Programming Assignment 1

Troy Timmerman

September 18, 2023

Natural Language Description

First, the algorithm sorts the *Points* array using MERGE-SORT. Then, we initialize an array of length equal to m, or the number of minima that the user requests.

Next, in $\Omega(n)$ time, we iterate once through points calculating the Manhattan Distance of each point, and the point one index value larger in the array.

For each distance, we test if it is less than the largest minimum in A. If it is, we replace $A[-1]$ with that distance. Then we MERGE-SORT A again to make sure $A[-1]$ is the largest value in the array.

Because the array is sorted by x, we should have pretty close to the minimum values in our array. However, consider the case where we have points $(1, 2), (1, 200), (1, 3)$. The distance between the first and second point is 198, but the point between the first and third point is only 2. If we want to make sure our response has the lowest Manhattan Distance, we will need to make sure that we do not omit distances such as point 1 and 3.

To do this, we can use the largest minimum value in our array of minima. Since Manhattan distance is the distance between x values plus the distance between y values, unless a point has an x value that is as closer to a point than the largest minimum, it will be too large and can be exluded.

Starting with the first index value in the array of all points, we will index a subarray from that point until x is greater than the largest minimum. Then we will run the brute force method on this subarray. The brute force method calculates the distance between each point and selects the smallest minima from those points.

Each time, we will calculate m values from the brute force calculation and compare those values to the largest minima, replacing and resorting the list if it is smaller.

After each run of merge sort, we can move to the largest index that has been compared, and select the next points where the distance between that index and those points' x-values are less than the largest minimum.

Once the last index value has been reached, the minimum array should contain the m smallest minima. However, if there are ties, then some of the smallest values may be excluded.

Pseudocode

ITERATIVE-MINIMUM-DISTANCE($Points, m$)

```
1  Points = MERGE-SORT(Points)// CLRS page 39
2  A = [(∞, (0, 0), (0, 0) for 1 : m]
3  for i = 1 to Points.length − 1
4       d = MANHATTAN-DISTANCE(Points[i], Points[i + 1])
5       if d < A[−1]
6            A[−1] = (d, Points[i], Points[i + 1])
7            A = MERGE-SORT(A)// CLRS pg 39
8            // n.b. for this and all other uses, sort based on the first element in the tuple
9  i = 1
10 while i < A.length
11      increment = 1
12      while Points[i + increment].x − Points[i].x < A[−1]
13           increment = increment + 1
14           if increment > 1
15                minima = BRUTE-FORCE-MINIMUM-DISTANCE(Points[i : i + increment], A.length)
16                for j = 1 to minima.length
17                     if minima[j] < A[−1]
18                          for k = 1 to A.length
19                               if minima[j] = A[k]
20                                    break
21                          A[−1] = minima[j]
22                          A = MERGE-SORT(A)// CLRS pg 39
23      i = i + increment
24 return A
```

```
BRUTE-FORCE-MINIMUM-DISTANCE(Points, m)
1  A = [] // Initialize an empty array
2  for i = 1 to Points.length
3       p1 = Points[i]
4       for j = 2 to Points.length
5            p2 = Points[j]
6            if (p1 ≠ p2) ∧ (i < j)
7                 d = MANHATTAN-DISTANCE(p1, p2)
8                 A.append((d, p1, p2))
9  A = MERGE-SORT(A) // CLRS page 39
10 return A[1 : m]

MANHATTAN-DISTANCE(point1, point2)
1  d = |point1.x − point2.x| + |point1.y − point2.y|
2  return d
```

Worst Case Running Time

Because the algorithm uses the brute force algorithm, the worst case is $O(n^2)$. This occurs when all points in the array have the same x value. In this case, the subarray will consist of the entire array. All points will be compared.
This leads to n time for the first iteration through the points, and $n-k$ for each iteration k through the array again comparing points. This gives $O(n \cdot (n-k))$ and can be simplified to $O(n^2)$.
However, this is an unlikely composition of the array. Normally, the brute force method should have a small number of elements and take only a small part of runtime. In this case, it would take $O(n \lg n)$ time to sort the points array. Iterating through the array and calculating the distance in lines 3-7 would take n time. Assuming the worst case, that each set of points has a smaller and smaller distance, merge sort would happen n times. That would equal $n \cdot m \lg m$. If the second iteration has small subarrays, the time of that iteration should be close to n as well.
This would put the average runtime somewhere around $\theta(n \lg n + n \cdot m \lg m + n$, which is close to $\Theta(2 \cdot n)$ or $\Theta(n)$.

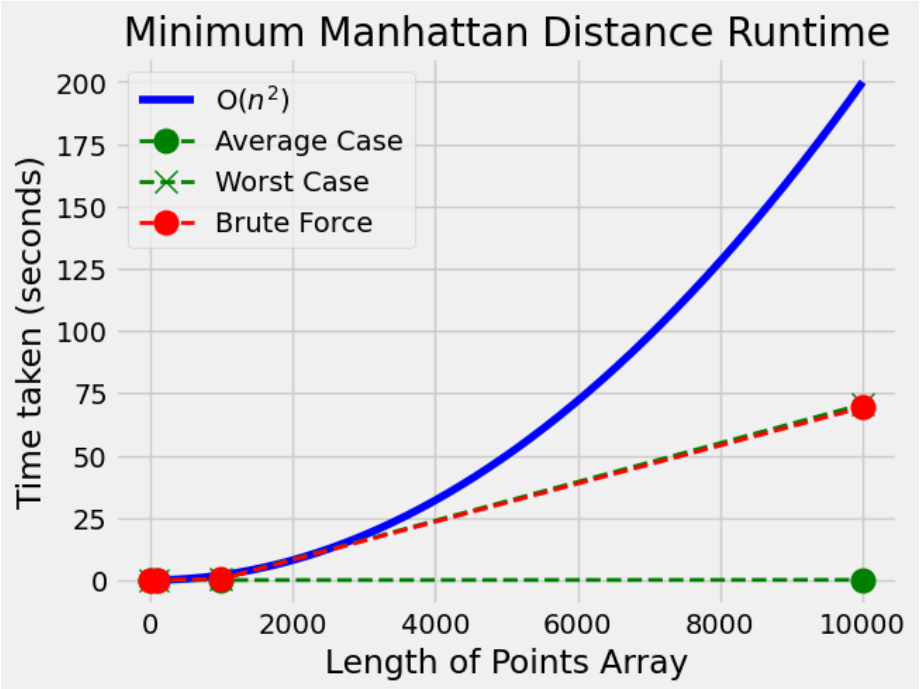
Trace Runs

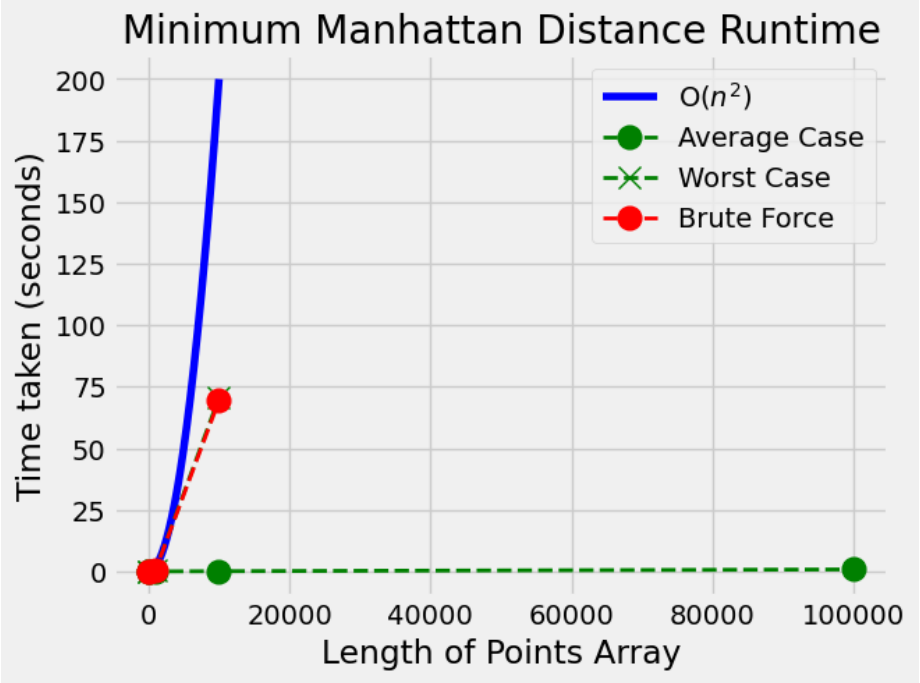
The trace run is demonstrated in /results/manhattan-distance-trace.log.

Asymptotic Behavior Runs

Average runtimes are in /results/manhattan-distance-runtime.log

Analysis


Minimum Manhattan Distance Runtime
O(n^2)
Average Case
Worst Case
Brute Force
Time taken (seconds)
200
175
150
125
100
75
50
25
0
0
2000
4000
6000
8000
10000
Length of Points Array


Minimum Manhattan Distance Runtime
O(n^2)
Average Case
Worst Case
Brute Force
Time taken (seconds)
200
175
150
125
100
75
50
25
0
0
20000
40000
60000
80000
100000
Length of Points Array

The average time of the algorithm is much faster than the $O(n^2)$ worst case runtime. However, when the input all has the same x-value, the worst case becomes apparent, and the runtime is identical to the brute-force algorithm.

Retrospection

There are several possibilities to improve my algorithm. The biggest way, is to remove the worst case of being $O(n^2)$ when the entire sub-array is brute-forced as in the case when each point has the same x-value. To do this, after creating a subarray based on x distance, we could filter the array again based on y distance in the same manner. This would ensure that there is not a case where all points would be brute-forced at once.
Secondly, in order to reduce runtime further, I could take advantage of the already sorted nature of A. Instead of replacing the last index and resorting, I could insert the point and distance in the list where it fits and remove the largest element.

Works Cited

digydigy, et al. “How to Log Source File Name and Line Number in Python.” Stack Overflow, 1 June 1955, stackoverflow.com/questions/533048/how-to-log-source-file-name-and-line-number-in-python.

“Timing Functions With Decorators - Python.” GeeksforGeeks, GeeksforGeeks, 5 Apr. 2021, www.geeksforgeeks.org/timing-functions-with-decorators-python/.