```
pragma solidity ^0.5.17;

contract test {
    // structure for charging station data
    struct station{
        int16 id;
        int16 x;
        int16 y;
        int16 price;
        int16 visits;
        int16 chrg_rem;
        int16 rating;
    }
    // structure for calculations of station and vehicle with respect to vehicle
    struct aftermath{
        int16 id;
        int16 x;
        int16 y;
        int16 price;
        int16 visits;
        int16 distance;
        int16 chrg_rem;
        int16 rating;
        int16 direction;
    }
    // structure to store optimised data
    struct optimised{
        int16 id;
        int16 x;
        int16 y;
        int price;
        int16 visits;
        int distance;
        int chrg_rem;
        int rating;
        int16 direction;
        int score;
    }
    // structure to store vehicle data
    struct vehicle{
        int16 x;
        int16 y;
        int16 battery_perc;
        int16 mileage;
        int16 dir;
    }

    // array of station structure
    station[] public arr;

    // array of aftermath structure
    aftermath[] public calculated;

    // array for unoptimised recommendation
    aftermath[] public unopt_possible;

    // array for optimised recommendation
```

```
    optimised[] public opt_possible;

    // vehicle object
    vehicle public veh;

    // vehicle range using mileage and battery of vehicle
    int16 public vrange;

    // preference of user(0/1)
    int16 public preference;

    // calculate distance, discounted price(if any), direction of station, charge at arrival
    function calc_dist() public {
        for(uint i = 0; i<arr.length ;i++){

            int16 tempa = arr[i].x - veh.x;
            int16 tempb = arr[i].y - veh.y;
            int16 dir = 0;
            int16 distance = 0;
            int16 cost = 0;
            int16 net_remaining = 0;
            if((tempa >= 0) && (tempb >= 0))
                dir = 2;
            else if ((tempa <= 0) && (tempb <= 0))
                dir = 5;
            else if ((tempa >= 0) && (tempb <= 0))
                dir = 3;
            else
                dir =  7;

            if(tempa<0)
                tempa = tempa * -1;
            if(tempb<0)
                tempb = tempb * -1;
            distance = tempa + tempb;

            cost = arr[i].price;
            if(arr[i].visits >4)
                cost = cost - cost/5;

            net_remaining = arr[i].chrg_rem - distance/4;

            calculated.push(aftermath(arr[i].id,  arr[i].x, arr[i].y, cost, arr[i].visits, distance,
net_remaining,  arr[i].rating,dir));

            if(vrange >= calculated[i].distance)
                unopt_possible.push(calculated[i]);
            if(vrange >= calculated[i].distance && (((calculated[i].chrg_rem / veh.battery_perc) >3)
|| ((veh.dir % calculated[i].direction)==0))){
                int temp =  ((calculated[i].rating * calculated[i].chrg_rem * 10000) /
(calculated[i].distance * calculated[i].price));
                opt_possible.push(optimised(calculated[i].id, calculated[i].x, calculated[i].y,
calculated[i].price, calculated[i].visits, calculated[i].distance, calculated[i].chrg_rem,
calculated[i].rating, calculated[i].direction, temp));
                }
        }
    }
```

```
    // add station data
    function add() public {
        arr.push(station(0,  30, 84, 7, 4, 60,  7));
        arr.push(station(1,  71, 95, 10, 5, 90,  8));
        arr.push(station(2,  62, 68, 15, 8, 80,  6));
        arr.push(station(3,  22, 59, 12, 4, 68,  8));
        arr.push(station(4,  85, 41, 13, 6, 96,  8));

    }

    // sort stations by non decreasing distance, and then price if same distance
    function sort_dist() public {
        for(uint i = 0 ; i < unopt_possible.length ; i++){
            for(uint j = 0 ; j < unopt_possible.length ; j++){
                if(unopt_possible[i].distance < unopt_possible[j].distance){
                    aftermath memory tempo = unopt_possible[i];
                    unopt_possible[i] = unopt_possible[j];
                    unopt_possible[j] = tempo;

                }
            }
        }
        for(uint i = 0 ; i < unopt_possible.length ; i++){
            for(uint j = 0 ; j < unopt_possible.length ; j++){
                if(unopt_possible[i].distance == unopt_possible[j].distance &&
unopt_possible[i].price == unopt_possible[j].price){
                    aftermath memory tempo = unopt_possible[i];
                    unopt_possible[i] = unopt_possible[j];
                    unopt_possible[j] = tempo;

                }
            }
        }

    }

    // sort stations by non decreasing price, and then distance if same price
    function sort_cost() public {
        for(uint i = 0 ; i < unopt_possible.length ; i++){
            for(uint j = 0 ; j < unopt_possible.length ; j++){
                if(unopt_possible[i].price < unopt_possible[j].price){
                    aftermath memory tempo = unopt_possible[i];
                    unopt_possible[i] = unopt_possible[j];
                    unopt_possible[j] = tempo;

                }
            }
        }
        for(uint i = 0 ; i < unopt_possible.length ; i++){
            for(uint j = 0 ; j < unopt_possible.length ; j++){
                if(unopt_possible[i].price == unopt_possible[j].price && unopt_possible[i].distance
== unopt_possible[j].distance){
                    aftermath memory tempo = unopt_possible[i];
                    unopt_possible[i] = unopt_possible[j];
                    unopt_possible[j] = tempo;
```

```
                }
            }
        }

    }

    // call sort function based on user preference
    function sort() public {
        if(preference==0)
            sort_cost();
        else
            sort_dist();
    }

    // sort stations based on factor
    function sort_fact() public {
        for(uint i = 0 ; i < opt_possible.length ; i++){
            for(uint j = 0 ; j < opt_possible.length ; j++){
                if(opt_possible[i].score > opt_possible[j].score){
                    optimised memory tempo = opt_possible[i];
                    opt_possible[i] = opt_possible[j];
                    opt_possible[j] = tempo;

                }
            }
        }
    }

    // returns number of optimised stations possible
    function ret() public view returns(uint){
        return opt_possible.length;
    }

    // initialise vehicle data and call helping functions(add, calc_dist,sort_fact)
    function init(int16 x, int16 y, int16 battery_perc, int16 mileage, int16 dir, int16 pref) public
{
        veh.x = x;
        veh.y = y;
        veh.battery_perc = battery_perc;
        veh.mileage = mileage;
        veh.dir = dir;
        preference = pref;

        vrange = veh.battery_perc * veh.mileage;

        add();
        calc_dist();
        sort_fact();
    }
}
```