Assignment 2

Erik Norlin, CID: norliner

September 18, 2023

Problem 3

a)

This algorithm needs to run in cubic time because we have n^2 pairs of houses (including duplicates and pairs with themselves) and we also have to compute all distances between all pairs. To compute all distances between all houses, we *must* include the longest distance which is a computation of $n-1$ iterations, i.e., $(\sum_{i=1}^{n-1} x_i)$. If we do not compute the same distance twice between a chosen pair, the minimum running time for this algorithm becomes $T(n) \approx \frac{1}{2}(n-1)n^2 \in \Omega(n^3)$. On the other hand, if the algorithm computes the same distance twice between a chosen pair the maximum running time becomes $T(n) = (n-1)n^2 \in O(n^3)$. This implies that the upper and the lower bound of the running time are of the same magnitude, showing that the algorithm runs in a tight bound in $\Theta(n^3)$; the algorithm runs no longer or shorter than cubic time.

Pseudo code for this algorithm

```
D = [0] # n*n array to contain computed distances
for i = 1,...,n
    for j = 1,...,n
        if i < j
            for k = i,...,j-1
                D[i,j] += x_k # x_k is the distance from H_k to H_(k+1)
            D[j,i] = D[i,j] # Making array symmetric
```

The number of computations of this algorithm is $\approx \frac{1}{2}(n-1)n^2$, $\frac{1}{2}$ since we only compute the upper diagonal of the array. As argued above, this algorithm runs in $\Theta(n^3)$.

b)

A more clever algorithm in this case that runs faster than cubic time is to compute the distance between two houses only once. In the naive algorithm we computed the distance between two houses from scratch for every pair. However, we can utilize the distances already computed. This would result in a quadratic running time since there are n^2 number of pairs.

Pseudo code for this algorithm

```
D = [0] # n*n array to contain computed distances
for i = 1,...,n
    for j = 1,...,n-1
        if i < j
            D[i,j+1] = D[i,j] + x_j # x_j is the distance from H_j to H_(j+1)
            D[j,i] = D[i,j+1] # Making array symmetric
```

This algorithm make $\approx \frac{n^2}{2}$ computations which $\in O(n^2)$.

c)

It is not possible to get down the running time further since the running time cannot be of lower order than the input size which is n^2 pairs in our case; we must touch each element once at minimum.