Divide and Conquer -

Closest Pair of Points

GeeksforGeeks

A computer science portal for geeks

Divide and Conquer -

Closest Pair of Points

GeeksforGeeks

A computer science portal for geeks

Problem Statement:-

We are given an array of n points in the plane, and the problem is to find out the closest pair of points in the array.

Recall the following formula for distance between two points p and q:-

$$|pq| = \sqrt{(px - qx)^2 + (py - qy)^2}$$

Problem Statement:-

We are given an array of n points in the plane, and the problem is to find out the closest pair of points in the array.

Recall the following formula for distance between two points p and q:-

$$|pq| = \sqrt{(px - qx)^2 + (py - qy)^2}$$

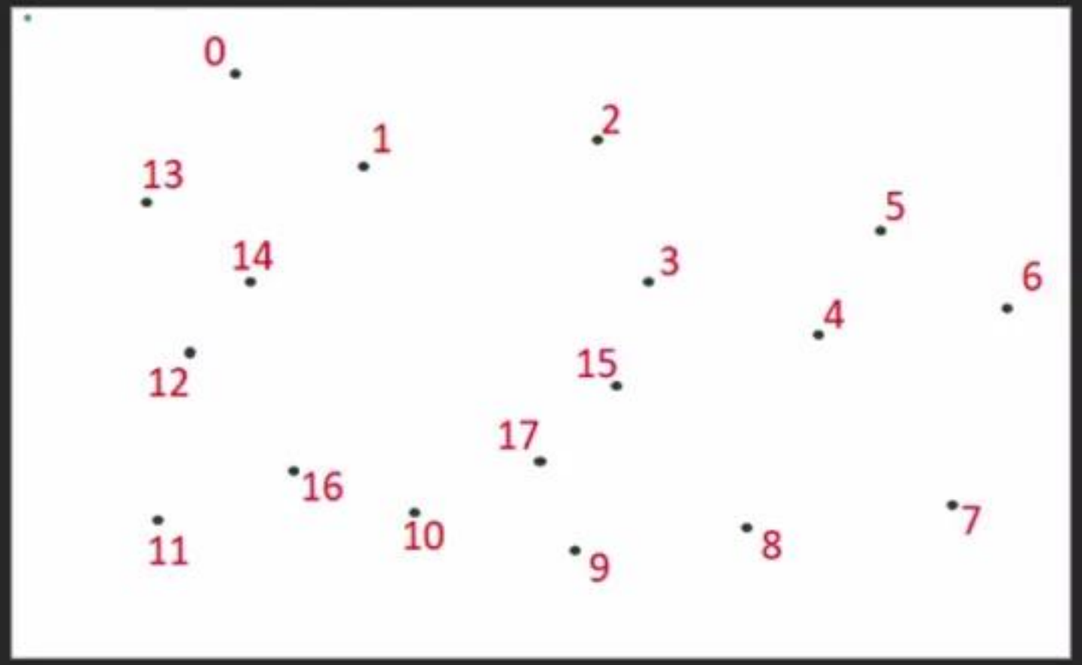
Algorithm:-

n=18

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

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

P[ ] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17}



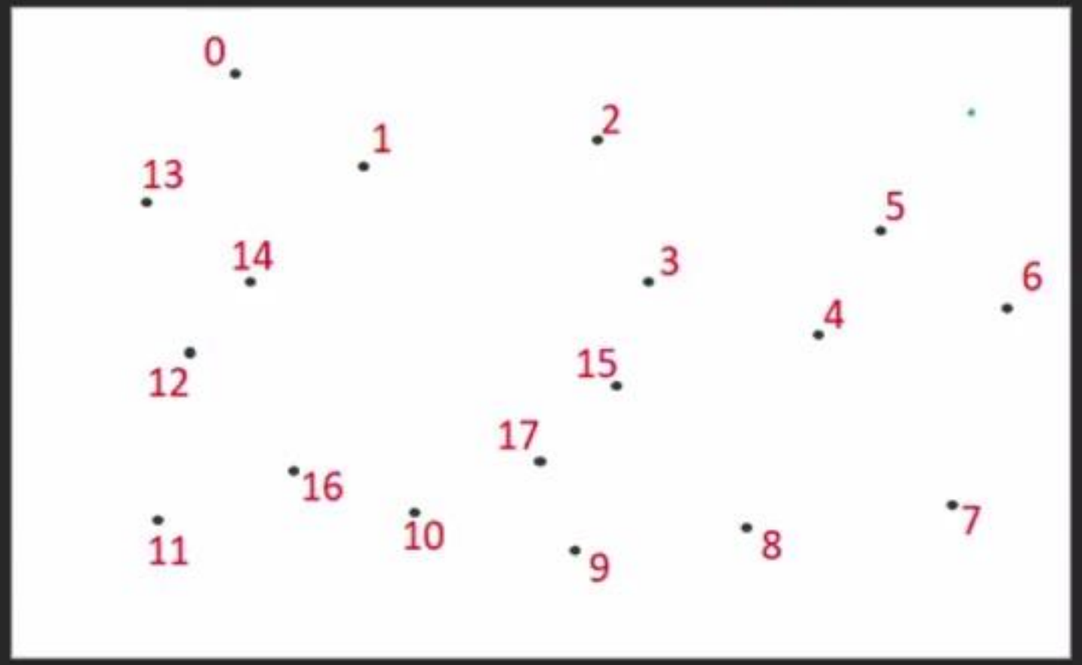
Algorithm:-

n=18

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.

P[ ] = {13,11,12,0,14,16,1,10,17,9,2,15,3,8,4,5,7,6}



P[n/2]=P[9]=17

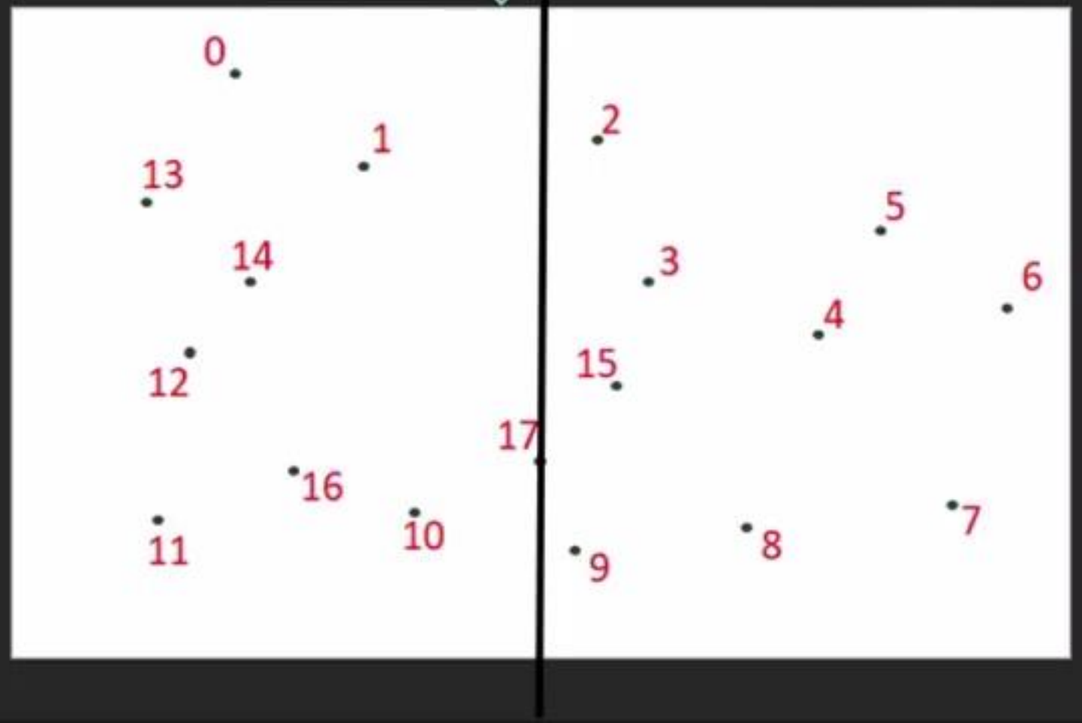
Algorithm:-

n=18

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].

P[ ] = {13,11,12,0,14,16,1,10,17,9,2,15,3,8,4,5,7,6}

P_L[ ] = {13,11,12,0,14,16,1,10,17} P_R[ ] = {9,2,15,3,8,4,5,7,6}



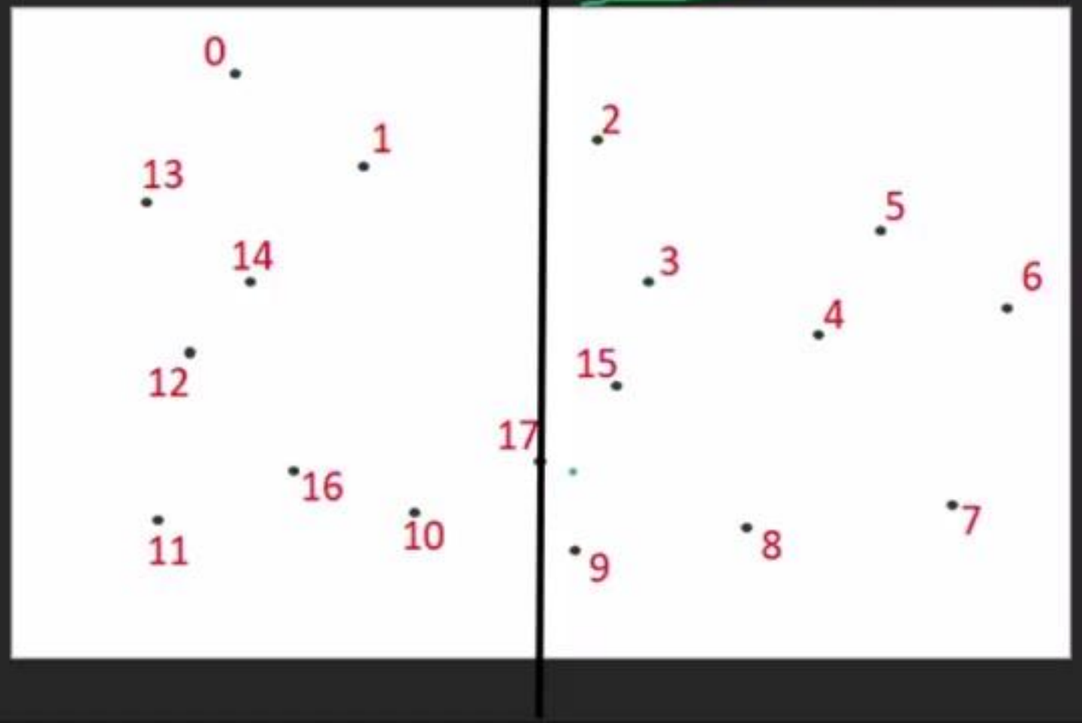
Algorithm:-

n=18

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].

P[ ] = {13,11,12,0,14,16,1,10,17,9,2,15,3,8,4,5,7,6}

P_L[ ] = {13,11,12,0,14,16,1,10,17} P_R[ ] = {9,2,15,3,8,4,5,7,6}



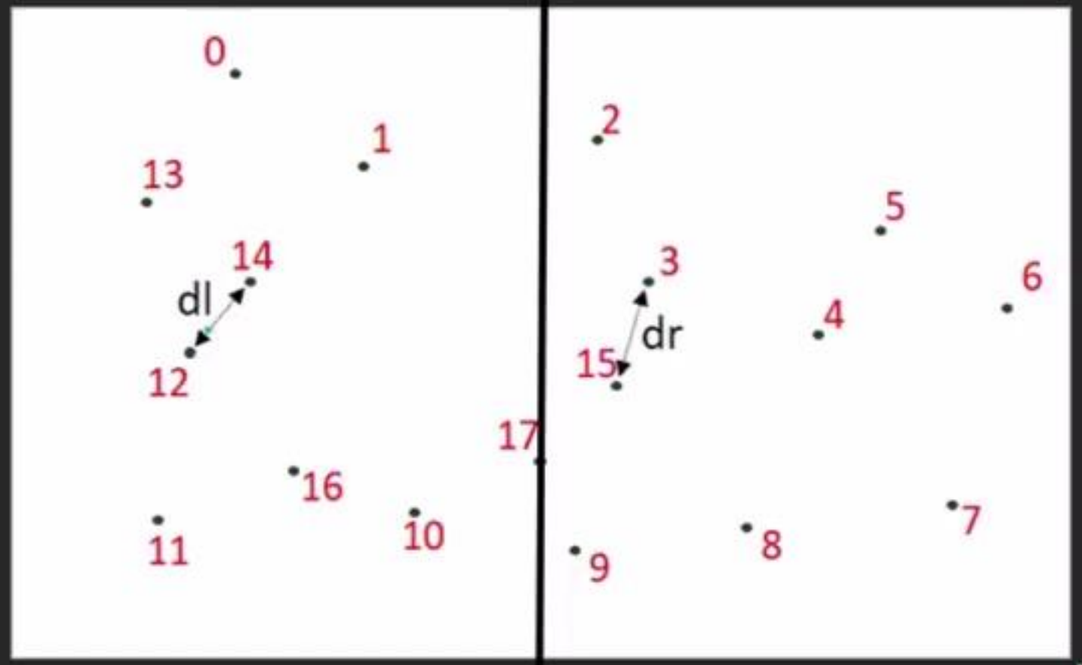
Algorithm:-

n=18

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.

P[ ] = {13,11,12,0,14,16,1,10,17,9,2,15,3,8,4,5,7,6}

P_L[ ] = {13,11,12,0,14,16,1,10,17} P_R[ ] = {9,2,15,3,8,4,5,7,6}



d=min(dl,dr)

$|pq| = \sqrt{(px - qx)^2 + (py - qy)^2}$

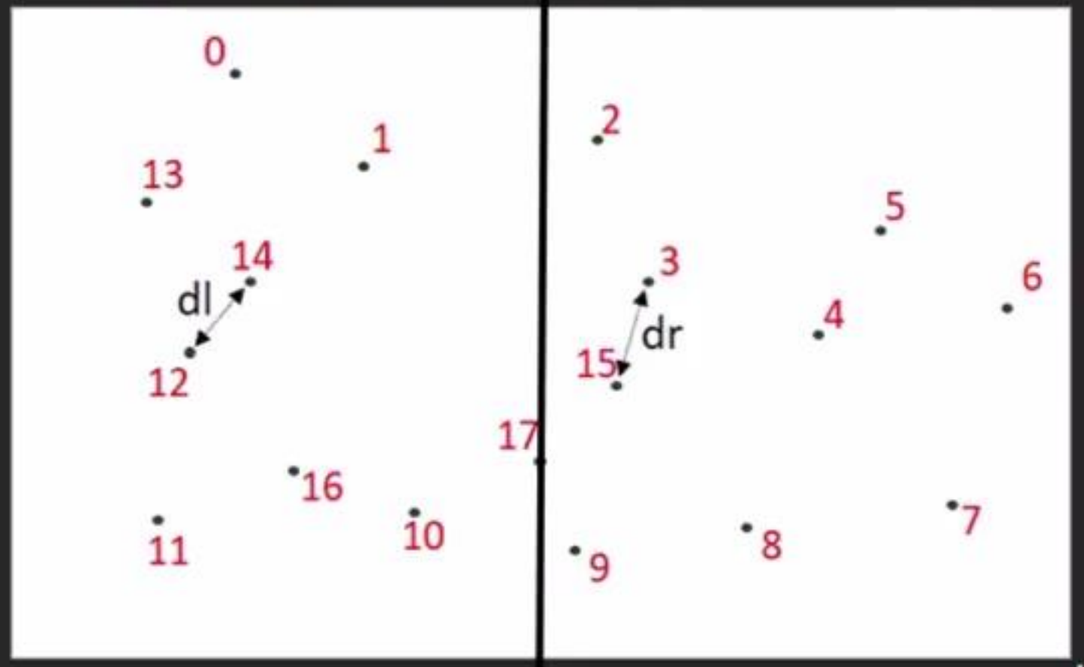
Algorithm:-

n=18

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.

P[ ] = {13,11,12,0,14,16,1,10,17,9,2,15,3,8,4,5,7,6}

P_L[ ] = {13,11,12,0,14,16,1,10,17} P_R[ ] = {9,2,15,3,8,4,5,7,6}



d=min(dl,dr)

$|pq| = \sqrt{(px - qx)^2 + (py - qy)^2}$

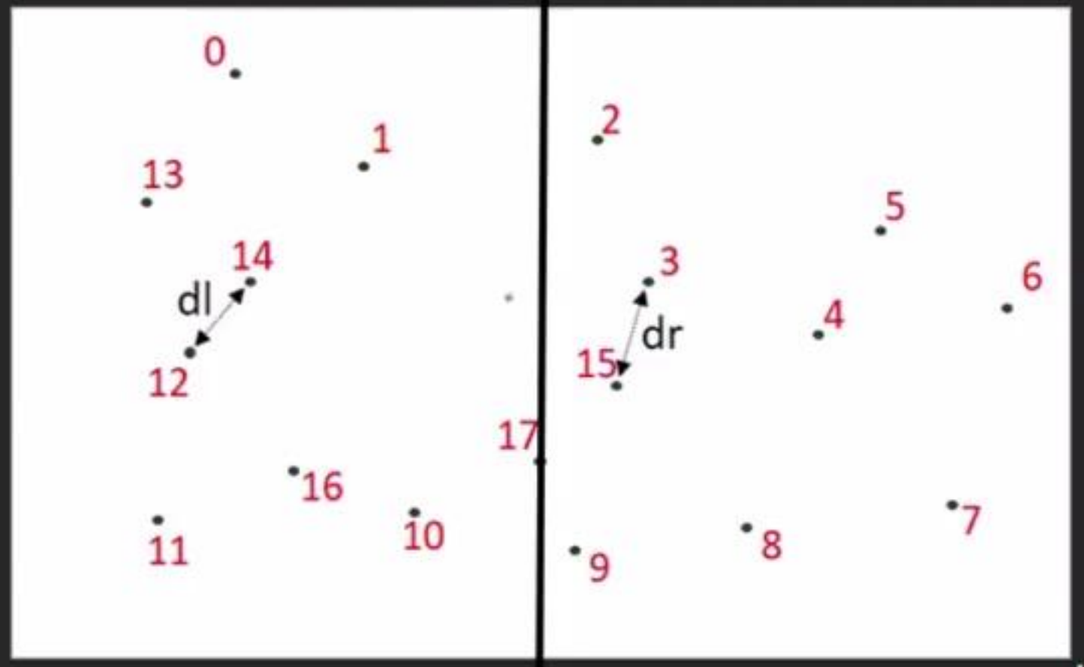
Algorithm:-

n=18

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.

P[ ] = {13,11,12,0,14,16,1,10,17,9,2,15,3,8,4,5,7,6}

P_L[ ] = {13,11,12,0,14,16,1,10,17} P_R[ ] = {9,2,15,3,8,4,5,7,6}



d=min(dl,dr)

$|pq| = \sqrt{(px - qx)^2 + (py - qy)^2}$

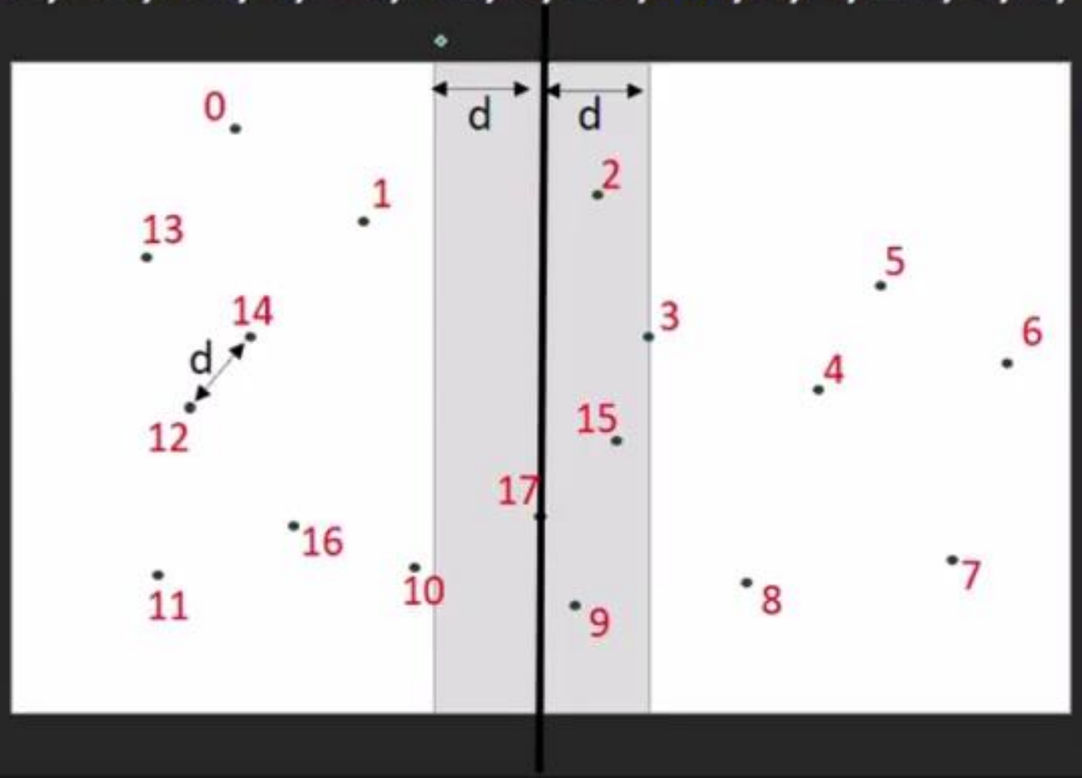
Algorithm:-

n=18

From above 3 steps, we have an upper bound d of minimum distance.

4) 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.

P[ ] = {13,11,12,0,14,16,1,10,17,9,2,15,3,8,4,5,7,6}



P_L[ ] = {13,11,12,0,14,16,1,10,17}

P_R[ ] = {9,2,15,3,8,4,5,7,6}

$|pq| = \sqrt{(px - qx)^2 + (py - qy)^2}$

d=min(dl,dr)

strip[ ] = {17,9,2,15,3}

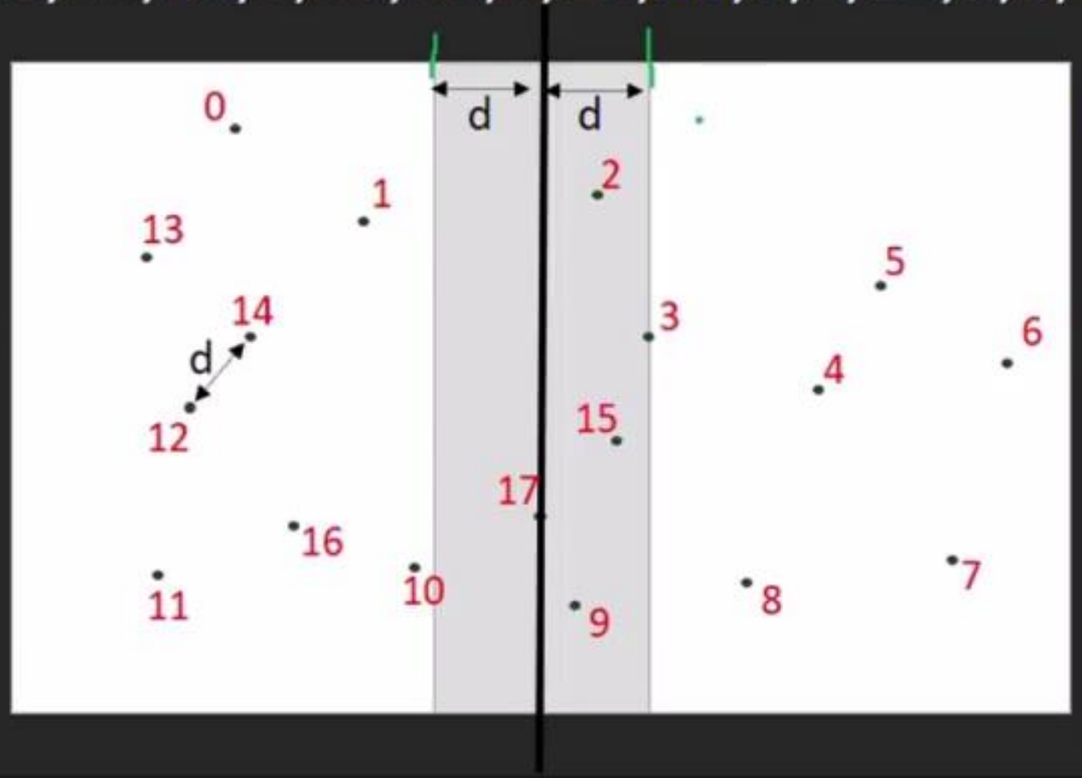
Algorithm:-

n=18

From above 3 steps, we have an upper bound d of minimum distance.

4) 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.

P[ ] = {13,11,12,0,14,16,1,10,17,9,2,15,3,8,4,5,7,6}



P_L[ ] = {13,11,12,0,14,16,1,10,17}

P_R[ ] = {9,2,15,3,8,4,5,7,6}

$|pq| = \sqrt{(px - qx)^2 + (py - qy)^2}$

d=min(dl,dr)

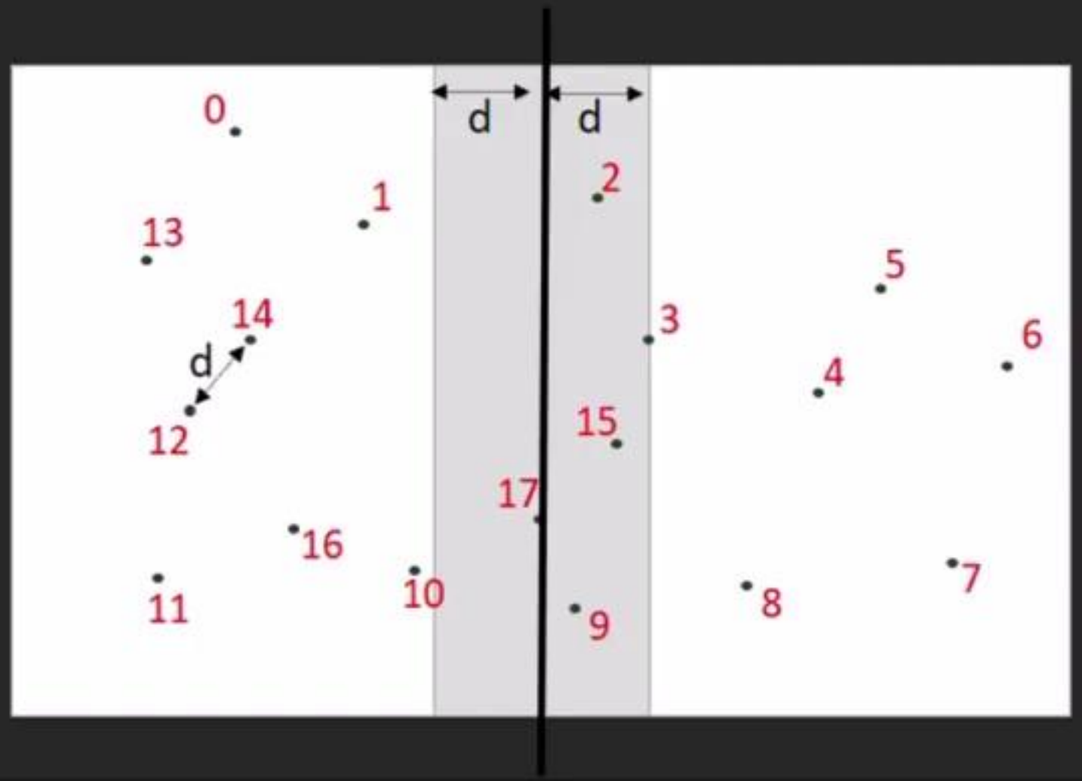
strip[ ] = {17,9,2,15,3}

Algorithm:-

n=18

5) Sort the array strip[] according to y coordinates. This step is O(nLogn).

strip[ ] = {2,3,15,17,9}



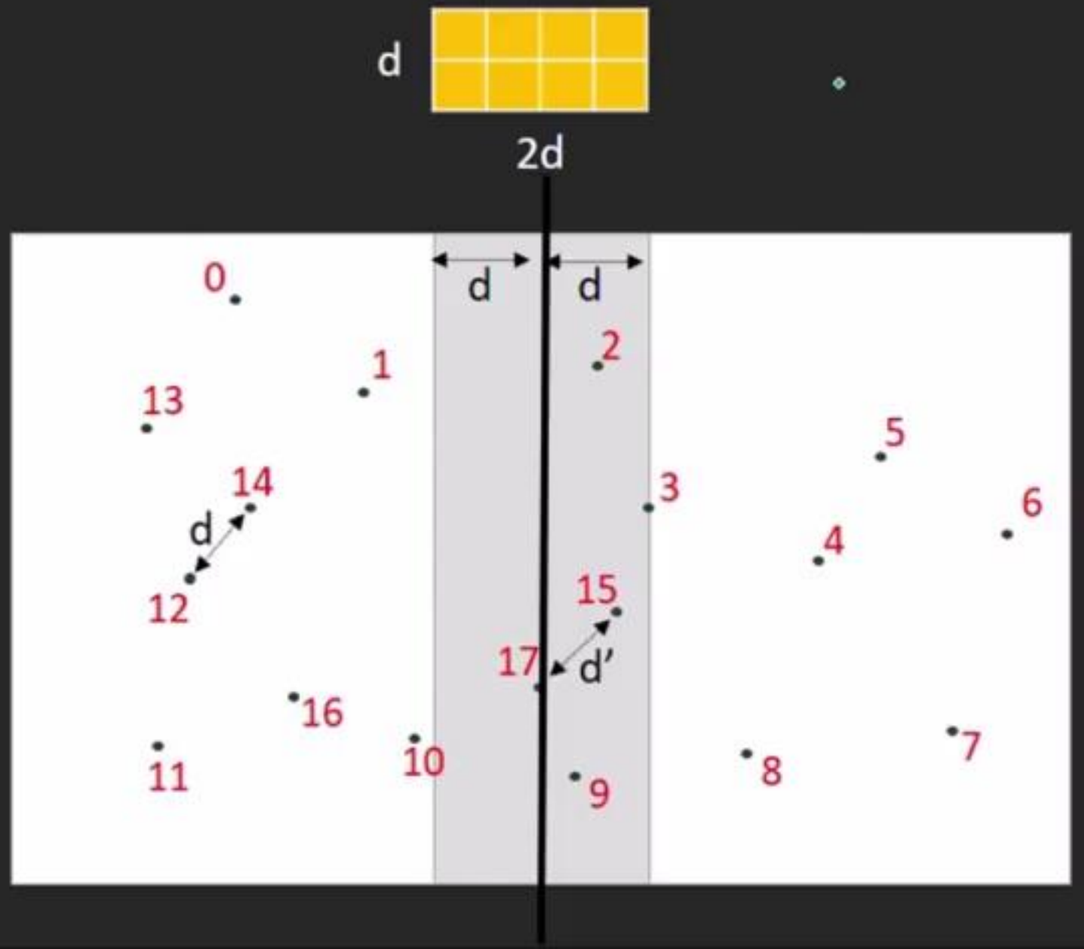
Algorithm:-

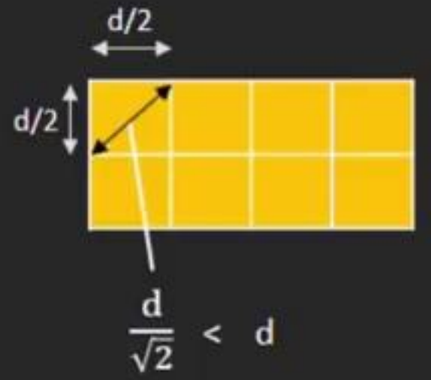
n=18

6) Find the smallest distance d' in strip[].

No square is shared between 2 halves. So, it's either on left or right.

strip[ ] = {2,3,15,17,9}





If 15 & 17 are the closest points in strip,
d' = distance b/w 15 & 17.

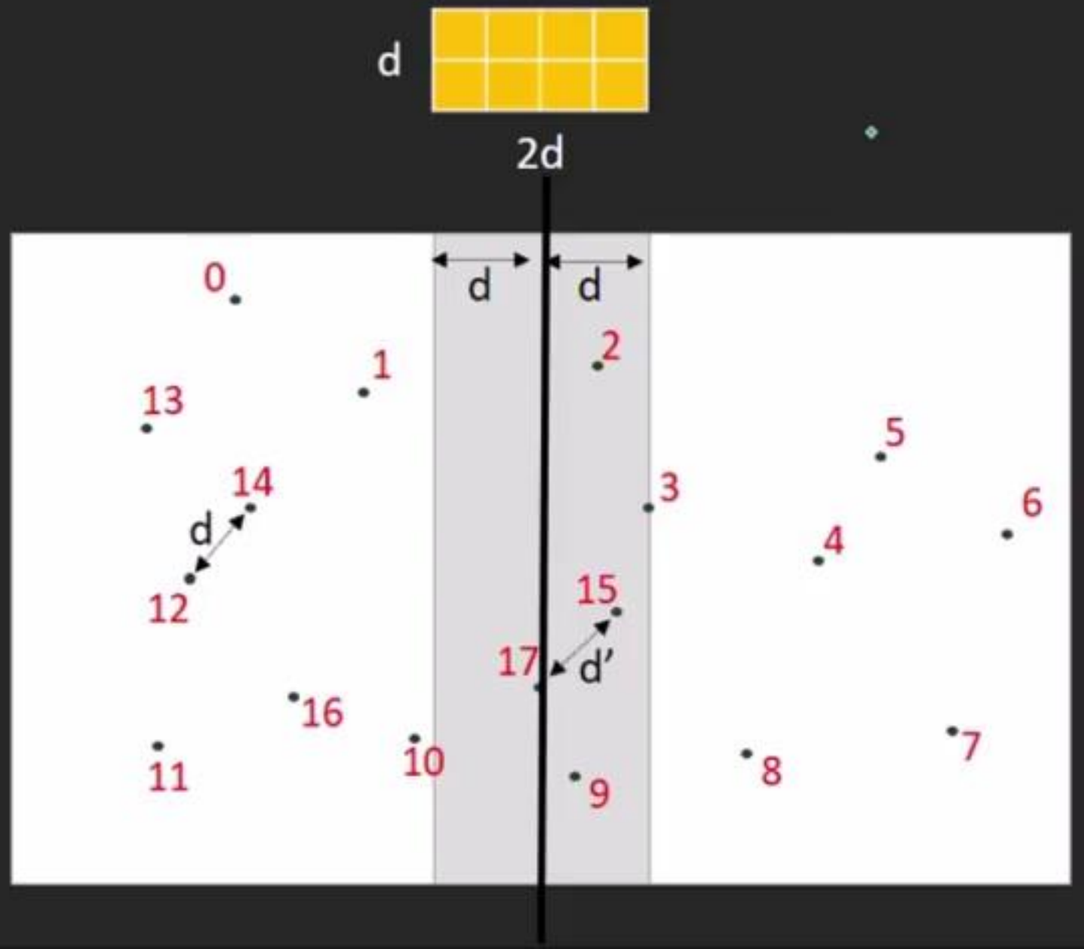
Algorithm:-

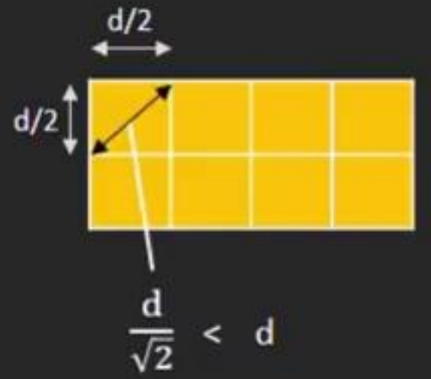
n=18

6) Find the smallest distance d' in strip[].

No square is shared between 2 halves. So, it's either on left or right.

strip[ ] = {2,3,15,17,9}





If 15 & 17 are the closest points in strip,
d' = distance b/w 15 & 17.

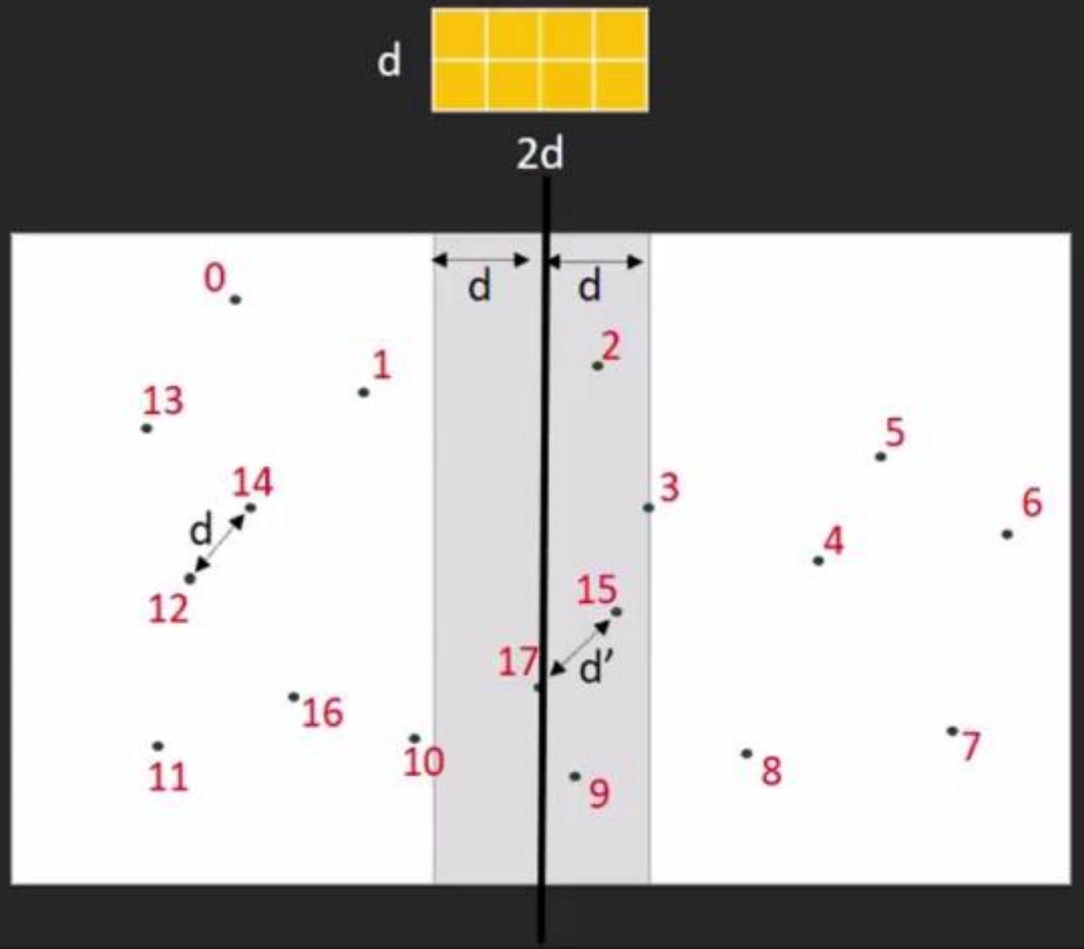
Algorithm:-

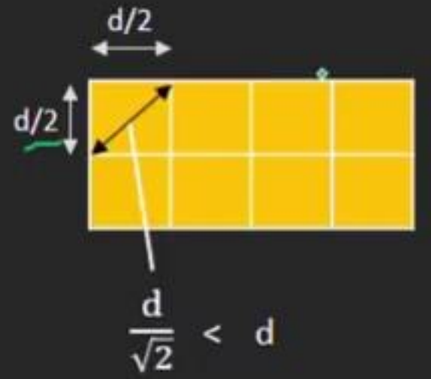
n=18

6) Find the smallest distance d' in strip[].

No square is shared between 2 halves. So, it's either on left or right.

strip[ ] = {2,3,15,17,9}





If 15 & 17 are the closest points in strip,
d' = distance b/w 15 & 17.

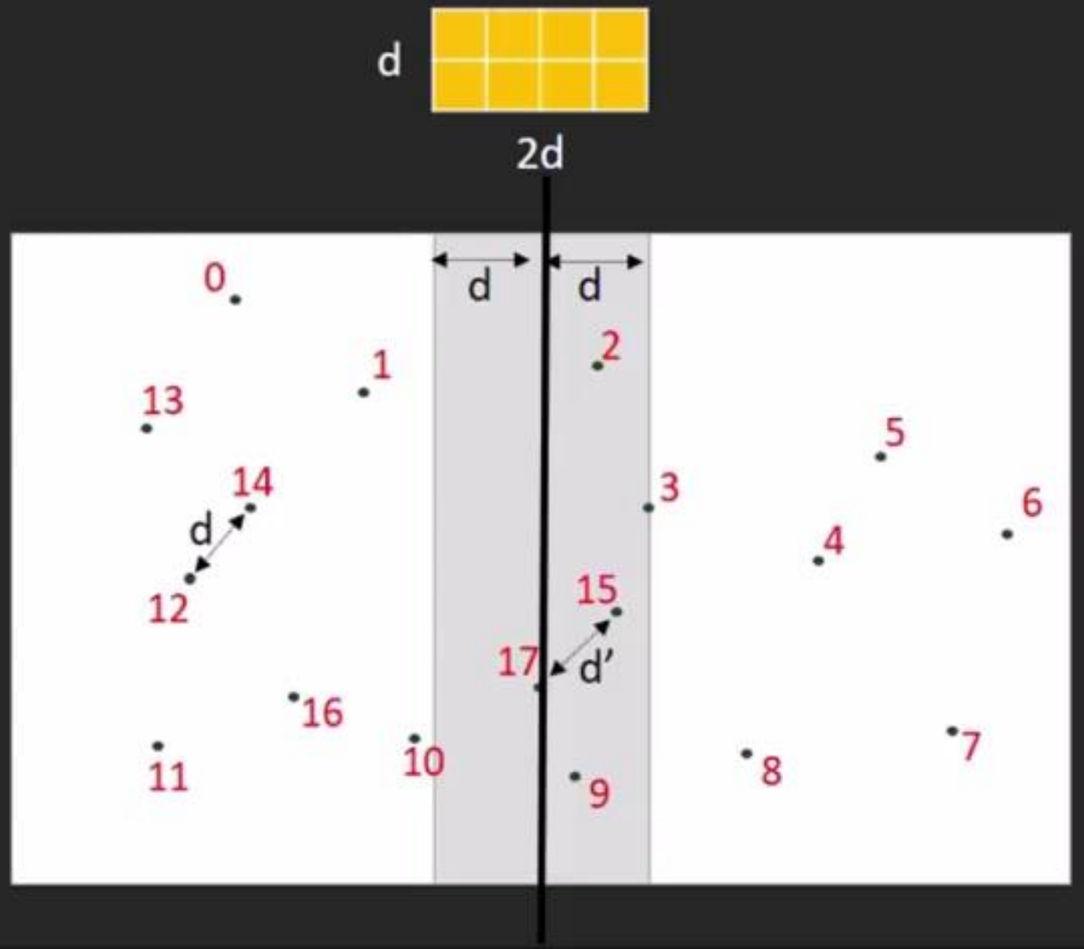
Algorithm:-

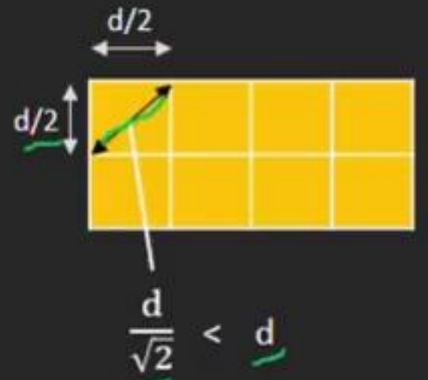
n=18

6) Find the smallest distance d' in strip[].

No square is shared between 2 halves. So, it's either on left or right.

strip[ ] = {2,3,15,17,9}





If 15 & 17 are the closest points in strip,
d' = distance b/w 15 & 17.

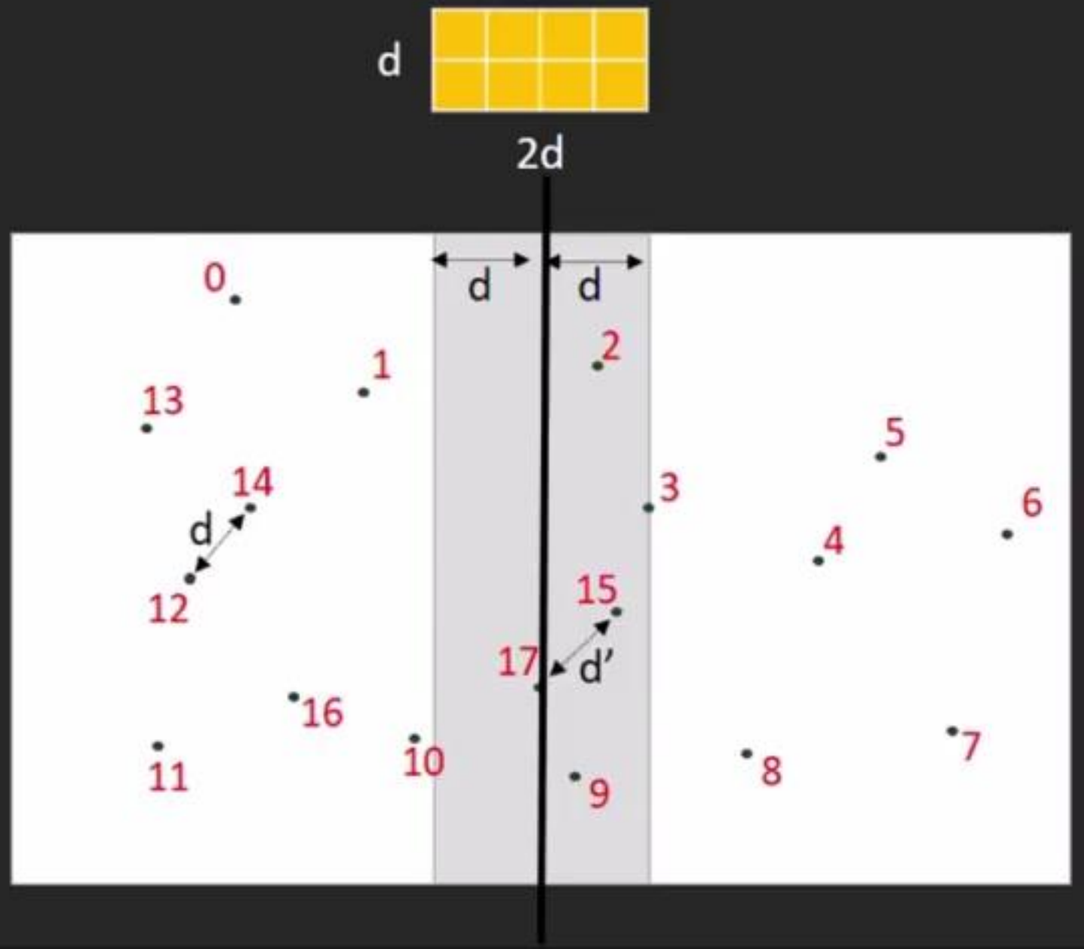
Algorithm:-

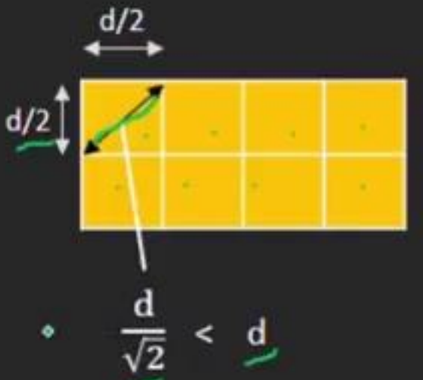
n=18

6) Find the smallest distance d' in strip[].

No square is shared between 2 halves. So, it's either on left or right.

strip[ ] = {2,3,15,17,9}





If 15 & 17 are the closest points in strip,
d' = distance b/w 15 & 17.

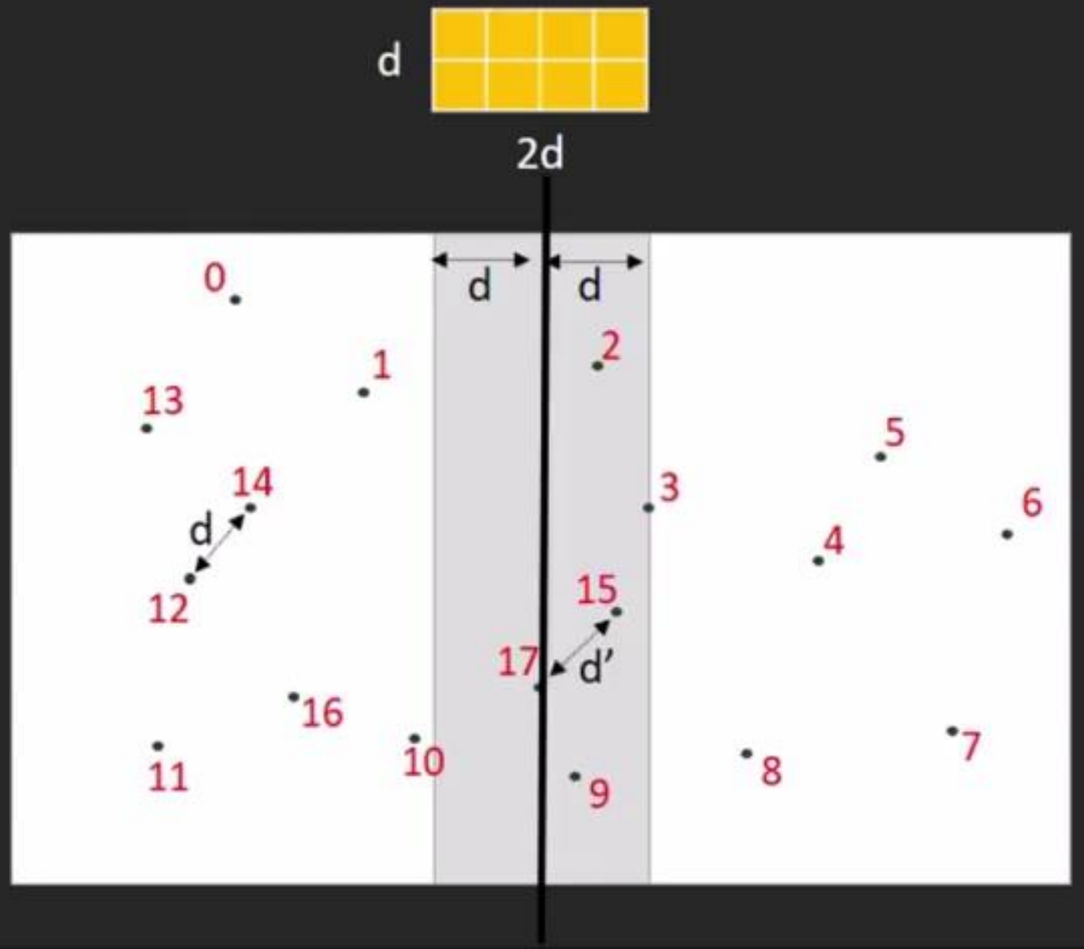
Algorithm:-

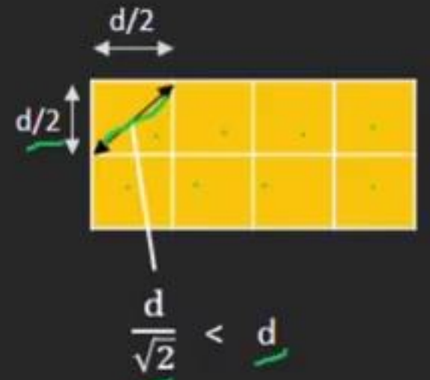
n=18

6) Find the smallest distance d' in strip[].

No square is shared between 2 halves. So, it's either on left or right.

strip[ ] = {2,3,15,17,9}





If 15 & 17 are the closest points in strip,

d' = distance b/w 15 & 17.

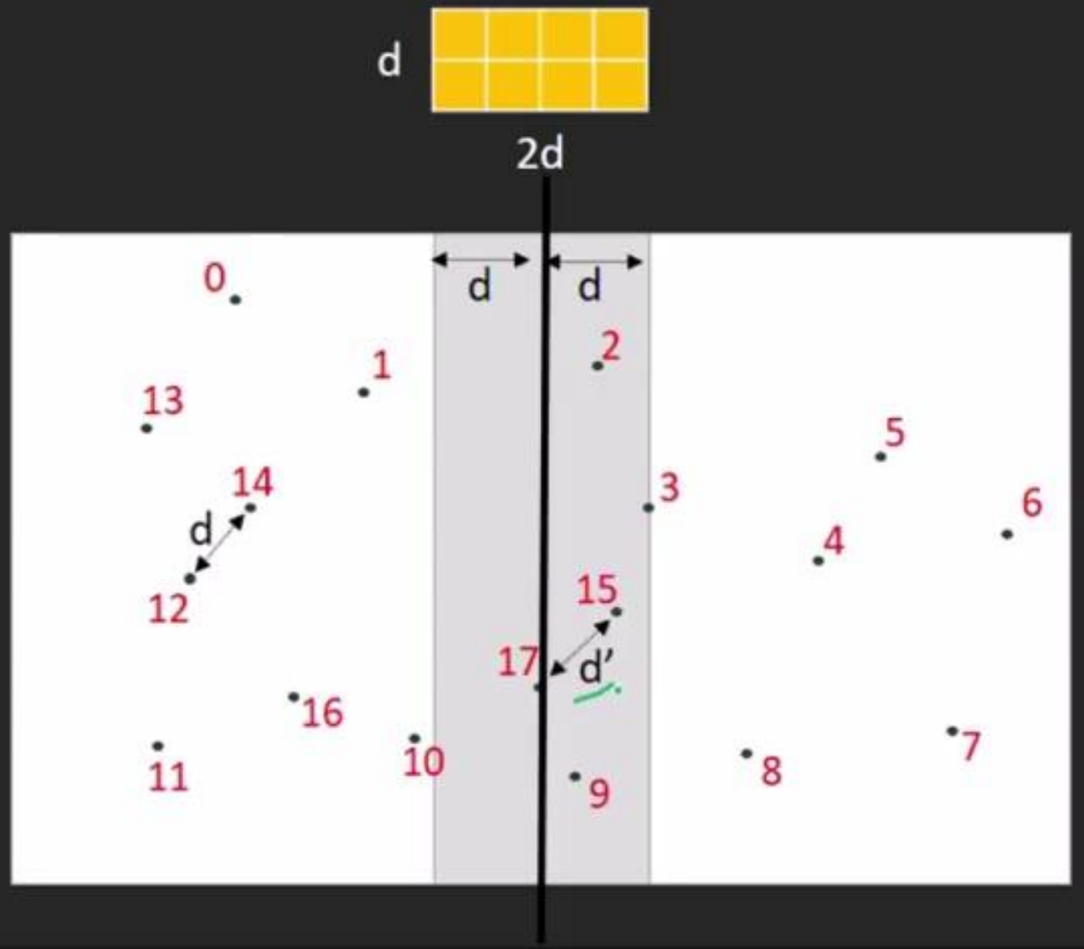
Algorithm:-

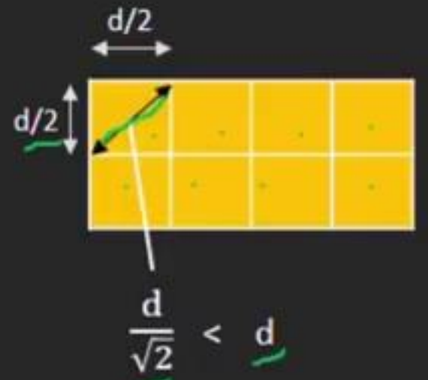
n=18

6) Find the smallest distance d' in strip[].

No square is shared between 2 halves. So, it's either on left or right.

strip[ ] = {2,3,15,17,9}





If 15 & 17 are the closest points in strip,
d' = distance b/w 15 & 17.

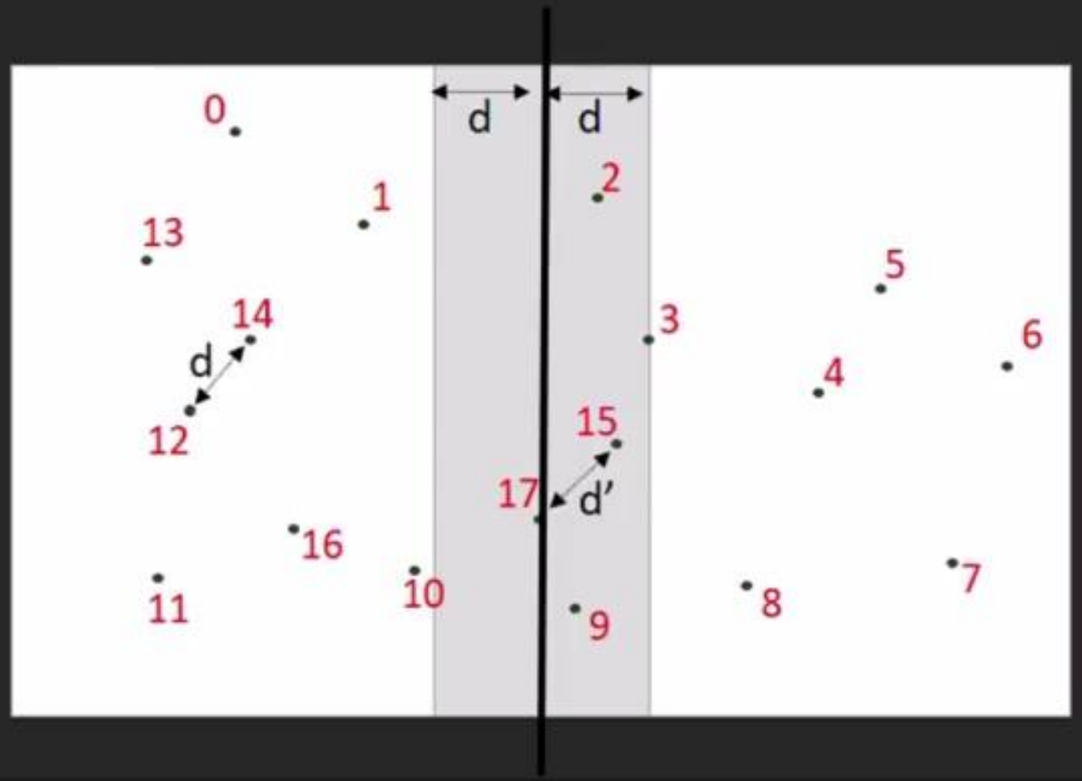
Algorithm:-

n=18

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

Return min(d,d')

Now, we have considered each point with every other point for the smallest distance (even if they lie on different sides) and have successfully computed the distance between the closest pair of points.



If d' < d *return d'*

else *return d*

Understanding the code:-

```
float closestUtil(Point P[], int n)
{
    if (n <= 3)
        return bruteForce(P, n);
    int mid = n/2;
    Point midPoint = P[mid];
    float dl = closestUtil(P, mid);
    float dr = closestUtil(P + mid, n-mid);
    float d = min(dl, dr);
    Point strip[n];
    int j = 0;
    for (int i = 0; i < n; i++)
        if (abs(P[i].x - midPoint.x) < d)
            strip[j] = P[i], j++;

    return min(d, stripClosest(strip, j, d) );
}
```

```
float closest(Point P[], int n)
{
    qsort(P, n, sizeof(Point), compareX);
    return closestUtil(P, n);
}

int main()
{
    Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3,
4}};
    int n = sizeof(P) / sizeof(P[0]);
    printf("The smallest distance is %f ", closest(P, n));
    return 0;
}
```

Understanding the code:-

```
float closestUtil(Point P[], int n)
{
  if (n <= 3)
      return bruteForce(P, n);
  int mid = n/2;
  Point midPoint = P[mid];
  float dl = closestUtil(P, mid);
  float dr = closestUtil(P + mid, n-mid);
  float d = min(dl, dr);
  Point strip[n];
  int j = 0;
  for (int i = 0; i < n; i++)
      if (abs(P[i].x - midPoint.x) < d)
          strip[j] = P[i], j++;

  return min(d, stripClosest(strip, j, d) );
}
```

```
float closest(Point P[], int n)
{
   qsort(P, n, sizeof(Point), compareX);
   return closestUtil(P, n);
}

int main()
{
   Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3,
4}};
   int n = sizeof(P) / sizeof(P[0]);
   printf("The smallest distance is %f ", closest(P, n));
   return 0;
}
```

Understanding the code:-

```
struct Point
{
    int x, y;
};
int compareX(const void* a, const void* b)
{
    Point *p1 = (Point *)a,  *p2 = (Point *)b;
    return (p1->x - p2->x);
}
int compareY(const void* a, const void* b)
{
    Point *p1 = (Point *)a,   *p2 = (Point *)b;
    return (p1->y - p2->y);
}

float dist(Point p1, Point p2)
{
    return sqrt( (p1.x - p2.x)*(p1.x - p2.x) +
                 (p1.y - p2.y)*(p1.y - p2.y)
               );
}
```

```
float bruteForce(Point P[], int n)
{
    float min = FLT_MAX;
    for (int i = 0; i < n; ++i)
        for (int j = i+1; j < n; ++j)
            if (dist(P[i], P[j]) < min)
                min = dist(P[i], P[j]);
    return min;
}
 float min(float x, float y)
{
    return (x < y)? x : y;
}
 float stripClosest(Point strip[], int size, float d)
{
    float min = d;
    qsort(strip, size, sizeof(Point), compareY);
    for (int i = 0; i < size; ++i)
        for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min; ++j)
            if (dist(strip[i],strip[j]) < min)
                min = dist(strip[i], strip[j]);

    return min;
}
```

Understanding the code:-

```
float closestUtil(Point P[], int n)
{
    if (n <= 3)
        return bruteForce(P, n);
    int mid = n/2;
    Point midPoint = P[mid];
    float dl = closestUtil(P, mid);
    float dr = closestUtil(P + mid, n-mid);
    float d = min(dl, dr);
    Point strip[n];
    int j = 0;
    for (int i = 0; i < n; i++)
        if (abs(P[i].x - midPoint.x) < d)
            strip[j] = P[i], j++;

    return min(d, stripClosest(strip, j, d) );
}
```

```
float closest(Point P[], int n)
{
    qsort(P, n, sizeof(Point), compareX);
    return closestUtil(P, n);
}

int main()
{
    Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3,
4}};
    int n = sizeof(P) / sizeof(P[0]);
    printf("The smallest distance is %f ", closest(P, n));
    return 0;
}
```

Understanding the code:-

```
struct Point
{
    int x, y;
};
int compareX(const void* a, const void* b)
{
    Point *p1 = (Point *)a,  *p2 = (Point *)b;
    return (p1->x - p2->x);
}
int compareY(const void* a, const void* b)
{
    Point *p1 = (Point *)a,   *p2 = (Point *)b;
    return (p1->y - p2->y);
}

float dist(Point p1, Point p2)
{
    return sqrt( (p1.x - p2.x)*(p1.x - p2.x) +
                 (p1.y - p2.y)*(p1.y - p2.y)
               );
}
```

```
float bruteForce(Point P[], int n)
{
    float min = FLT_MAX;
    for (int i = 0; i < n; ++i)
        for (int j = i+1; j < n; ++j)
            if (dist(P[i], P[j]) < min)
                min = dist(P[i], P[j]);
    return min;
}
 float min(float x, float y)
{
    return (x < y)? x : y;
}
 float stripClosest(Point strip[], int size, float d)
{
    float min = d;
    qsort(strip, size, sizeof(Point), compareY);
    for (int i = 0; i < size; ++i)
        for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min; ++j)
            if (dist(strip[i],strip[j]) < min)
                min = dist(strip[i], strip[j]);

    return min;
}
```

Understanding the code:-

```
struct Point
{
    int x, y;
};
int compareX(const void* a, const void* b)
{
    Point *p1 = (Point *)a,  *p2 = (Point *)b;
    return (p1->x - p2->x);
}
int compareY(const void* a, const void* b)
{
    Point *p1 = (Point *)a,   *p2 = (Point *)b;
    return (p1->y - p2->y);
}

float dist(Point p1, Point p2)
{
    return sqrt( (p1.x - p2.x)*(p1.x - p2.x) +
                 (p1.y - p2.y)*(p1.y - p2.y)
               );
}
```

```
float bruteForce(Point P[], int n)
{
    float min = FLT_MAX;
    for (int i = 0; i < n; ++i)
        for (int j = i+1; j < n; ++j)
            if (dist(P[i], P[j]) < min)
                min = dist(P[i], P[j]);
    return min;
}
 float min(float x, float y)
{
    return (x < y)? x : y;
}
 float stripClosest(Point strip[], int size, float d)
{
    float min = d;
    qsort(strip, size, sizeof(Point), compareY);
    for (int i = 0; i < size; ++i)
        for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min; ++j)
            if (dist(strip[i],strip[j]) < min)
                min = dist(strip[i], strip[j]);

    return min;
}
```

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. Time for this is 2T(n/2).
- After dividing, it finds the strip in O(n) time.
- Sorting the strip in done in O(nLogn) time.
- Finally find the closest points in strip in O(n) time as explained earlier.
- So T(n) can expressed as follows:-

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

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

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

Hence, $T(n) = O(n(logn)^2)$

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. Time for this is 2T(n/2).
- After dividing, it finds the strip in O(n) time.
- Sorting the strip in done in O(nLogn) time.
- Finally find the closest points in strip in O(n) time as explained earlier.
- So T(n) can expressed as follows:-

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

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

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

Hence, $T(n) = O(n(logn)^2)$

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. Time for this is 2T(n/2).
- After dividing, it finds the strip in O(n) time.
- Sorting the strip in done in O(nLogn) time.
- Finally find the closest points in strip in O(n) time as explained earlier.
- So T(n) can expressed as follows:-

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

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

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

Hence, $T(n) = O(n(logn)^2)$

Thank you for watching!
Please leave us your comments.

CONVERT YOUTUBE VIDEO INTO JPG/PDF IMAGES

WWW.GALLERYMKER.COM