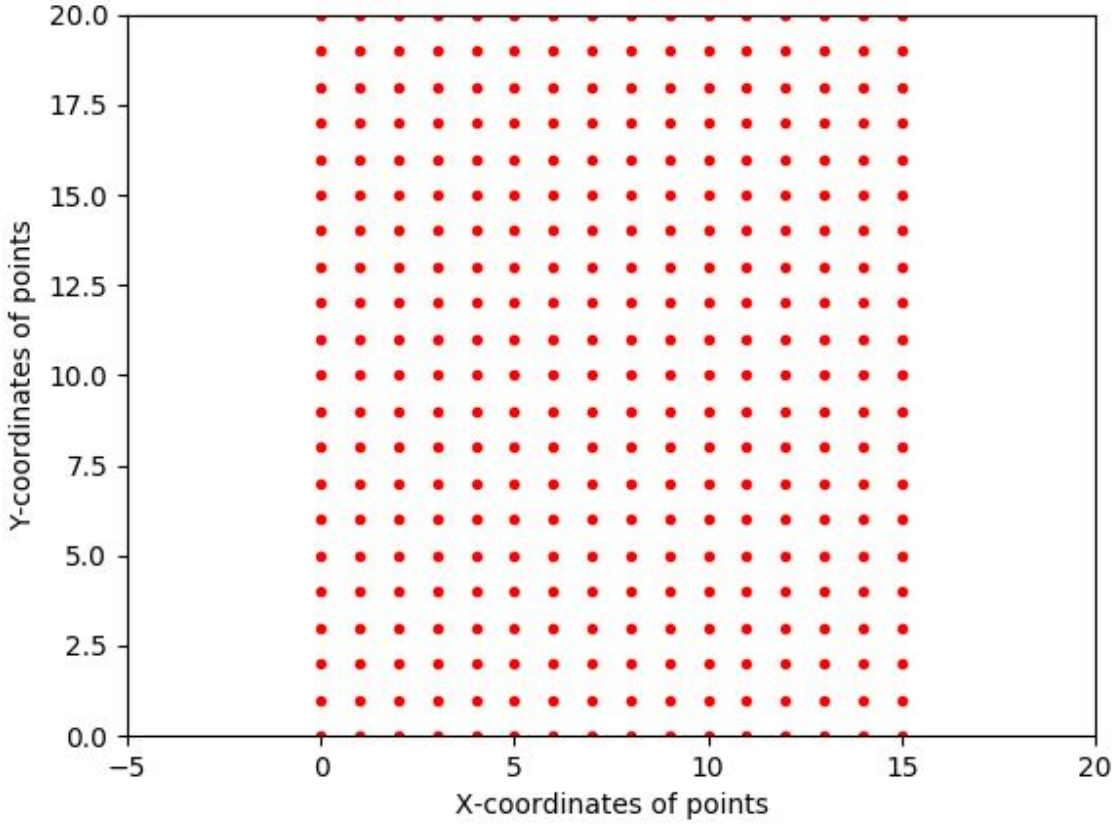
Generating Input for 2D Closest Pair Problem

Use of analytic methods to search for benchmark input

The strategy that we followed was to ensure that large number of points end up in the delta strip during the combine step of the algorithm . Here is a sample plot depicting the way we tried to generate the input points.



Here there are 16 different columns and each column has 20 points .

In general we have N columns and each column has K points . The number of columns is a power of 2 , this ensures that the left and the right halves after division have the same structure as the initial set of points . This works until we are left with a single column of points .

Now let us consider the first recursion step . The points get divided into two equal halves and the minimum distance in the left and right half is 1 . Now all the points for which $|(x - x_{median})| < 1$ will end up in the delta strip . According to the above distribution of points we will have K points in the strip . We will keep getting K points for each subsequent recursion

level until we are left with only a single column of points . After this the number of points in the delta strip gets halved with each increasing recursion level.

We tried with various values of N and K keeping the total number of points close to $5 * 10^5$ and calculated the time taken by the algorithm. Since the initial sorting of points is also $O(nlogn)$ we randomly shuffle the input points before running the algorithm.
This is the data we obtained for $N_{total} = 512000$ for my input.

Value of N	Value of K	Time taken(in seconds)	Number of distance calculations
16	32000	0.96	14747139
8	64000	0.97	15707139
4	128000	1.01	16603139
2	256000	1.02	17371139
1	512000	0.98	17883139

The maximum number of distance comparisons occurred in the case when $N = 1$ and $K = 512000$. When the input is a single column of points at each recursion level we get the maximum number of points in the delta strip that are possible at that step . However each point in the delta strip is compared only to the next point whereas in the worst case it is possible that each point gets compared to the next 3 points . Because of this we can't say whether our input achieves the maximum number of comparisons that are possible among all configurations of points.

Use of Gradient Descent to search for benchmark input

To find the input on which the algorithm's performance is poor, we formulated a loss function which can be used to assess any given input. Next, we formulated a gradient function which calculates the gradient of the loss function with respect to the input.

Loss function
The loss function is defined as $L(x) = c/M$. Here x is the set of input points , c is a constant and M is the number of distance comparisons made during the execution of the algorithm. Our aim is to maximize the number of distance comparisons and hence to minimize the loss function.

Gradient

To calculate the gradient of $L(x)$ at an input x where x=$\{x_1, y_1, x_2, y_2 ....x_{n-1}, y_{n-1}, x_n, y_n\}$ and (x_i, y_i) is a coordinate of i^{th} point, we calculated the partial derivatives with respect to each parameter. The gradient G is a vector of partial derivatives of loss function with respect to each parameter of input x. $G(x_i)$ is defined as:

$G(x_i) = (L(\{x_1, y_1, x_2, y_2, ..x_i + \Delta x_i, ...x_n, y_n\}) - L(\{x_1, y_1, x_2, y_2, ...x_n, y_n\}))/\Delta x_i$

We kept Δx_i to be in range 1 to 5.

Gradient Descent

We iterate in order to find the required input by following the update rule as $x = x - \alpha * G$. Here α is the learning rate. We kept α to be 100 since the gradients were of the order 10^{-3} and hence to make significant change in the input in every iteration α had to be kept large.

Observation

We ran gradient descent on random input and we observed that the loss function decreased and eventually converged. But since the loss function is not smooth and there exists many points of non-differentiability and local regions of zero gradient we are not sure if the converged points represent the overall minima. And also this process was very slow as we had to calculate the gradient with respect to all points in each iteration. For an initial input of random points , after applying gradient descent the points seemed to have converged to a straight line and hence we thought to start with the input suggested in the earlier part of the report as a starting point for this gradient descent learner. We observed that the loss function remained nearly constant even after a number of iterations which suggests that the input is one of the configurations which leads to minimum loss.

Results

We were able to achieve some significant difference in the time taken by the algorithm from the input that we generated .We tried to make sure that the coordinates of points in the random input and the input that we generated have the same value on an average.We measured the time using the CPU clock and the time measured fluctuated a bit even when the input was same. So we ran the algorithm several times to obtain the time values on an average. The time that we obtained on our input was 1.02 seconds and 17371139 number of distance comparisons. The random set of points resulted in an average of 0.83 seconds and 3010126 number distance comparisons . Thus we are able to achieve a difference of about 23% in the running time of the algorithm .