**Team: G6T18**

**Members: Tan Chin Hoong**

For Question 1, I first started by looping through the flags dictionary, which I changed whenever I added any flags. From that flags dictionary, I looped through and got a POINTS/DISTANCE ratio. I then took the max of all the best ratios, and appended the flag's ID which I got from the max function. After this, I used a two opt to sort my array. Inside my twoOpt iterations, I used the swapSides function, which has the complexity of o(N), as it is just copying the array elements. I then used the get_dist_and_points_q1 ( O(N) )from the utility.py to get the distance of the new array. I then checked if the  a complexity of new distance was smaller than the previous best distance, and if it is, I updated the array. This was done till I looped through the entire array. This entire iteration of TwoOpt, resulted in a complexity of O(N^3) as within the 2 for loops, I had a swapSides and get_dist_and_points, which ran concurrently. After getting the bestRoute, I then did a loop to remove the flags that result in having extra points, to bring down the distance as flags can be deducted. This function sorts the route first and finds the flags which have said points. I then did a single for loop, in which I did a list comprehension (O(N)) and a get_dist_and_points(O(N)). If removal of the flag resulted in a better distance, I then updated the bestRoute. This resulted in this function having a complexity of O(N^2).

**Overall complexity of the entire q1 : O(N^3)**

For Question 2, if the points was low or the number of player was 1, I would use the code that I used for the previous question. The complexity for this scenario was O(N^3).

Else,  I started by first doing a K means clustering to get the number of clusters based on the elbow method and kmeans. This has a complexity that is based on the number of centroids gotten – O(N^K) where k is the number of centroids. Looping through the flags, I then find the best flag that will allow me to get the best distance ratio from each point in the holder. After this, I will do a limited two Opt the holder array. This for loop has a worse case complexity of O(N^2), as each cluster is loop through with the entire flags list. Each flags list then produces a candidate flag. The cluster dictionary is then reset to prevent previous points from being added.

The key idea is to allow each player to go to only points in that cluster. Comparing each next point from each cluster to find the best candidate and then appending it to that cluster. This allows us to get a final array that results in points from each clusters. Each cluster is then optimised via a limited two opt.

**Overall complexity of q2 : max ( O(N^3) , max(O(N^k), O(N^2)) )**