A Note Concerning the Closest Point Pair Algorithm

Martin Richards[a]

[a] *University Computer Laboratory*
New Museums Site, Pembroke St., Cambridge CB2 3QG, UK
mr@cl.cam.ac.uk

Abstract

An algorithm, described by Sedgewick, finds the distance between the closest pair of n given points in a plane using a variant of mergesort. This takes $O(n \log n)$ time. To prove this it is necessary to show that, in the merge phase of the algorithm, no more than a constant number of distances need to be checked for each point considered. Cormen, Leiserson and Rivest show that checking seven distances is sufficient while Sedgewick suggests that this should be four. This paper shows that checking three distances is sufficient and that a slight modification of the algorithm reduces the number to two.

Key words: closest point pair algorithm; mergesort; analysis of algorithms

1 The Algorithm

To find the minimum separation of any pair of n given points in a plane, a list of the points is formed ordered by their x-coordinates and a global variable *min*, to hold the minimum separation distance, is initialised to infinity. The algorithm then calls a routine, SORT, that is a modified version of merge sort, to return the list of points ordered by their y-coordinates. During the merge phase, *min* is updated whenever a closer pair of points is discovered.

If SORT is given a list containing fewer than two points, it returns the list unchanged, otherwise it applies SORT recursively to the first and second halves of the given list in turn. At this stage, the minimum separation of any pair of points in either half will be in *min*. If a closer pair exists, its points must be on either side of an imaginary line separating the two halves with each point no further than *min* from it. An efficient check for such pairs can be made as the two lists are merged to form the sorted result. If P, the next point to be merged, lies within a distance min of the dividing line, it is compared with a few other similar recently processed points. Cormen, Leisersen and Rivest[1] show that it is sufficient to check P with no more than seven other points. Manber[2] suggests the number is five while Sedgewick[3] suggests four. This paper shows that checking three distances is sufficient.

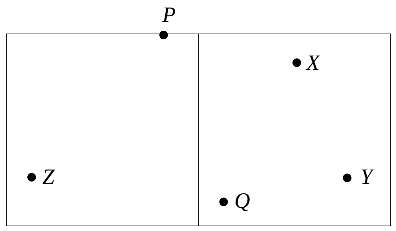
2 Proof

Let min_0 be the value of min at the start of the current merge phase and let the next point to be merged that is within min_0 of the dividing line be called P. P could potentially be one of a closer pair and is therefore pushed onto a stack of such points. Suppose the next four points on the stack are A, B, C, and Q. They clearly satisfy $P_y \geq A_y \geq B_y \geq C_y \geq Q_y$[1]. The algorithm checks whether any of the distances PA, PB or PC[2] are less than min. It does not have to check PQ since, as will be shown, if $PQ < min$ then P will be at least as close to one of A, B, or C.

Consider the $2min_0 \times min_0$ rectangle centered about the dividing line and having P on its top edge. The rectangle is composed of two adjacent $min_0 \times min_0$ squares on either side of the dividing line. Without loss of generality, assume P is on the top edge of the left hand square. If $PQ < min$, Q must be in the right hand square and A, B and C must all be in the rectangle. We have to consider four cases depending on how many of A, B and C are in the right hand square.

1. If A, B, C and Q are all in the right hand square they will have a minimum separation of min_0 and so can only be at its corners. P will be closest to the point at the top left corner which cannot be Q.

2. The next case is when just one of A, B or C in is the left hand square. Let that point be renamed Z and the other two renamed X and Y. The arrangement might be as follows:



[1] P_x and P_y denote the x- and y-coordinates of point P, and similarly for other points

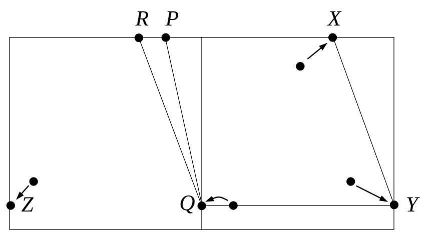
[2] PA denotes the distance between points P and A, and similarly for other point pairs

It follows from $X_y \geq Q_y$ and $PQ < PX$ that $X_x \geq Q_x$. We can similarly deduce $Y_x \geq Q_x$. Without loss of generality, assume that $X_x \leq Y_x$. Since Q, X and Y are on the same side they must be at least min_0 apart, and since no two points in their square can be more than $\sqrt{2}min_0$ apart, we can deduce angle $\angle QXY \leq 90°$. This combined with $X_x \geq Q_x$, $X_y \geq Q_y$ and $X_x \leq Y_x$ implies $X_y \geq Y_y$.

If the arrangement satisfies the constraints given above, then another valid arrangement can be obtained by moving Z to the point on the left edge at the same level as Q, moving Q horizontally to the middle edge, moving Y horizontally to the right edge and then down to the same level as Q, and, finally, moving X onto the top edge and to the right until $XY = min_0$.

Clearly this does not increase PQ, and QY remains $\geq min_0$. Since X is on or above the circle radius min_0 centered at the new position of Q, it is above the line joining P with the new position of Y. Moving X to the top edge of the rectangle, in a direction orthogonal to this line, will therefore increase both PX and PY. So if PX is now $< PQ$, this would have been true of the original arrangement.

If we now add a point R to the top edge such that RQ is parallel to XY, we have the following:



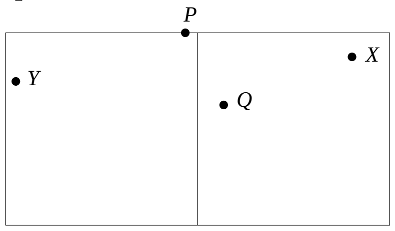
Note that $RXYQ$ is a rhombus with sides of length min_0. Since $PQ < min(\leq min_0)$, P must be to the right of R forcing the $PX \leq PQ$. This shows that the distance PQ does not need to be checked when there are two other points above Q in the right square and one other in the left.

3. The next case when two of A, B and C shares the left hand square with P. By swapping P and Q and inverting the figure this arrangement can be seen to be equivalent to the previous case and so the same result holds.

4. Finally, if A, B and C are in the left square with P, they must be at its vertices. Q must then be on the bottom edge which violated the constraint $PQ < min$

We can thus deduce, for all cases, that if $PQ < min$ then P is at least as close to A, B or C. In practice, the algorithm only pushes P onto the stack if it is less than min (not min_0) from the dividing line. This can only reduce the number of points in the stack and so does not violate the result.

By considering an arrangement such as the following, it is clear that checking P with only the top two points in the stack is insufficient.



3 Final Comments

The number of checks can be further reduced to two by the observation that only two of PA, PB and PC need be checked, since, either at least one of A, B or C will be on the same side as P, or they will all be opposite to P, in which case PC will be no smaller than PA or PB. To implement this two stacks are used, one for the left side and one for the right. P is pushed onto the stack for its side and then the distances are checked between P and the top two points on the other stack.

Reducing the number of distance checks from four to three or even two makes no noticeable speed improvement in practice since, for typical data, having more than one point in the rectangle is extremely rare.

References

[1] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[2] U. Manber. *Introduction to Algorithms, A Creative Approach.* Addison-Wesley, 1990.

[3] R. Sedgewick. *Algorithms in C.* Addison-Wesley, 1990.