Closest Pair

- Problem
  - Given n points on a two-dimension space, find the closest pair
- A simple algorithm
  - Calculate the distance for all possible pairs, find a smallest one
    - Total $\binom{n}{2}$ pairs
    - Cost: $\Theta(n^2)$
- A better algorithm
  - Divide-and-conquer

A Divide-and-Conquer Algorithm

1. Split points equally half-by-half based on the x-coordinate
2. Find the closest pair for left half and the right half
3. Based on the results in step 2, find the closest pair for the original sets

- Cost
  - $T(n) = 2T(n/2)+g(n)$
  - $T(n) \in O(n \log n)$ if $g(n) \in \Theta(n)$

Algorithm

```
double closestPair(Points p)
{
   n = p.size();
   mergeSort(p);  // by x-coordinate
   return recursiveClosestPair(p, 1, n)
}
```

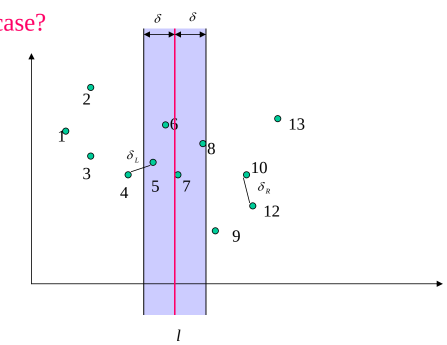
```
double recursiveClosestPair(p, i, j)
{
   if ( j-i < 3) {
     return adhocClosest(p, i, j);
     sort p[i..j] by y-coordinate;
   }

   k = (i+j)/2;
   deltaL = recursiveClosestPair(p, i,
     k);
   deltaR = recursiveClosestPair(p, i,
     k);
   delta = min(deltaL, deltaR);
   return findClosestInStrip(p, i, j,
     delta);
}
```

Note: p[i..j] are sorted by x-coordinate before getting in recursiveCloestPair();
sorted by y-coordinate after it returns;

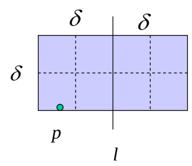
Observation 1:

- We only need to compare the pairs cross the bound
  - Only consider the points in the gray strip
  - Worst case?



Observation 2:

- At most 8 points in a $\delta \times 2\delta$ rectangle
  - Each $\frac{\delta}{2}\times\frac{\delta}{2}$ square can contain at most one point
  - Only consider distance to 7 other closest points by y-coordinate



findClosestInStrip

m = j-i+1

```
findClosestInStrip(p, i, j, delta)
{
   k = (i+j)/2;        l = p[k].x;
   // p[i..k] sorted by y-coordinate
   // p[k+1..j] sorted by y-coordinate
   merge(p, i, k, j);  // p[i..j] sorted by y-coordinate

   t = 0;
   for (k=i; k<=j; k++) {
      if (p[k].x > l – delta
        && p[k].x < l+delta)  // in the strip
         v[++t] = p[k];
   }
   for (k=1; k<t; k++) {
         for (s=k+1; s<=min(t, k+7); s++)
         delta = min(delta, dist(v[k], v[s]));
   }
   return delta
}
```

Cost?

Θ(m)

Θ(m)

O(m)

Total: Θ(m)

Algorithm Analysis

```
double closestPair(Points p)
{
   n = p.size();
   mergeSort(p);                        ← Θ(n logn)
   return recursiveClosestPair(p, 1, n)
}
```

?

Algorithm Analysis

```
double recursiveClosestPair(p, i, j)    ← T(n)
{
   if ( j-i < 3) {
     return adhocClosest(p, i, j);
   }

   k = (i+j)/2;
   deltaL = recursiveClosestPair(p, i, k);   ← T(n/2)
   deltaR = recursiveClosestPair(p, i, k);   ← T(n/2)
   delta = min(deltaL, deltaR);
   return findClosestInStrip(p, i, j, delta);   ← Θ(n)
}
```

$T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$