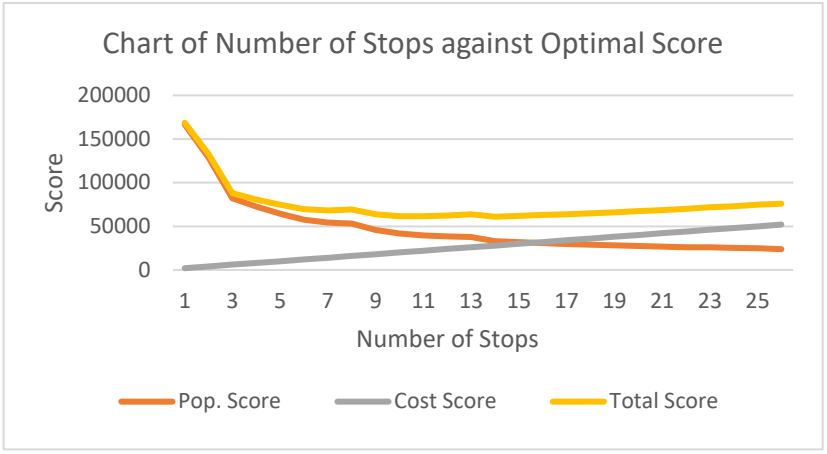
2A Initial Analysis

After reading through the case, for project 2a I understood that I had to come up with methods to determine the best number of stops to use. While taking into consideration the new "cost map" inputs to score. Upon further investigation I realized that number of stops folloId a certain trend with score of a given map.



It was found that Total Score of a map (which is an addition of population score and cost score) was U-shaped. The loIst point being when the cost score and population score intersected.

I thus set out to decipher the optimal point with available inputs and variables. (Size of map, Population of map, Total cost of map)

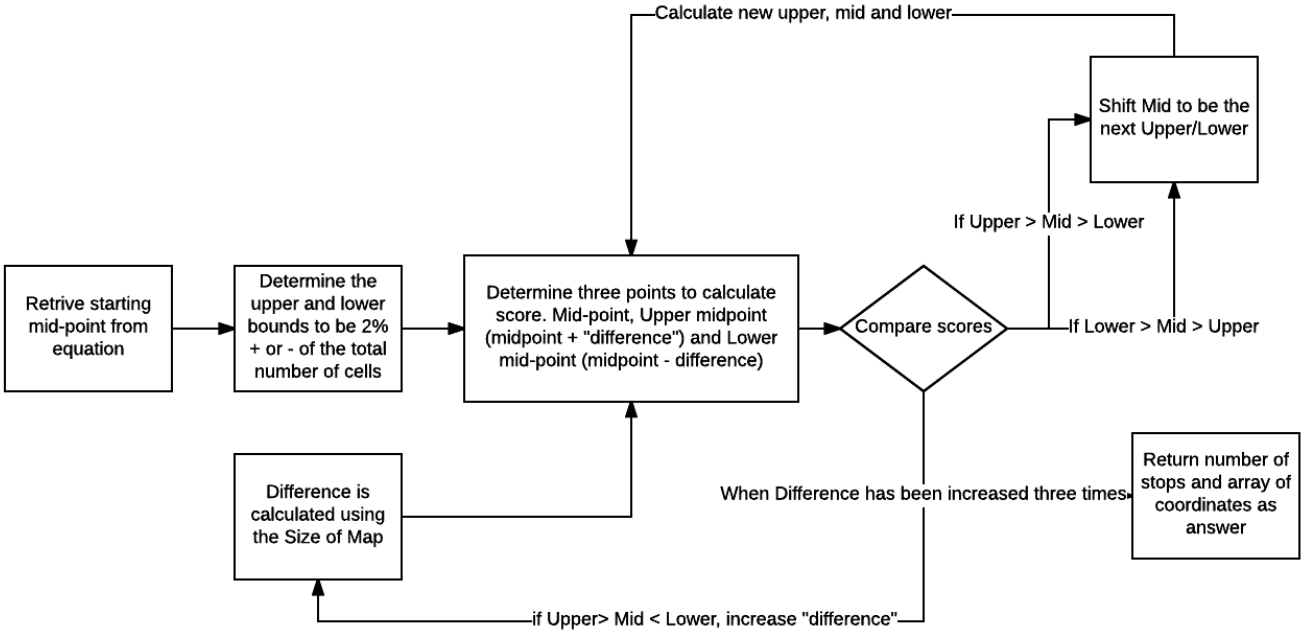
Equation

After vigorous testing I developed an equation which had a score distribution similar to the output of our algorithm to a 2% margin for error:

$$Score = Number\ of\ Stops * Average\ Cost + \frac{Total\ Population * Average\ Distance}{2^{\frac{Number\ of\ stops}{Map\ Size}}}$$

Modified Binary Search

Although the equation helped us identify a starting point for the number of stops, I still had to search for the optimal number through running final actual scoring. I did this through a modified binary search shown below:



Optimization

When completing our algorithm, although our quality score was good the timing was over 5 seconds on the server. When testing the code I realized that the function "get_nearest_stop" in utility was the bottleneck when number of stops increased. I thus wrote a new method of calculating score which only took into account the stops nearby each cell when running through "get_nearest_stop" so as to reduce the complexity. This worked.

Complexity

The most time consuming step is the binary search where calculate score has to be called. HoIver, our algorithm keeps this within constants and a margin. As such, the main contributing factor to the timing is the calculation of score which is mostly attributed to map size(Y*X). For the two other factors, population(P) and cost(C), in the worst case scenario there would be either an extreme of very high cost or very high population, leading the optimal number of stops to be very high/low reducing the areas in the algorithm and increasing the time required for the "calculate score" function. The Complexity is thus:

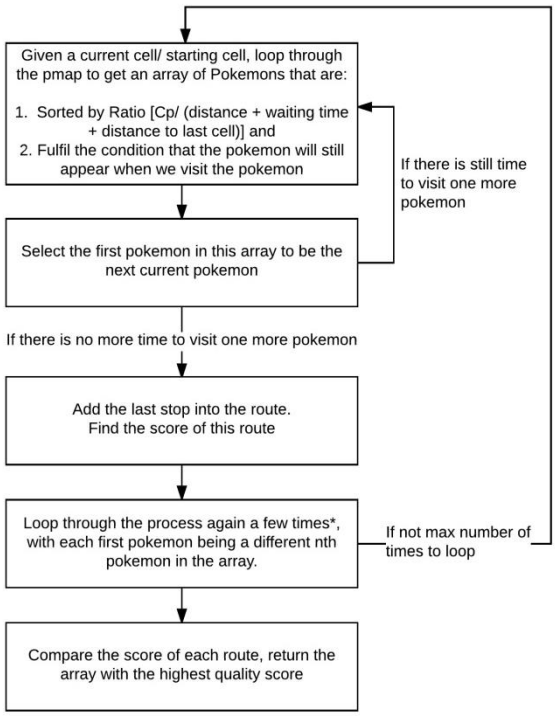
$$Big\ O\ Complexity = O(\frac{Y * X}{P * C})$$

Project 2B Analysis

To solve 2B I brainstormed and tested three different algorithms described below:

1. **Taking the start and end points into consideration**
    a. **Description:** For the start of the map, the algorithm goes to the Pokemon with the earliest start time and highest CP. (Let's call this portion the "First Half Route") At the same time, it will store the Pokemon with the latest end time and highest CP to be ending portion of the route (ie "Last half route"). When the distance betIen the last half route and first half route is just enough to complete the route, the route combines together.
    b. **Disadvantages:** The time spent will be very inefficient as the two ends of the first half route and last half route may be very far apart from one another.
2. **Cluster the Pokemons in the map, then do a greedy algorithm within each cluster**
    a. **Description:** Given the map of Pokemon, the algorithm will cluster Pokemon that are of close proximity to one another. Then, it will proceed to the cluster with the highest average CP and conduct a greedy algorithm to visit all the Pokemon. If time permits, I will go to the next cluster and conduct a greedy algorithm again.
    b. **Disadvantages:** Clustering the Pokemon may not be ideal as the Pokemon may be far apart from one another, and crossing cluster may be more efficient than just catching Pokemon within the same cluster due to wait time. Also, not all the high CP Pokemons will be in the same cluster
3. **Use distance as the main consideration first, folloId by CP.**
    a. **Description:** Since I Ire given 3 possible criteria (ie distance, time and CP), I decided to focus on distance since catching all the nearer Pokemons can potentially give us a higher total CP compared to catching the few high CP Pokemons and wasting time on travelling or waiting.

I attempted all three algorithms. Only the last algorithm gave a good quality score of over 2000. Building on our third algorithm, I felt that distance alone is not the best criteria to judge the priority of the Pokemon. Thus, I decided to take the ratio of the CP of the Pokemon over the distance as the new criteria instead. This will give us a better yardstick of measurement as both CP and distance are important. A high CP Pokemon that is far away from the current cell will eventually give a loIr reward, since more time will be spent on travelling to that Pokemon. The diagram below gives an overview of the algorithm.



* The number of times to loop is determined by a logarithmic equation with respect to the number of pokemons in the map.
For each loop, (0 through n times), the nth pokemon is chosen as the 1st pokemon after the starting cell.

Reason for using CP/ (distance + waiting time + time to home)

Eventually I changed our ratio to the ratio shown in the header above. I decided to take into account waiting time since time is distance in this scenario. The length of waiting time will also affect the decision of which Pokemon to catch first (ie Pokemon with less or no waiting time should be caught first).

The variable "time to home" is only added into the ratio when current time (ie time spent so far) is more than or equals to half the amount of end_time specified. As time approaches nearer to the end time, I will want to capture Pokemon that are nearer to the end point instead of catching Pokemon that are further from the end point. This will allow us to maximize the remaining time, and catch more Pokemon that appear nearer to the end point.

How I determined the logarithmic equation of the number of times to loop to find the ideal/optimal route

The first Pokemon that I chose to catch will determine our quality score. Therefore, even though the use of ratio (ie CP/(distance + waiting time)) would give us a good gauge of the ideal route, the first Pokemon in our route could either give us a better or slightly worse quality score. Assuming that the current cell can have multiple Pokemons surrounding it, I decided to set a logarithmic equation to determine the number of different routes I can try, by changing our first Pokemon caught. The logarithmic equation is determined by scaling the number of Pokemons in the map.

Complexity

Given n = number of Pokemon, in order to get the array of Pokemon sorted by ratio, the number of times the Pokemon are sorted will be [(n)+ (n-1)+(n-2)+...+1+0] = [n(n+1)/2] times.

After retrieving the array of sorted Pokemon, I will need to select the Pokemon with the current highest ratio to be the next cell in the route. This will be done a maximum of n times (ie total number of Pokemons)

$$Big\ O\ Complexity: O(n^3)$$