Brian Lee
3620101

CMPSC 130B Programming Assignment 1 Report

a) The brute-force approach works regardless of whether or not the array is sorted. Initialize the minimum distance value to the maximum possible double value. Then, for every point in the array except the last point, calculate the distance between that point and each of the points that come after it in the array. If that distance is less than the current value of the minimum distance variable, set the minimum distance variable to the new distance. This method is guaranteed to find the smallest possible distance between two points in an array of points with O(n^2) complexity.

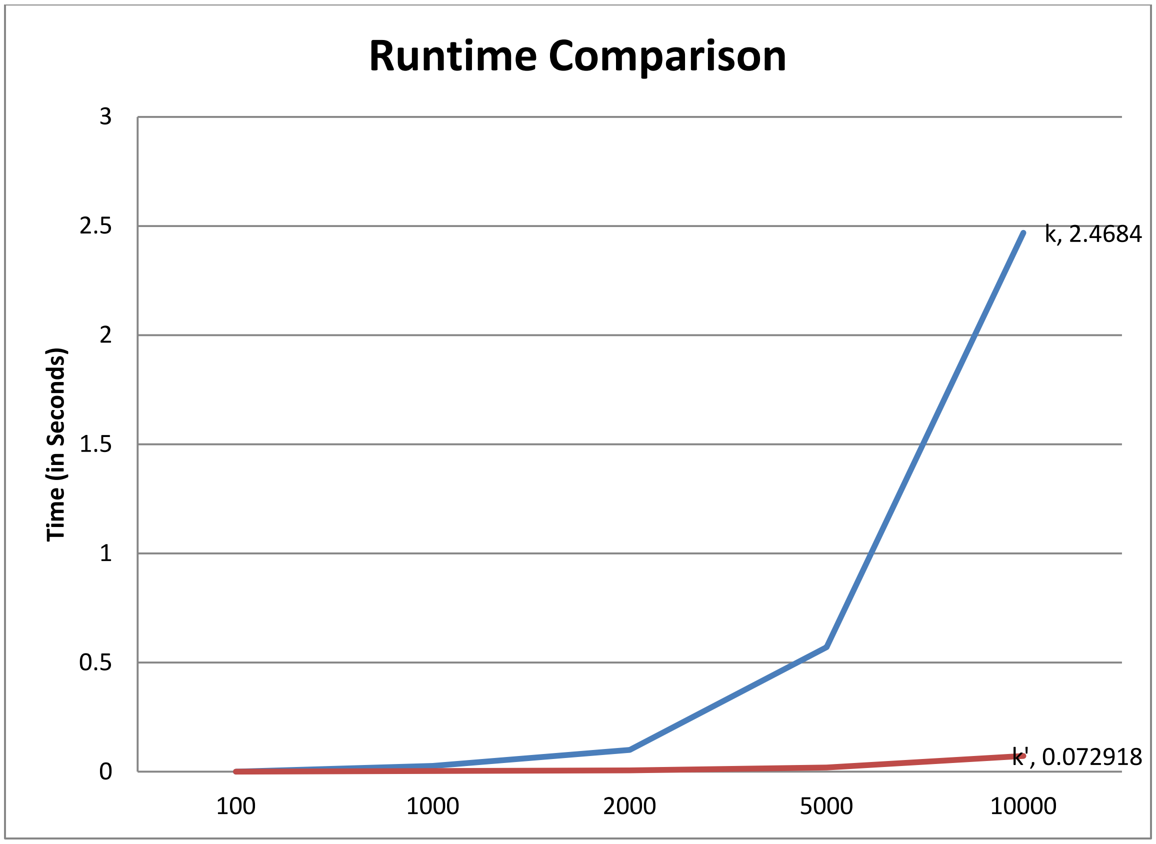
```
bruteForce(Point* arr[], int size)
{
    for(i = 0; i < size; ++i)
    {
        for(int j = i + 1; j < size; ++j
        {
            minDist = min(minDist, Distance(arr[i], arr[j]);
        }
    }
}
```

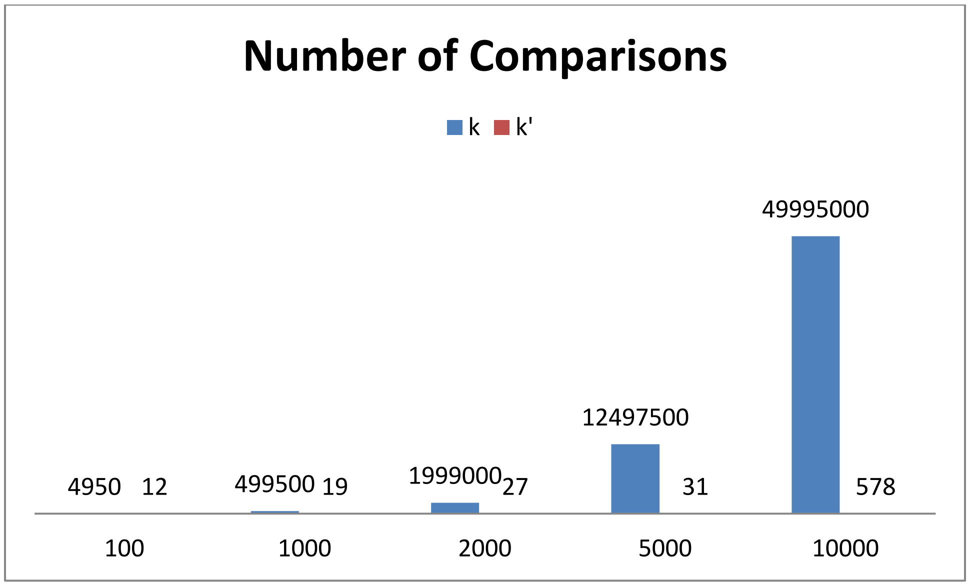
b) The divide-and-conquer approach involves taking an array of points sorted by their x coordinates and dividing them in two before recursively finding the smallest distances of the sub arrays. Once we find the minimum distance between the two sub-arrays, we find all points that are, at most, the current minimum distance away from the center of the full array and place them in another sub-array. Once the smallest distance of the sub-array has been found, check if it's smaller than the current minimum distance. Depending on the implementation, this will result in either an O($n \log(n)$) or O($n \log^2(n)$ complexity.

```
dnC(vector<Point> list)
{
    if(list.size() < 3)
        return Brute(list, list.size());
    middle = list.size()/2;
```

```
        midPoint = list[middle];
        vector<Point> left = left of midPoint;
        vector<Point> right = right of midPoint;
        minDist = min(dnC(left), dnC(right));
        vector<Point> center;
        for(int i = 0; i < list.size(); ++i)
        {
                if(distance(list.at(i)) < minDist)
                        center.push_back(list.at(i));
        }
        return min(minDist, dnC(center));
}
```

Plots


Runtime Comparison
Time (in Seconds)
3
2.5
2
1.5
1
0.5
0
100
1000
2000
5000
10000
k, 2.4684
k', 0.072918


Number of Comparisons
k
k'
4950
12
499500
19
1999000
27
12497500
31
49995000
578
100
1000
2000
5000
10000