# *Closest Pair Report*

*Kasper Hjort Berthelsen, Jesper Kato Jensen, Frederik Schou Madsen, Heidi Regina Schröder, Julie Schou Sørensen & Jon Voigt Tøttrup*

*September 30, 2018*

## *Results*

Our implementation produces the expected results on all input–output file pairs.

The following table shows the closest pairs in the input files `wc-instance-65534.txt`. Here $n$ denotes the number of points in the input, and $(u, v)$ denotes a closest pair of points at distance $\delta$.

| $n$ | $u$ | $v$ | $\delta$ | running time |
|---:|---:|---:|---:|---:|
| 65.534 | (0.5,-0.0) | (-0.5,0.0) | 1.0 | $\sim$ 1 sec |

## *Implementation details*

The implementation follows Section 5.4 of Kleinberg and Tardos, *Algorithm Design*, Addison–Wesley 2006. It is a divide and conquer technique that divides the two-dimensional space and works by recurrence.

We start by constructing the set of points $P = \{p_1, ..., p_n\}$. Each point is implemented as a `point`-class that has a name, a $x$-coordinate and a $y$-coordinate as well as a `distance`-method. The total set of points is then implemented as an ArrayList of points.

After constructing $P$ we construct $P_x$ and $P_y$, that is, $P$ sorted by its $x$-coordinate and $y$-coordinate, respectively. This takes $n \log n$ time. We call the recurrence `ClosestPairRec`, that divides the space in a left half, $Q$, and a right half, $R$. Again we construct sets that are sorted by $x$-coordinates and $y$-coordinates; $Q_x, Q_y, R_x$ and $R_y$. $Q_x$ and $R_x$ are found in linear time by looping though $P_x$, but $Q_y$ and $R_y$ are sorted again, which factors the fastest found running time by $n \log n$.

For the comparison of the points in the middle of the two half, we construct the set of sorted points $S_y = \{s : x^* - \delta \geq s \geq x^* + \delta\}$, where $x^*$ is the right most point in $Q$. For each point $s$ we inspect the following 15 points in $S_y$, as explained (5.10) of Kleinberg and Tardos, *Algorithm Design*, Addison–Wesley 2008. Below you find the corresponding part of our code.

```
closestPair.p0 = P.get(0);
```

```
closestPair.p1 = P.get(1);
dmin = P.get(0).distance(P.get(1));

for(int i=0;i<P.size()-1;i++) {
    for(int j=1; j<Math.min(P.size()-i, 15); j++) {
        d = P.get(i).distance(P.get(i+j));
        if(d<dmin) {
            dmin = d;
            closestPair.p0 = P.get(i);
            closestPair.p1 = P.get(i+j);
        }
    }
}
return closestPair;
```

Because of the extra sorting in each recurrence step, our running time is $O(n \log^2 n)$ for $n$ points.