Foundations of Algorithms Fall 2024
Programming Assignment 1

Rong Fan

1. 2.a.i Consider a set, P , of n points, $(x_1, y_1), ..., (x_n, y_n)$, in a two dimensional plane. A metric for the distance between two points (x_i, y_i) and (x_j, y_j)in this plane is the Euclidean distance $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. Construct an algorithm for finding the $m \leq \binom{n}{2}$ closest pairs of points in P . Your algorithm inputs are P and m. Return the distances between the m closest pairs of points, including their x and y coordinates.

Algorithm 1. Closest m Pairs Problem

Input: array of pairs P of length n with each pair P_i contains two floats, and an integer m, $m \leq n(n-1)/2$
Output: array of triplets T containing the about closest m pairs of points, the elements in the triplets are: a float indicating the distance, an int indicating the index of the first point and another int indicating the index of the second point.

The idea is to find the distance of each pair of points and store the triplet in a max-heap. We can keep the heap with maximum length m, and keep the pairs with the smallest distance, so at the end of the operation, the heap will be the output.

```
1: function CLOSESTMPAIRS(P, m)
2:     max-heap M                          ▷ stores the triplets and sorted by the first element in the triplet
3:     for i ← 1 to n − 1 do
4:         for j ← i to n do
5:             calculate the distance d between P[i] and P[j]      ▷ Will create a function to do this
6:             Insert {d, i, j} into M                             ▷ Insert to a heap takes Θ(lgm)
7:             if size(M) > m then
8:                 pop M                                           ▷ Popping a heap takes Θ(lgm)
9:     return M
```

2. 2.a.ii We analyze the time complexity of this algorithm. We use a nested for loop to iterate through the input array of length n, and within each iteration, we perform at most an insert and a pop on a max-heap of size m. Therefore the time complexity of this algorithm is $\Theta(n^2 lgm)$. This is the worst case run time as no matter how the input change, the run time would not be different.

3. 2.c trace runs submitted as screen shots

4. 2.d test included in a screenshot of the chart storing the runtime of different input and output sizes

5. 2.e According to the run time table I provided, we see that the running time of our program aligns pretty well with the expected run time. The runtime grows linearly with respect to lgm and with respect to n^2 except when m is really large, where the time increases by a lot. Since we are implementing with a heap of size m, on potential reason for this drastic increase is we are running out of fast memory in our environment and we would have to use virtual memory which is slower.

6. 3.a Right now the worst case time complexity is $\Theta(n^2 lgm)$. We know that $m \leq \binom{n}{2}$, which means $m = \Theta(n^2)$. Therefore, the worst case time complexity is actually $\Theta(n^2 lg^2 n)$. However, if m is relatively small (a magnitude smaller than n), there is probably a way to do the entire process more efficiently, since we can find the closest pair in $\Theta(nlgn)$ time using a divide and conquer algorithm. Simply repeating it m times would make the time complexity $\Theta(nmlogn)$ which is more efficient. If we can find some smart way merging the points while keeping the m closest pairs, maybe there could be an algorithm running at $\Theta(nlog(mn))$.