**Brute Force inversions**

This does a compare one with all for all elements and thus is a simplistic way of calculating inversion with no logic involved. It is expected to take a longer time compared to divide and conquer as it revisits many of the already computed elements in every iteration.

For data1.txt

Running Time: 5996 ms

Inversion Count: 623897407

**Divide and Conquer Inversions**

This algorithm leverages the technique of merge sort to split the array and sort the sub arrays to efficiently compute the comparisons. The division of problems to subproblems of half the size reduces the number of comparisons needed in each step. The merge creates a superset of the sorted halves while computing the inversions at the same time based on the inversion rule A[i] > A[j] for i < j.

For data1.txt

Running Time: 12 ms

Inversion Count: 623897407

Comparison of the inversion algorithms: Theoretically, the inversion count in brute force is O(n^2) and that of the divide and conquer is O(nlogn). Comparing the running times the divide and conquer is 499.7 approximately 500 times faster than the brute force method. This is consistent with the theoretical analysis. The divide and conquer method is thus optimal in a sense that it computes correctly the 623897407 points in a fraction of time compared to brute force.

**Brute Force Convex Hull**

The convex hull brute force is again simplistic in a sense that it considers all possible pair of line segments with the set of points to determine if the remaining points for each line segment lie on the same side. Clearly this will take cubic time as there are n^2 line segments and n-2 possible points per line segments. This takes an exceptionally long time to compute results. Also, there are additional cases to check for collinearity which further hurts running time.

For data2.txt

Running Time: 23961 ms

Convex Hull Points: 24

**Divide and Conquer Convex Hull: Quick Hull**

The quick hull optimizes by drawing triangles from two extreme points that are part of the hull to another extreme point that lies in between the line segments between the two points. It then processes the remaining points by partitioning them in three splits caused by the triangle formed by the previous extremes and new extreme: between left most and newly found extreme lying outside the triangle, between rightmost lying outside triangle and discards the internal points in the triangle since they will never be a part of the convex hull set. Thus, it recursively computes the hull in each of the two partitions until there are no more points to process. The collinearity is naturally prevented in the removal phase by including the logic of any points inside and along edges of the triangle of the three points. This is a much smarter and more efficient way as the distance calculation steps are reduced in each recursive step alongside reducing the subproblem sizes. The result is a much more optimized convex hull construction strategy as seen in the results below:

For data2.txt

Running Time: 22ms

Convex Hull Points: 24

**Comparison of Brute Force and Divide and Conquer Convex Hull Algorithms**: Theoretically, the brute force has O(n^3) complexity as opposed to the divide and conquer having O(nlogn) complexity. The running time is also drastically different which is consistent to the theory. The divide and conquer (22ms) is 1089.14 times faster than brute force (23961ms). This shows that the quickhull is optimal in the fact that it correctly calculates the 24 points and in a fraction of time compared to brute force.