Algorithm-05

—— Closest pair of points problem

A. Problem Description

The closest pair of points problem or closest pair problem is a problem of computational geometry: given n points in metric space, find a pair of points with the smallest distance between them.

B. Description of algorithm

Following are the detailed steps of a O(n (Logn)^2) algortihm.

Input: An array of n points *P[]*

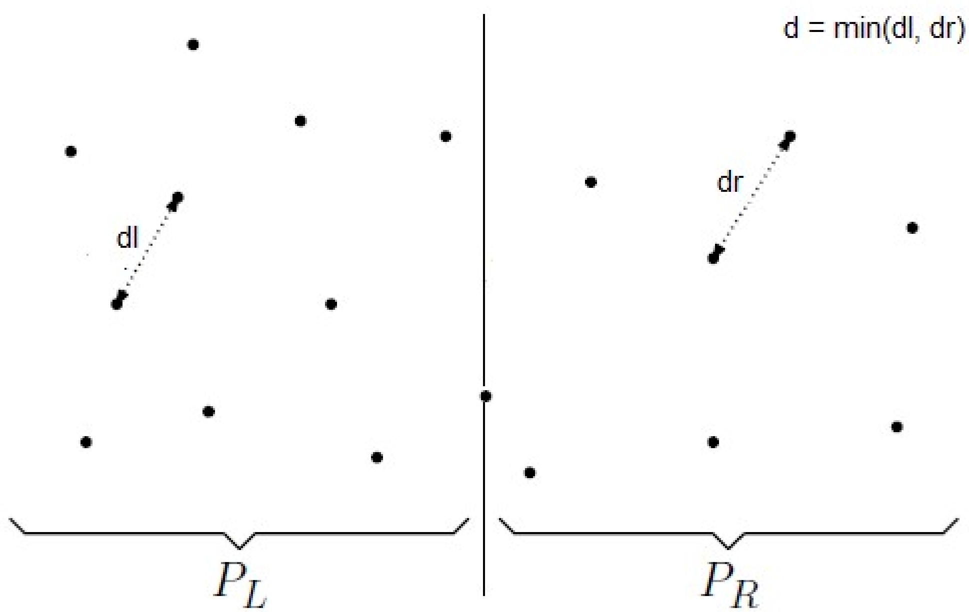
Output: The smallest distance between two points in the given array.

As a pre-processing step, input array is sorted according to x coordinates.

1) Find the middle point in the sorted array, we can take *P[n/2]* as middle point.

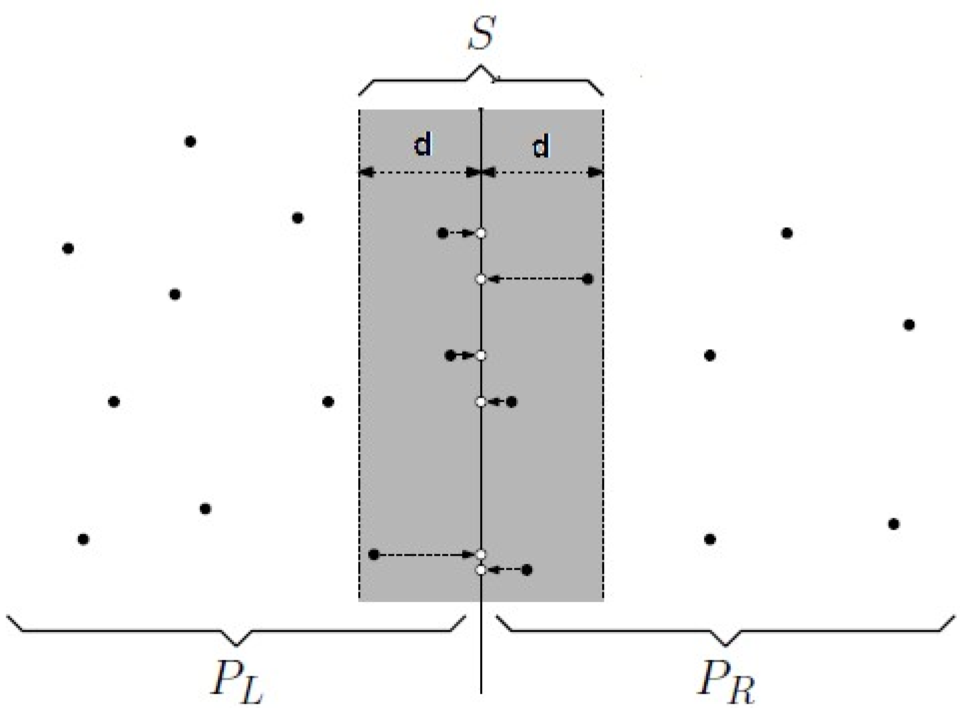
2) Divide the given array in two halves. The first subarray contains points from P[0] to P[n/2]. The second subarray contains points from P[n/2+1] to P[n−1].

3) Recursively find the smallest distances in both subarrays. Let the distances be dl and dr. Find the minimum of dl and dr. Let the minimum be d.



4) From above 3 steps, we have an upper bound d of minimum distance. Now we need to consider the pairs such that one point in pair is from left half and other is from

right half. Consider the vertical line passing through passing through P[n/2] and find all points whose x coordinate is closer than d to the middle vertical line. Build an array strip[] of all such points.



5) Sort the array strip[] according to y coordinates. This step is O(nLogn). It can be optimized to O(n) by recursively sorting and merging.

6) Find the smallest distance in strip[]. This is tricky. From first look, it seems to be a $O(n^2)$ step, but it is actually $O(n)$. It can be proved geometrically that for every point in strip, we only need to check at most 7 points after it (note that strip is sorted according to Y coordinate). See this for proof.

7) Finally return the minimum of d and distance calculated in above step (step 6)

C. *Time Complexity*

Let Time complexity of above algorithm be $T(n)$. Let us assume that we use a $O(nLogn)$ sorting algorithm. The above algorithm divides all points in two sets and recursively calls for two sets. After dividing, it finds the strip in $O(n)$ time, sorts the strip in $O(nLogn)$ time and finally finds the closest points in strip in $O(n)$ time. So $T(n)$ can expressed as follows

$T(n) = 2T(n/2) + O(n) + O(nLogn) + O(n)$

$T(n) = 2T(n/2) + O(nLogn)$

$T(n) = T(n \times Logn \times Logn)$

D. *Code[Python]*

```
#!/usr/bin/python
# Filename: CPair.py

import math

class PointX:
  def __init__(self, ID = -1, x = -1, y = -1):
    self.ID = ID
    self.x = x
    self.y = y
  '''
  def setValue(self, ID, x, y):
    this.ID = ID
    this.x = x
    this.y = y
    '''

class PointY:
  def __init__(self, index = -1, x = -1, y = -1):
    self.index = index
    self.x = x
    self.y = y
  '''
  def setValue(self, index, x, y):
    this.index = index
    this.x = x
    this.y = y
    '''

def MergeSortX(a, n):
  b = []
  for i in range(0, n):
    b.append(PointX())
  s = 1
  while s < n:
    MergePass(a, b, s, n)
    s += s
    MergePass(b, a, s, n)
```

```
    s += s

def MergeSortY(a, n):
  b = []
  for i in range(0, n):
    b.append(PointY())
  s = 1
  while s < n:
    MergePass(a, b, s, n)
    s += s
    MergePass(b, a, s, n)
    s += s

def MergePass(x, y, s, n):
  i = 0
  while i <= n - 2 * s:
    Merge(x, y, i, i + s - 1, i + 2 * s - 1)
    i += 2 * s
  if i + s < n:
    Merge(x, y, i, i + s - 1, n - 1)
  else:
    for j in range(i, n):
      y[j] = x[j]

'''
def MergePassY(x, y, s, n):
  i = 0
  while i <= n - 2 * s:
    MergeY(x, y, i, i + s - 1, i + 2 * s - 1)
    i += 2 * s
  if i + s < n:
    MergeY(x, y, i, i + s - 1, n - 1)
  else:
    for j in range(i, n):
      y[j] = x[j]
      '''

def Merge(c, d, l, m, r):
  i = l
  j = m + 1
  k = l
```

```
  while i <= m and j <= r:
    if c[i] <= c[j]:
      d[k] = c[i]
      k += 1
      i += 1
    else:
      d[k] = c[j]
      k += 1
      j += 1
  if i > m:
    for q in range(j, r + 1):
      d[k] = c[q]
      k += 1
      q += 1
  else:
    for q in range(i, m + 1):
      d[k] = c[q]
      k += 1
      q += 1

def Distance(p, q):
  dx = math.fabs(p.x - q.x)
  dy = math.fabs(p.y - q.y)
  return math.sqrt(dx ** 2 + dy ** 2)

def CPair(X, n, List):
  # list = [a, b, d]
  if n < 2:
    return False
  MergeSortX(X, n)
  Y = []
  Z = []
  for i in range(0, n):
    Y.append(PointY(i, X[i].x, X[i].y))
    Z.append(PointY())
  MergeSortY(Y, n)
  closet(X, Y, Z, 0, n - 1, List)
  return True

def closet(X, Y, Z, l, r, List):
  # list = [a, b, d]
```

```
# case: 2-points
if r - l == 1:
  List[0] = X[l]
  List[1] = X[r]
  List[2] = Distance(X[l], X[r])
  # print d
  return
# case: 3-points
elif r - l == 2:
  d1 = Distance(X[l], X[l + 1])
  d2 = Distance(X[l + 1], X[r])
  d3 = Distance(X[l], X[r])
  # d1 is minimum
  if d1 <= d2 and d1 <= d3:
    List[0] = X[l]
    List[1] = X[l + 1]
    List[2] = d1
    return
  # d2 is minimum
  elif d2 <= d3:
    List[0] = X[l + 1]
    List[1] = X[r]
    List[2] = d2
  # d3 is minimum
  else:
    List[0] = X[l]
    List[1] = X[r]
    List[2] = d3
# case: 3_plus-points
else:
  m = (l + r) / 2
  f = l
  g = m + 1
  for i in range(l, r + 1):
    if Y[i].index > m:
      Z[g] = Y[i]
      g += 1
    else:
      Z[f] = Y[i]
      f += 1
  closet(X, Z, Y, l, m, List)
```

```
'''
ar = PointX()
br = PointX()
dr = -1
'''
Listr = [PointX(), PointX(), -1]
closet(X, Z, Y, m + 1, r, Listr)
if Listr[2] < List[2]:
  List[0] = Listr[0]
  List[1] = Listr[1]
  List[2] = Listr[2]
Merge(Z, Y, l, m, r)
k = l
for i in range(l, r + 1):
  if math.fabs(Y[m].x - Y[i].x) < List[2]:
    Z[k] = Y[i]
for i in range (l, k):
  for j in range(i + 1, k):
    if Z[j].y - Z[i].y >= List[2]:
      break
    dp = Distance(Z[i], Z[j])
    if dp < List[2]:
      List[2] = dp
      List[0] = X[Z[i].index]
      List[1] = X[Z[j].index]
```