Arya Asgari
Michael Elliott

Implementation 1 Write-up

Pseudocode:

Brute force:

```
func bruteforce(points):
    min_dist = dist(points[0], points[1])
    for outer_index from 0 to num(points) – 1:
        for inner_index from outer_index to num(points):
            min_dist = minimum(min_dist, dist(points[i], points[j])
    return min_dist
```

Naive Divide and Conquer:

```
func divideandconquer(points):
    if num(points) <= 3:
        return bruteforce(points)
    left_half = points[0 : num(points) / 2]
    right_half = points[num(points) / 2 : num(points)]
    left_min_dist = divideandconquer(left_half)
    right_min_dist = divideandconquer(right_half)
    min_dist = minimum(left_min_dist, right_min_dist)
    between_min_dist = closest_cross_pairs(
        {point : point ∈ points,
         x_coord(median(points)) - min_dist
        < x_coord(point)
        < x_coord(median(points)) + min_dist})
    return minimum(min_dist, between_min_dist)
```

Enhanced Divide and Conquer:

```
    enhanceddnc(lst, ysortedlst):
        If n <= 3
            Compute and return the minimum distance
        Else
            Compute separation line L at median x coord
            Break lst into two halves sorted by X (lhalf, rhalf)
            For p in ysortedlist
                If p x-coordinate <= median
                    lyhalf.append(p)
                Else
                    ryhalf.append(p)
```

```
D1= closest-pair(left half, lyhalf)
D2=closest-pair(right half, ryhalf)
D=min(d1,d2)
M = [medianx - d, medianx + d]
Sorted_m = sortbyY(M)
Dm = closest-cross-pair(sorted_m, D)
Return Dm
```

Asymptotic Analysis of Runtime

Brute Force: O(n^2)

Naive Divide and Conquer:

Runtime: O($n\ log^2 n$)

Recurrence Relation: $T(n) = 2T(n/2) + cnlogn$

Solve:

$cn(log(n) + log(n/2) + log(n/4) + ... + log(n/2^k))$

$cn(log^2 n) - cn(log2^{1+2+3...+k})$

$cn(log^2 n) - cn(1+2+3+... + logn)$

$cn(log^2 n) - cn(\frac{logn(logn+1)}{2})$

= O($n\ log^2\ n$)

Enhanced Divide and Conquer:

Runtime: O(n logn)

Recurrence Relation: T(n) = 2T(n/2) + cn

Solve:

$T(n) = 2T(n/2) + cn$

$T(n) = 2T(n/2) + O(n)$

Using master theorem: $T(n) = aT([n/b]) + O(n^d)$

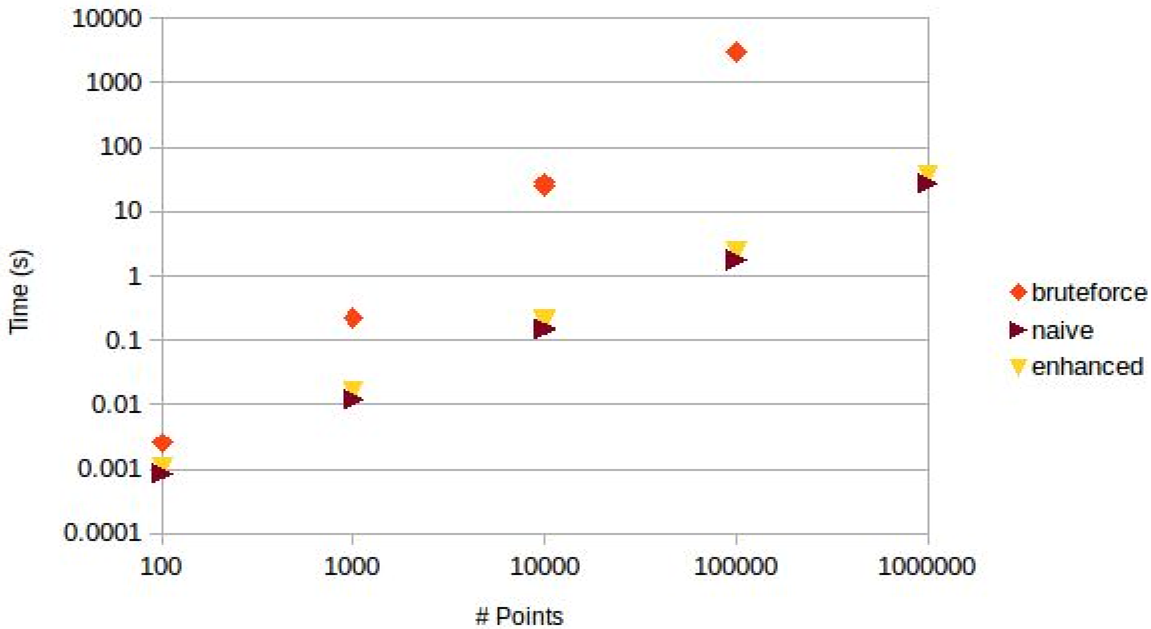
$a = 2,\ b = 2,\ d = 1$

Since $d = 1$ and $log_a b = log_2 2 = 1$

Then $d = log_a b$ which means that $T(n) = O(n^d logn)$

Therefore, $T(n) = O(nlogn)$

Plotting the runtime:



Discussion:

For our plot, the growth curves in some areas definitely did not match our expectations based on the theoretical bounds. The brute force growth curve went as expected. It performed 10x worse on 10,000 points than the others did on 1,000,000. However, we were surprised to find that our naive divide and conquer was slightly faster than our enhanced version. The asymptotic complexity of the enhanced divide and conquer proves that it should be faster than the naive version. There are a few reasons why we believe that the opposite occurred within our implementation. Specifically, it may be in the way we utilize the list of coordinates that is sorted by y in the enhanced version. It seems that searching through the list of points sorted by y coordinate takes longer than sorting it each iteration.