud[[bb]];
    Print["nextI first = ", nextI];
    gib aus
   ];
 ];

 If[StartPointCloud[[bb]][CellJ] == Start[CellJ] ||
 wenn
   StartPointCloud[[bb]][CellJ] == Start[CellJ] + 1 ||
   StartPointCloud[[bb]][CellJ] == Start[CellJ] - 1,

  If[StartPointCloud[[bb]][CoordJ] ≥ Start[CoordJ] && StartPointCloud[[bb]][
  wenn
       CoordI] ≤  nextJ[CoordI] && StartPointCloud[[bb]] ≠ Start,

    nextJ = StartPointCloud[[bb]];
    Print["nextJ first = ", nextJ];
    gib aus
   ];
 ];
];


(*CHeck if NextI and NectJ are final Values*)
                                    Werte

For[bb = 0, bb ≤ Length[StartPointCloud], bb++,
For–Schleife          Länge

 If[Abs[nextJ[CoordI] - Start[CoordI]] ≥
 ⋯ Absolutwert
    Abs[StartPointCloud[[bb]][CoordI] - Start[CoordI]] &&
    Absolutwert
   StartPointCloud[[bb]] ≠ Start &&  Abs[nextJ[CoordJ] - Start[CoordJ]] >
                                     Absolutwert
    Abs[StartPointCloud[[bb]][CoordJ] - Start[CoordJ]],
    Absolutwert
  nextJ = StartPointCloud[[bb]];
 ];

 If[Abs[nextI[CoordJ] - Start[CoordJ]] ≥
 ⋯ Absolutwert
    Abs[StartPointCloud[[bb]][CoordJ] - Start[CoordJ]] &&
    Absolutwert
   StartPointCloud[[bb]] ≠ Start &&  Abs[nextI[CoordI] - Start[CoordI]] >
                                     Absolutwert
    Abs[StartPointCloud[[bb]][CoordI] - Start[CoordI]],
    Absolutwert
  nextI = StartPointCloud[[bb]];
 ];
];
```

```
    Print["nexti = ", nextI];
    gib aus
    Print["nextj = ", nextJ];
    gib aus
    CreatePossiblePointsListsIAndJ[nextI, nextJ, Start, ConvexHull];
   ];

CreatePossiblePointsListsIAndJ[nextI_, nextJ_, Start_, ConvexHull_] :=
  Module[{IList = {}, JList = {}, IDir, JDir, distance, cache, PotNextI, PotNextJ},
   Modul
    aj = 2;
    restJ = splits;

    For[rr = 0, rr ≤ Length[ConvexHull], rr++,
    For-Schleife          Länge

     If[ConvexHull[[rr]][CellJ] ≤ aj && ConvexHull[[rr]][CellI] ≤ splits,
     wenn
      AppendTo[IList, ConvexHull[[rr]]];
      hänge an bei
       ];

     If[ConvexHull[[rr]][CellI] ≤ aj && ConvexHull[[rr]][CellJ] ≤ splits,
     wenn
      AppendTo[JList, ConvexHull[[rr]]];
      hänge an bei
     ];

    ];
    Print["IList = ", IList];
    gib aus
    Print["JList = ", JList];
    gib aus
    FindNeighbours[IList, JList, Start, nextI, nextJ, ConvexHull];
   ];
```