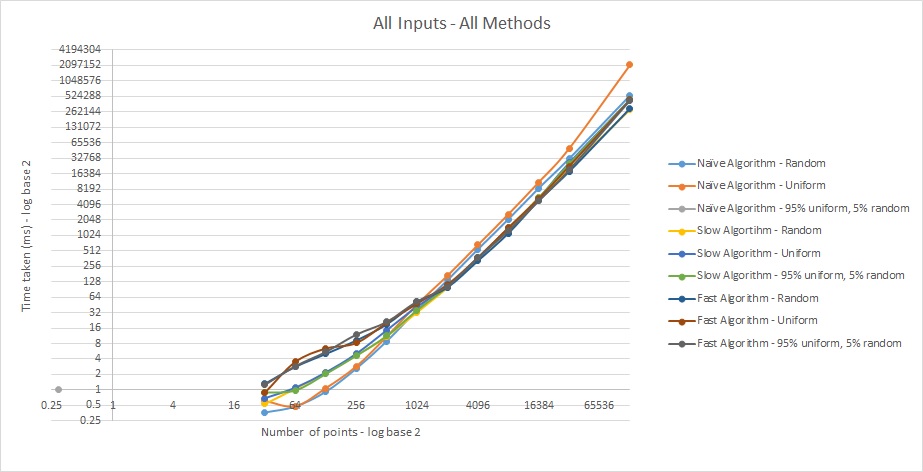
**Technical Explanation of Algorithms:**

Here is the table of slopes

|  |  |  |  |
| --- | --- | --- | --- |
| Slopes | Random Input | Uniform Points | Mixed |
| Naïve Algorithm | 1.932 | 1.864 | 1.546 |
| Slow Algorithm | 1.603 | 1.303 | 1.546 |
| Fast Algorithm | 0.9487 | 1.235 | 0.991 |

We observe that fast algorithm is almost 1, slow algorithm is around 1.5 and naïve algorithm is around 2. And these make sense because this is what we want since fast algorithm out performs slow algorithm and in turn slow algorithm outperforms naïve algorithm.

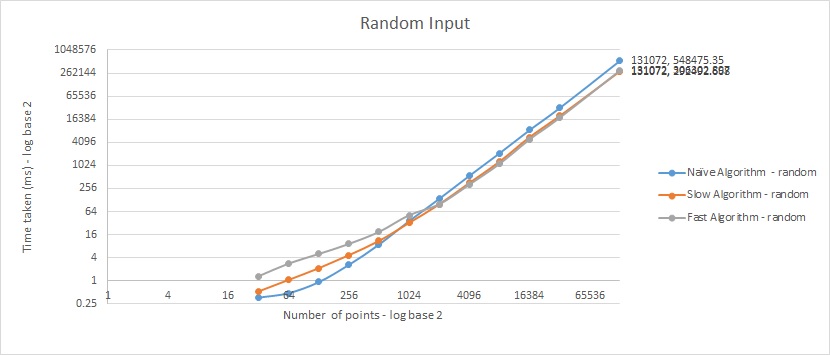
All methods and all inputs:



You can see that naïve algorithm takes more time compared to other algorithms , but naïve algorithms outperforms other algorithms for less number of points and input methods also effect the performance of the algorithms.

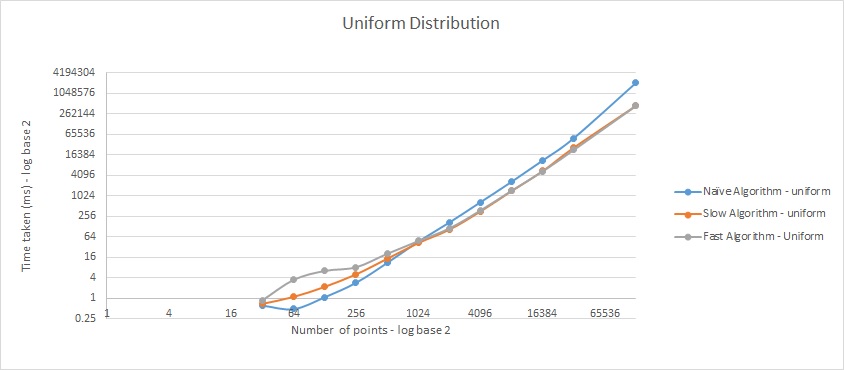
Here we take three inputs into account in 100 x 100 domain – random point generation, Uniform hexagonal points generation, mixed (95% uniform and 5% random).

1. Random Point generation



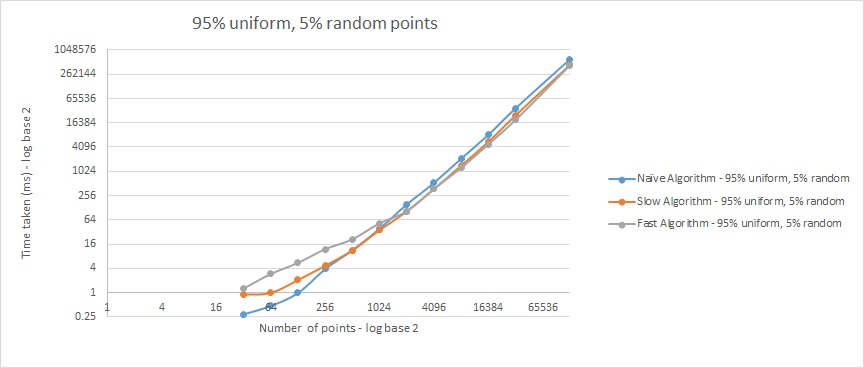
For low inputs naïve algorithm reforms better than both slow and fast algorithms, but after 2048 cross point they interchange and fast, slow algorithms perform better

Uniform Points :



For low inputs naïve algorithm reforms better than both slow and fast algorithms, but after 2048 cross point they interchange and fast, slow algorithms perform better

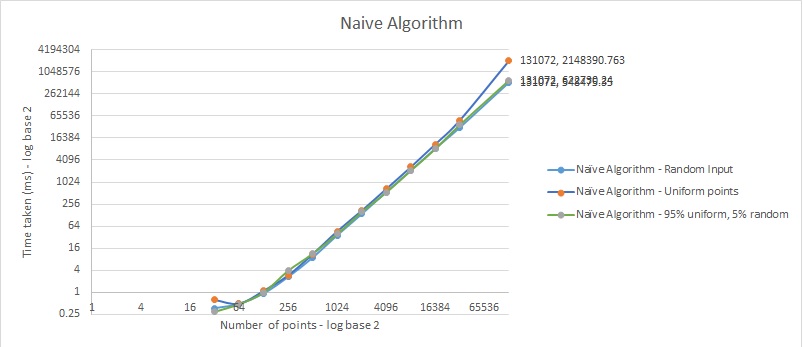
Mixed points:



For low inputs naïve algorithm reforms better than both slow and fast algorithms, but after 2048 cross point they interchange and fast, slow algorithms perform better

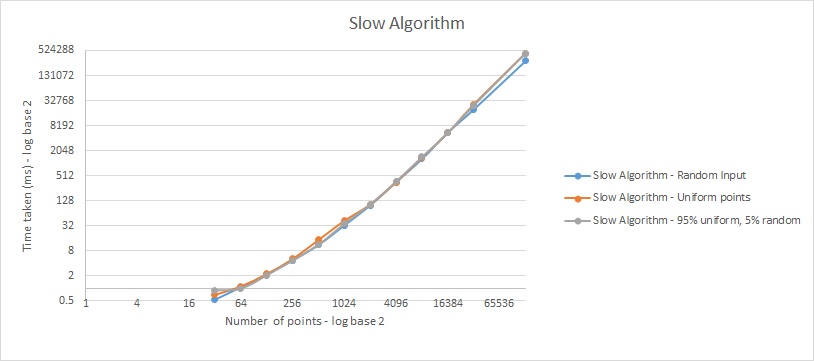
We use three methods

1. Naïve algorithm – O(n^2) algorithm checks each point with all others to find the minimum distance between them .



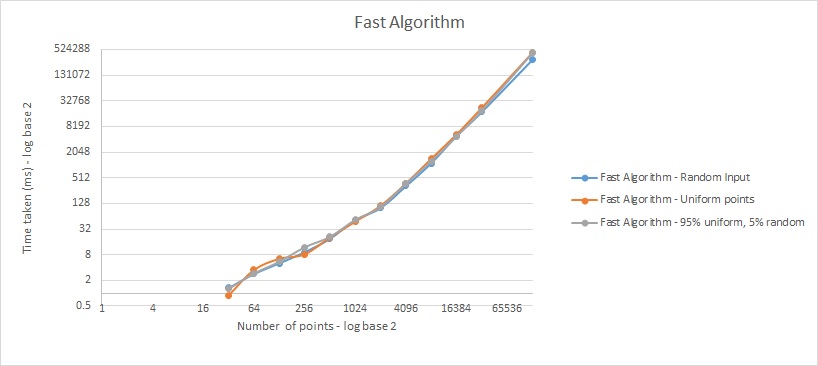
Naïve algorithms worst performance is for uniform points and works almost same for both random and mixed points

2. Slow algorithm – In this we sort the points according to the x axis and then by divide and conquer we find the minimum distance. With the minimum distance we get a slice of points with the given minimum distance and now we apply the naïve algorithm to get our desired output.



Slow algorithm work best for random input and performance almost same for both mixed and uniform.

3. Fast Algorithm - – In this we sort the points according to the x axis and then by divide and conquer we find the minimum distance. With the minimum distance we get a slice of points with the given minimum distance and now we sort the y axis in the slice and check to see each point is closer to other but here according to the geometric analysis each point has only 6 neighboring points so it will not be as slow as slow algorithm.



Fast algorithms works best for the random input and performs same for both uniform and mixed.

**Question 6 - Does the problem type affect the performance of the algorithms? Carefully explain why if there is a difference. If not, then explain why not. ?**

Naïve Algorithm :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Random Input |  |  | Uniform Input |  |  | Mixed input |
| No. points | Time (ms) |  | No. points | Time (ms) |  | No. points | Time (ms) |
| 4096 | 551.4732523 |  | 4096 | 668.4795 |  | 4096 | 532.0501 |
| 8192 | 2126.082877 |  | 8192 | 2653.185 |  | 8192 | 2124.602 |
| 16384 | 8569.329935 |  | 16384 | 11187.95 |  | 16384 | 8311.548 |
| 32768 | 32329.04292 |  | 32768 | 50791.44 |  | 32768 | 37743 |
| 131072 | 548475.35 |  | 131072 | 2148391 |  | 131072 | 622730.2 |

The average of the time taken for different inputs and as you can see it differs and it fact it takes more time for uniform as it had to check every point and all the values are more or less same to we need to update the mindistance values and other two are almost same.

Slow Algortihm

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Random Input |  |  | Uniform Input |  |  | Mixed input |
| No. points | Time (ms) |  | No. points | Time (ms) |  | No. points | Time (ms) |
| 16384 | 5602.73911 |  | 16384 | 5471.713 |  | 16384 | 5555.707 |
| 32768 | 20111.08983 |  | 32768 | 26604.32 |  | 32768 | 25235.03 |
| 131072 | 292492.668 |  | 131072 | 454944.8 |  | 131072 | 454479.9 |

Random inputs takes less time in slow algorithm also. And uniform input takes competitively more time. But we can observe both of them takes almost same time. – I think this is because in random input while we get the values in the minimum distance strip we will be left with less number of points in the strip and comparing is easy , similarly for mixed input (5% points may be near to hexagonal points and minimum distance is less ) so the strip size is also less so it performs okay and for uniform input we will be left with many points in the strip.

Fast algorithms

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Random Input |  |  | Uniform Input |  |  | Mixed input | |
| No. points | Time (ms) |  | No. points | Time (ms) |  | No. points | Time (ms) |
| 16384 | 4912.678413 |  | 16384 | 5383.667 |  | 16384 | 4844.993 |
| 32768 | 18115.25223 |  | 32768 | 22885.69 |  | 32768 | 19540.3 |
| 131072 | 306302.897 |  | 131072 | 451212 |  | 131072 | 440382.2 |

Random inputs takes less time in slow algorithm also. And uniform input takes competitively more time. But we can observe both of them takes almost same time. – I think this is because in random input while we get the values in the minimum distance strip we will be left with less number of points in the strip and comparing is easy , similarly for mixed input (5% points may be near to hexagonal points and minimum distance is less ) so the strip size is also less so it performs okay and for uniform input we will be left with many points in the strip. This may be better than slow algorithm but still behaves similay way, here we sort the y axis in the slice and check to see each point is closer to other but here according to the geometric analysis each point has only 6 neighboring points so it will not be as slow as slow algorithm.