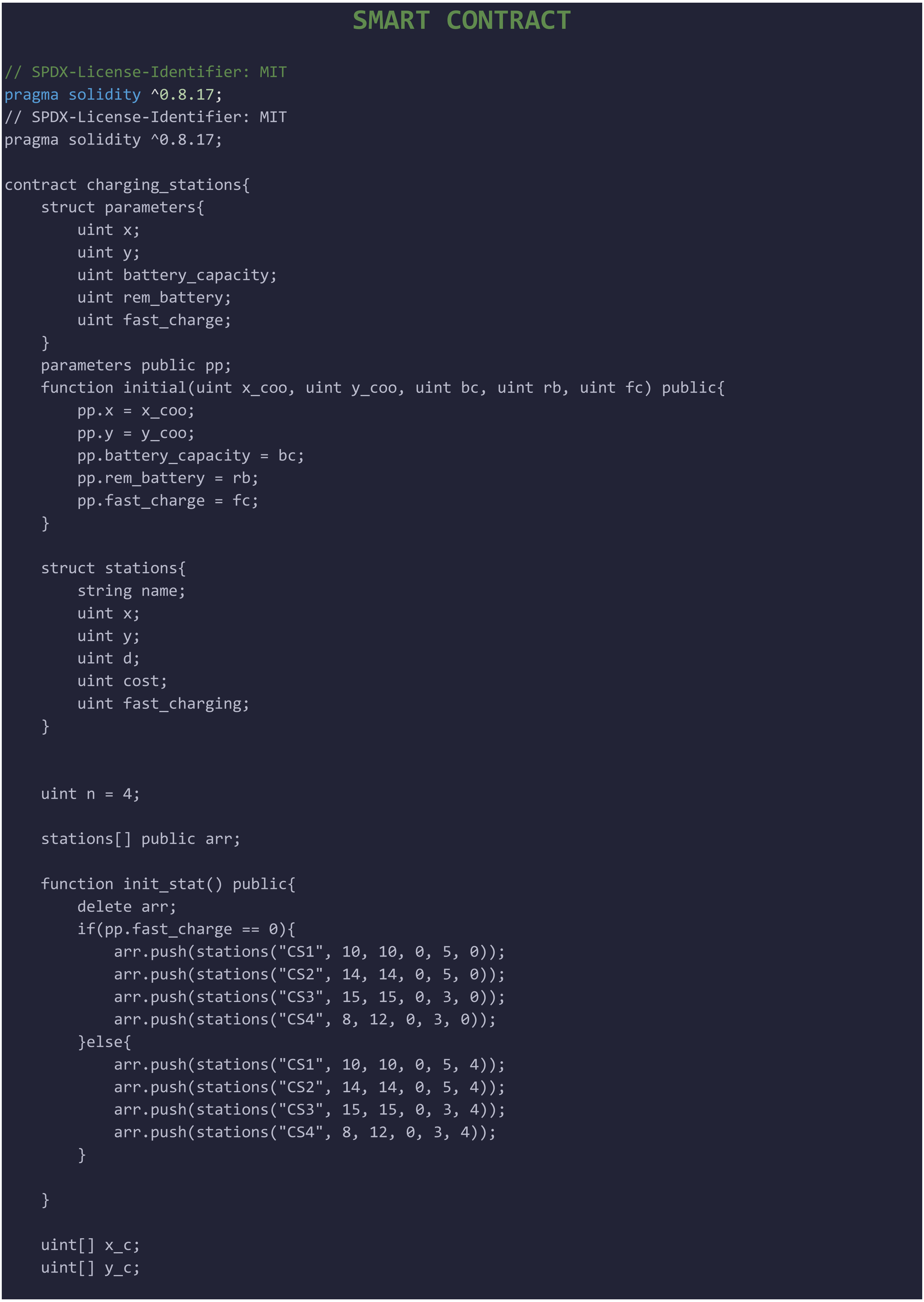
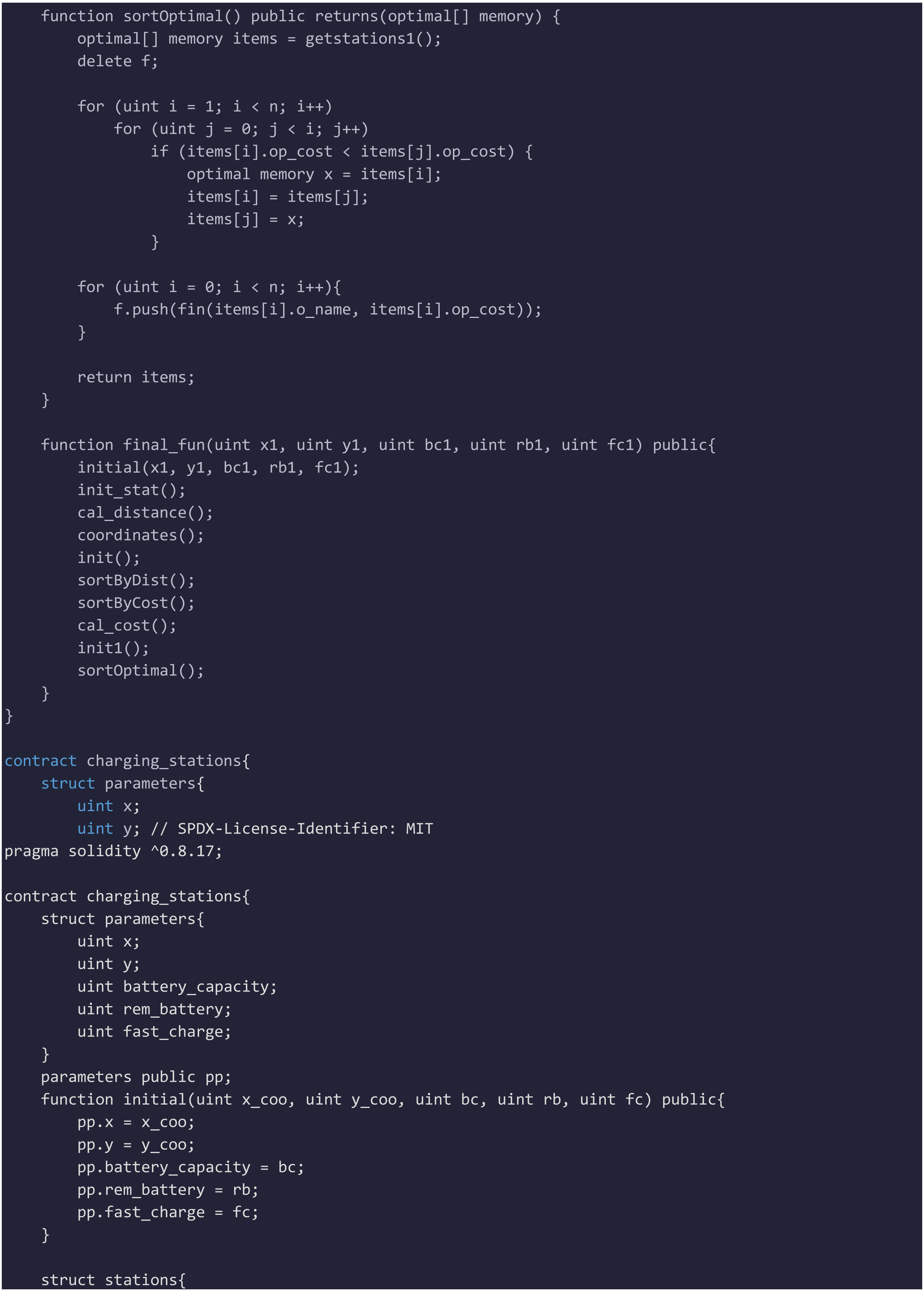


SMART CONTRACT

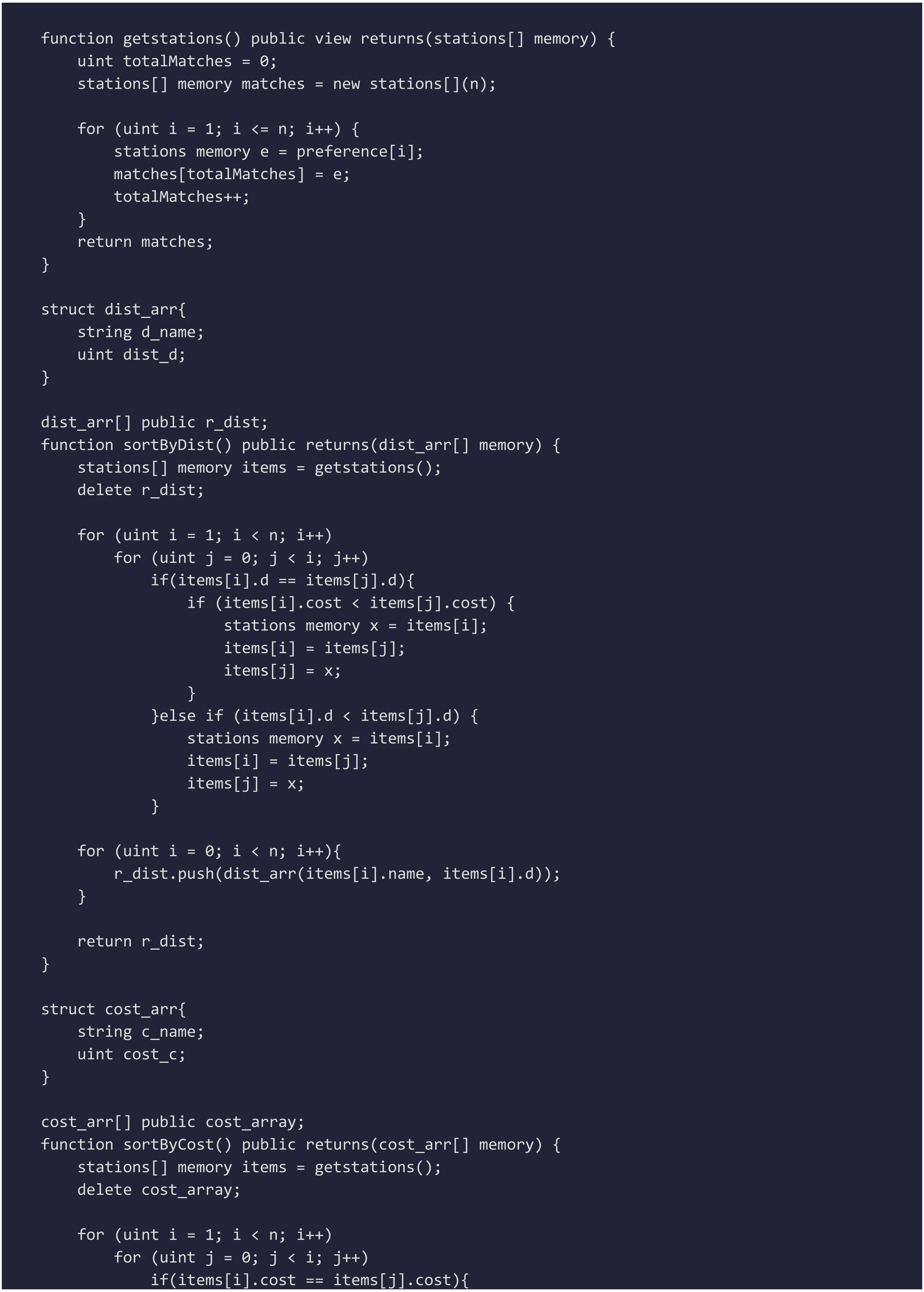
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract charging_stations{
    struct parameters{
        uint x;
        uint y;
        uint battery_capacity;
        uint rem_battery;
        uint fast_charge;
    }
    parameters public pp;
    function initial(uint x_coo, uint y_coo, uint bc, uint rb, uint fc) public{
        pp.x = x_coo;
        pp.y = y_coo;
        pp.battery_capacity = bc;
        pp.rem_battery = rb;
        pp.fast_charge = fc;
    }

    struct stations{
        string name;
        uint x;
        uint y;
        uint d;
        uint cost;
        uint fast_charging;
    }


    uint n = 4;

    stations[] public arr;

    function init_stat() public{
        delete arr;
        if(pp.fast_charge == 0){
            arr.push(stations("CS1", 10, 10, 0, 5, 0));
            arr.push(stations("CS2", 14, 14, 0, 5, 0));
            arr.push(stations("CS3", 15, 15, 0, 3, 0));
            arr.push(stations("CS4", 8, 12, 0, 3, 0));
        }else{
            arr.push(stations("CS1", 10, 10, 0, 5, 4));
            arr.push(stations("CS2", 14, 14, 0, 5, 4));
            arr.push(stations("CS3", 15, 15, 0, 3, 4));
            arr.push(stations("CS4", 8, 12, 0, 3, 4));
        }

    }

    uint[] x_c;
    uint[] y_c;
    function coordinates() public{
        delete x_c;
        delete y_c;
        for(uint i = 0; i < n; i++){
            x_c.push(arr[i].x);
            y_c.push(arr[i].y);
        }
    }

    function cal_distance() public {
        for(uint i = 0; i < n; i++){
            uint temp1 = arr[i].x - pp.x;
            uint temp2 = arr[i].y - pp.y;
            arr[i].d = (uint(temp1 + temp2));
        }
    }

    mapping(uint => stations) public preference;

    function init() public {
        for(uint i = 0; i < n; i++){
            preference[i + 1] = stations(arr[i].name, x_c[i], y_c[i], arr[i].d, arr[i].cost,
arr[i].fast_charging);
        }
    }

    function getstations() public view returns(stations[] memory) {
        uint totalMatches = 0;
        stations[] memory matches = new stations[](n);

        for (uint i = 1; i <= n; i++) {
            stations memory e = preference[i];
            matches[totalMatches] = e;
            totalMatches++;
        }
        return matches;
    }

    struct dist_arr{
        string d_name;
        uint dist_d;
    }

    dist_arr[] public r_dist;
    function sortByDist() public returns(dist_arr[] memory) {
        stations[] memory items = getstations();
        delete r_dist;

        for (uint i = 1; i < n; i++)
            for (uint j = 0; j < i; j++)
                if(items[i].d == items[j].d){
                    if (items[i].cost < items[j].cost) {
                        stations memory x = items[i];
                        items[i] = items[j];
                        items[j] = x;
                    }
                }else if (items[i].d < items[j].d) {
```

```
                stations memory x = items[i];
                items[i] = items[j];
                items[j] = x;
            }

    for (uint i = 0; i < n; i++){
        r_dist.push(dist_arr(items[i].name, items[i].d));
    }

    return r_dist;
}

struct cost_arr{
    string c_name;
    uint cost_c;
}

cost_arr[] public cost_array;
function sortByCost() public returns(cost_arr[] memory) {
    stations[] memory items = getstations();
    delete cost_array;

    for (uint i = 1; i < n; i++)
        for (uint j = 0; j < i; j++)
            if(items[i].cost == items[j].cost){
                if (items[i].d < items[j].d) {
                    stations memory x = items[i];
                    items[i] = items[j];
                    items[j] = x;
                }
            }else if (items[i].cost < items[j].cost) {
                stations memory x = items[i];
                items[i] = items[j];
                items[j] = x;
            }

    for (uint i = 0; i < n; i++){
        cost_array.push(cost_arr(items[i].name, items[i].cost));
    }

    return cost_array;
}


struct v_c{
    uint cc_v;
    uint ct_v;
    uint cw_v;
    uint cf_v;
}
v_c[] public v;


struct optimal{
    string o_name;
    uint op_cost;
}
```

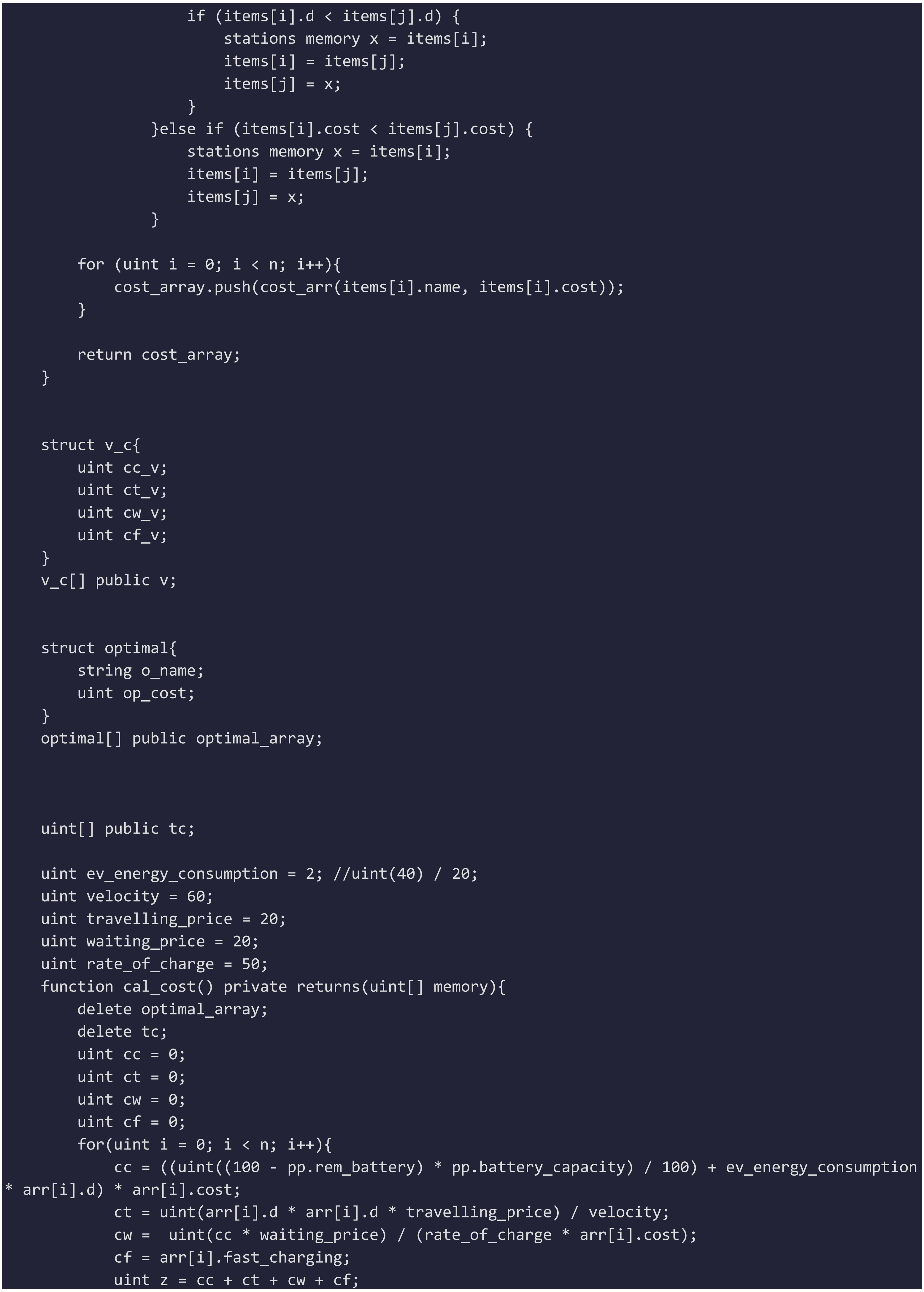
```
    optimal[] public optimal_array;



    uint[] public tc;

    uint ev_energy_consumption = 2; //uint(40) / 20;
    uint velocity = 60;
    uint travelling_price = 20;
    uint waiting_price = 20;
    uint rate_of_charge = 50;
    function cal_cost() private returns(uint[] memory){
        delete optimal_array;
        delete tc;
        uint cc = 0;
        uint ct = 0;
        uint cw = 0;
        uint cf = 0;
        for(uint i = 0; i < n; i++){
            cc = ((uint((100 - pp.rem_battery) * pp.battery_capacity) / 100) + ev_energy_consumption
* arr[i].d) * arr[i].cost;
            ct = uint(arr[i].d * arr[i].d * travelling_price) / velocity;
            cw =  uint(cc * waiting_price) / (rate_of_charge * arr[i].cost);
            cf = arr[i].fast_charging;
            uint z = cc + ct + cw + cf;
            tc.push(z);
            v.push(v_c(cc, ct, cw, cf));
            optimal_array.push(optimal(arr[i].name, z));
        }
        return tc;
    }

    mapping(uint => optimal) public preference1;

    function init1() public {
        for(uint i = 0; i < n; i++){
            preference1[i + 1] = optimal(arr[i].name, tc[i]);
        }
    }

    function getstations1() public view returns(optimal[] memory) {
        uint totalMatches = 0;
        optimal[] memory matches = new optimal[](n);

        for (uint i = 1; i <= n; i++) {
            optimal memory e = preference1[i];
            matches[totalMatches] = e;
            totalMatches++;
        }
        return matches;
    }

    struct fin{
        string cs;
        uint final_cost;
    }
    fin[] public f;
```



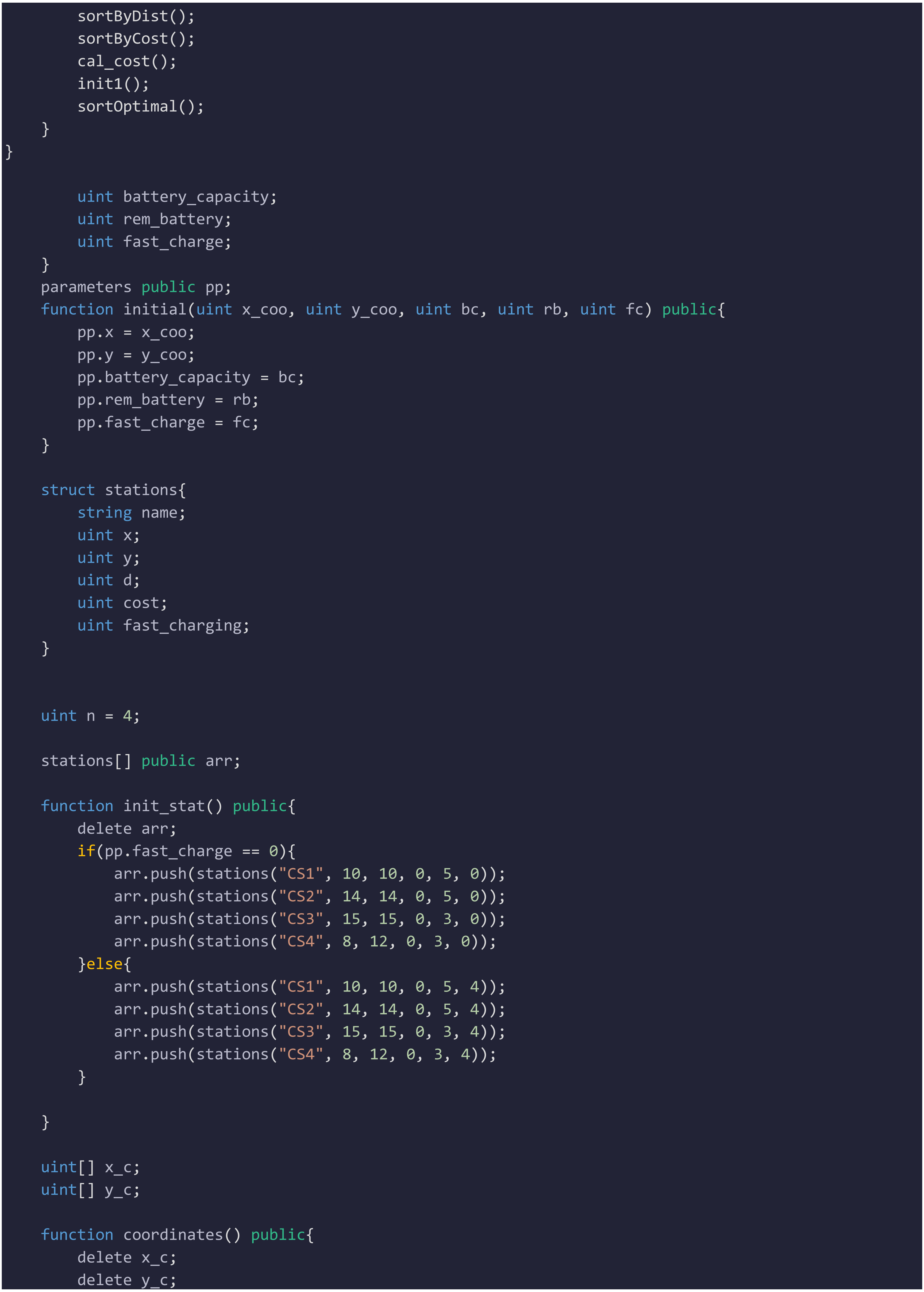
```
    function sortOptimal() public returns(optimal[] memory) {
        optimal[] memory items = getstations1();
        delete f;

        for (uint i = 1; i < n; i++)
            for (uint j = 0; j < i; j++)
                if (items[i].op_cost < items[j].op_cost) {
                    optimal memory x = items[i];
                    items[i] = items[j];
                    items[j] = x;
                }

        for (uint i = 0; i < n; i++){
            f.push(fin(items[i].o_name, items[i].op_cost));
        }

        return items;
    }

    function final_fun(uint x1, uint y1, uint bc1, uint rb1, uint fc1) public{
        initial(x1, y1, bc1, rb1, fc1);
        init_stat();
        cal_distance();
        coordinates();
        init();
        sortByDist();
        sortByCost();
        cal_cost();
        init1();
        sortOptimal();
    }
}

contract charging_stations{
    struct parameters{
        uint x;
        uint y; // SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract charging_stations{
    struct parameters{
        uint x;
        uint y;
        uint battery_capacity;
        uint rem_battery;
        uint fast_charge;
    }
    parameters public pp;
    function initial(uint x_coo, uint y_coo, uint bc, uint rb, uint fc) public{
        pp.x = x_coo;
        pp.y = y_coo;
        pp.battery_capacity = bc;
        pp.rem_battery = rb;
        pp.fast_charge = fc;
    }

    struct stations{
        string name;
        uint x;
        uint y;
        uint d;
        uint cost;
        uint fast_charging;
    }


    uint n = 4;

    stations[] public arr;

    function init_stat() public{
        delete arr;
        if(pp.fast_charge == 0){
            arr.push(stations("CS1", 10, 10, 0, 5, 0));
            arr.push(stations("CS2", 14, 14, 0, 5, 0));
            arr.push(stations("CS3", 15, 15, 0, 3, 0));
            arr.push(stations("CS4", 8, 12, 0, 3, 0));
        }else{
            arr.push(stations("CS1", 10, 10, 0, 5, 4));
            arr.push(stations("CS2", 14, 14, 0, 5, 4));
            arr.push(stations("CS3", 15, 15, 0, 3, 4));
            arr.push(stations("CS4", 8, 12, 0, 3, 4));
        }

    }

    uint[] x_c;
    uint[] y_c;

    function coordinates() public{
        delete x_c;
        delete y_c;
        for(uint i = 0; i < n; i++){
            x_c.push(arr[i].x);
            y_c.push(arr[i].y);
        }
    }

    function cal_distance() public {
        for(uint i = 0; i < n; i++){
            uint temp1 = arr[i].x - pp.x;
            uint temp2 = arr[i].y - pp.y;
            arr[i].d = (uint(temp1 + temp2));
        }
    }

    mapping(uint => stations) public preference;

    function init() public {
        for(uint i = 0; i < n; i++){
            preference[i + 1] = stations(arr[i].name, x_c[i], y_c[i], arr[i].d, arr[i].cost,
arr[i].fast_charging);
        }
    }
```



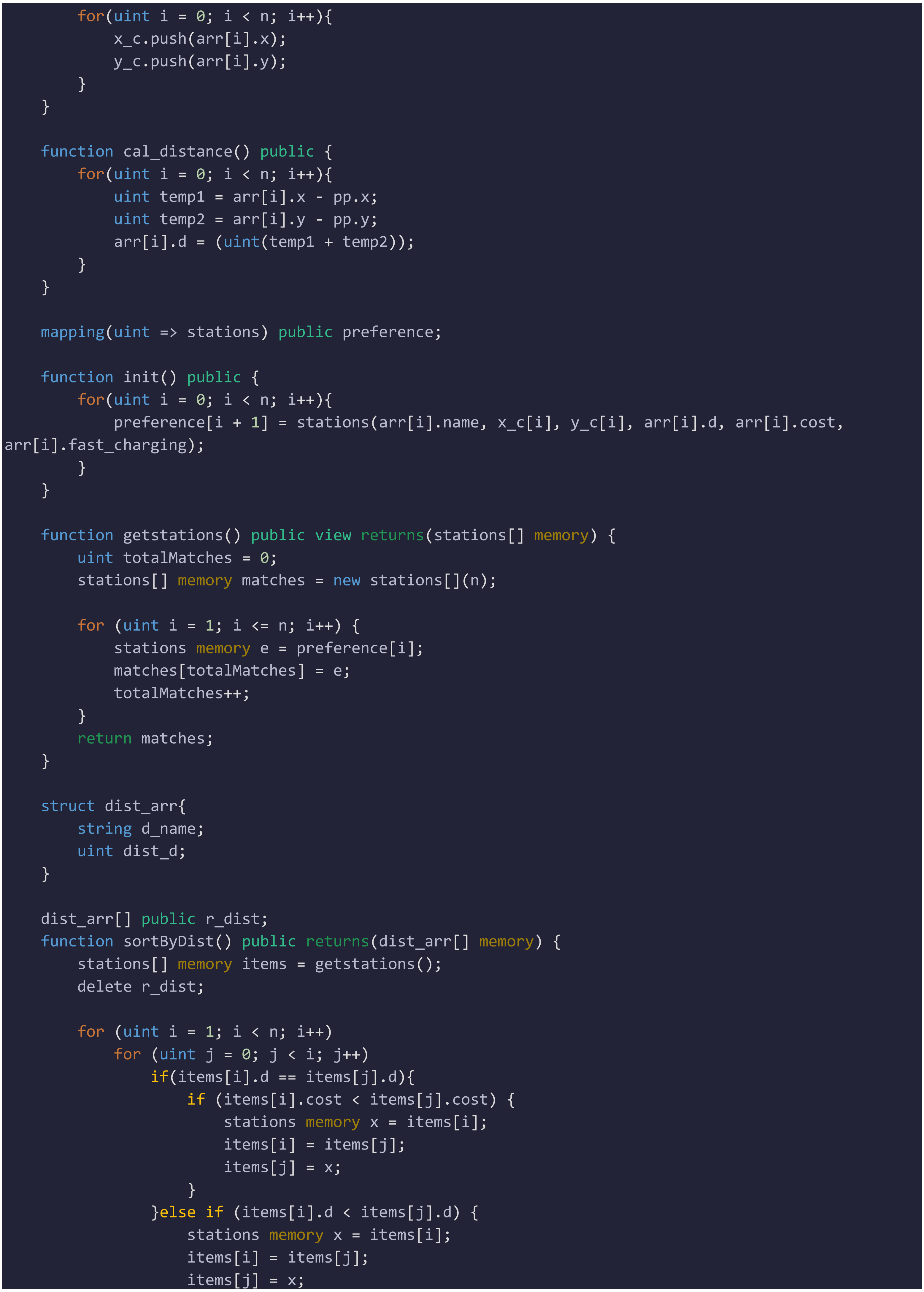
```
function getstations() public view returns(stations[] memory) {
    uint totalMatches = 0;
    stations[] memory matches = new stations[](n);

    for (uint i = 1; i <= n; i++) {
        stations memory e = preference[i];
        matches[totalMatches] = e;
        totalMatches++;
    }
    return matches;
}

struct dist_arr{
    string d_name;
    uint dist_d;
}

dist_arr[] public r_dist;
function sortByDist() public returns(dist_arr[] memory) {
    stations[] memory items = getstations();
    delete r_dist;

    for (uint i = 1; i < n; i++)
        for (uint j = 0; j < i; j++)
            if(items[i].d == items[j].d){
                if (items[i].cost < items[j].cost) {
                    stations memory x = items[i];
                    items[i] = items[j];
                    items[j] = x;
                }
            }else if (items[i].d < items[j].d) {
                stations memory x = items[i];
                items[i] = items[j];
                items[j] = x;
            }

    for (uint i = 0; i < n; i++){
        r_dist.push(dist_arr(items[i].name, items[i].d));
    }

    return r_dist;
}

struct cost_arr{
    string c_name;
    uint cost_c;
}

cost_arr[] public cost_array;
function sortByCost() public returns(cost_arr[] memory) {
    stations[] memory items = getstations();
    delete cost_array;

    for (uint i = 1; i < n; i++)
        for (uint j = 0; j < i; j++)
            if(items[i].cost == items[j].cost){
```



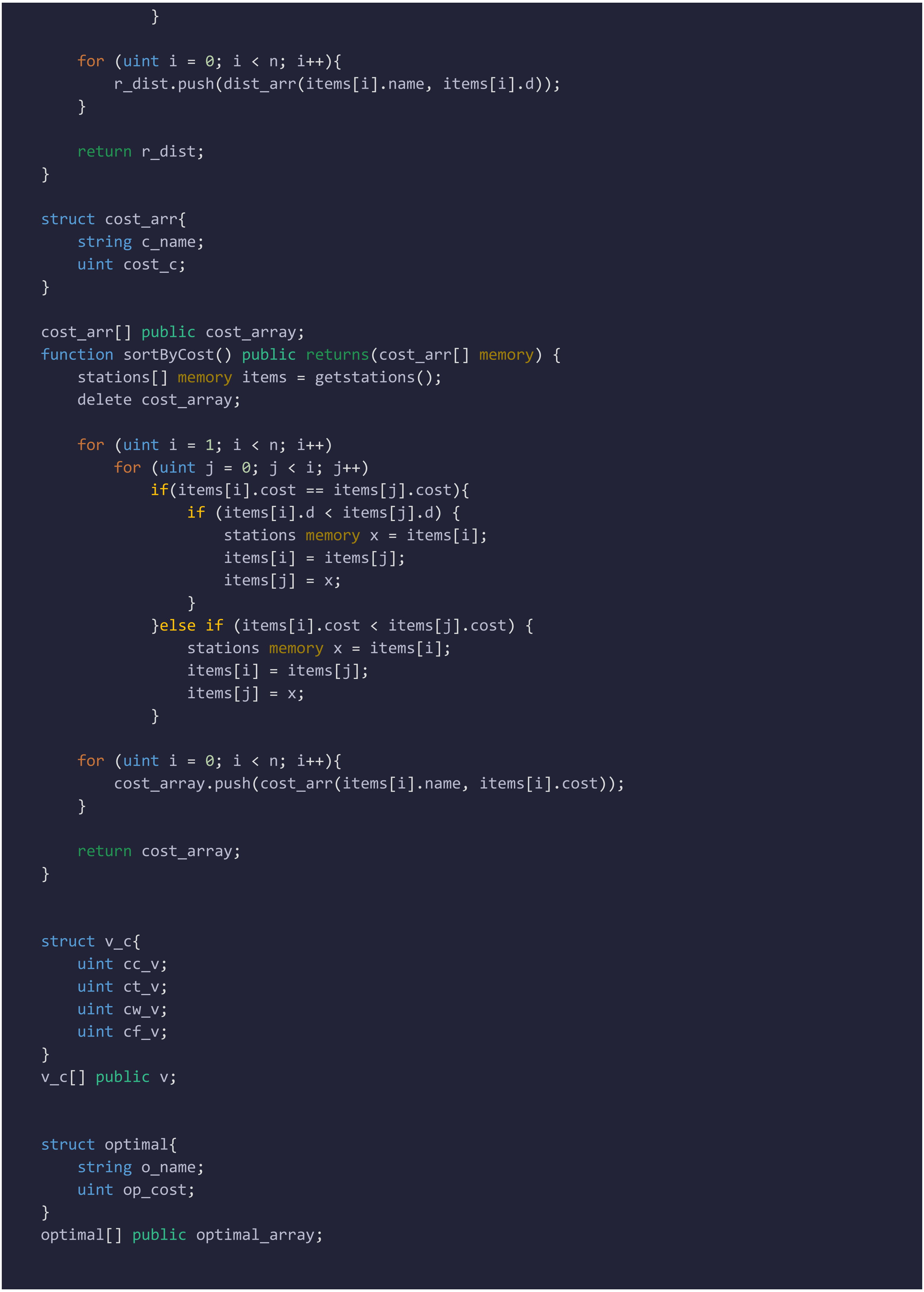
```
                    if (items[i].d < items[j].d) {
                        stations memory x = items[i];
                        items[i] = items[j];
                        items[j] = x;
                    }
                }else if (items[i].cost < items[j].cost) {
                    stations memory x = items[i];
                    items[i] = items[j];
                    items[j] = x;
                }

        for (uint i = 0; i < n; i++){
            cost_array.push(cost_arr(items[i].name, items[i].cost));
        }

        return cost_array;
    }


    struct v_c{
        uint cc_v;
        uint ct_v;
        uint cw_v;
        uint cf_v;
    }
    v_c[] public v;


    struct optimal{
        string o_name;
        uint op_cost;
    }
    optimal[] public optimal_array;



    uint[] public tc;

    uint ev_energy_consumption = 2; //uint(40) / 20;
    uint velocity = 60;
    uint travelling_price = 20;
    uint waiting_price = 20;
    uint rate_of_charge = 50;
    function cal_cost() private returns(uint[] memory){
        delete optimal_array;
        delete tc;
        uint cc = 0;
        uint ct = 0;
        uint cw = 0;
        uint cf = 0;
        for(uint i = 0; i < n; i++){
            cc = ((uint((100 - pp.rem_battery) * pp.battery_capacity) / 100) + ev_energy_consumption
* arr[i].d) * arr[i].cost;
            ct = uint(arr[i].d * arr[i].d * travelling_price) / velocity;
            cw =  uint(cc * waiting_price) / (rate_of_charge * arr[i].cost);
            cf = arr[i].fast_charging;
            uint z = cc + ct + cw + cf;
            tc.push(z);
            v.push(v_c(cc, ct, cw, cf));
            optimal_array.push(optimal(arr[i].name, z));
        }
        return tc;
    }

    mapping(uint => optimal) public preference1;

    function init1() public {
        for(uint i = 0; i < n; i++){
            preference1[i + 1] = optimal(arr[i].name, tc[i]);
        }
    }

    function getstations1() public view returns(optimal[] memory) {
        uint totalMatches = 0;
        optimal[] memory matches = new optimal[](n);

        for (uint i = 1; i <= n; i++) {
            optimal memory e = preference1[i];
            matches[totalMatches] = e;
            totalMatches++;
        }
        return matches;
    }

    struct fin{
        string cs;
        uint final_cost;
    }
    fin[] public f;
    function sortOptimal() public returns(optimal[] memory) {
        optimal[] memory items = getstations1();
        delete f;

        for (uint i = 1; i < n; i++)
            for (uint j = 0; j < i; j++)
                if (items[i].op_cost < items[j].op_cost) {
                    optimal memory x = items[i];
                    items[i] = items[j];
                    items[j] = x;
                }

        for (uint i = 0; i < n; i++){
            f.push(fin(items[i].o_name, items[i].op_cost));
        }

        return items;
    }

    function final_fun(uint x1, uint y1, uint bc1, uint rb1, uint fc1) public{
        initial(x1, y1, bc1, rb1, fc1);
        init_stat();
        cal_distance();
        coordinates();
        init();
```



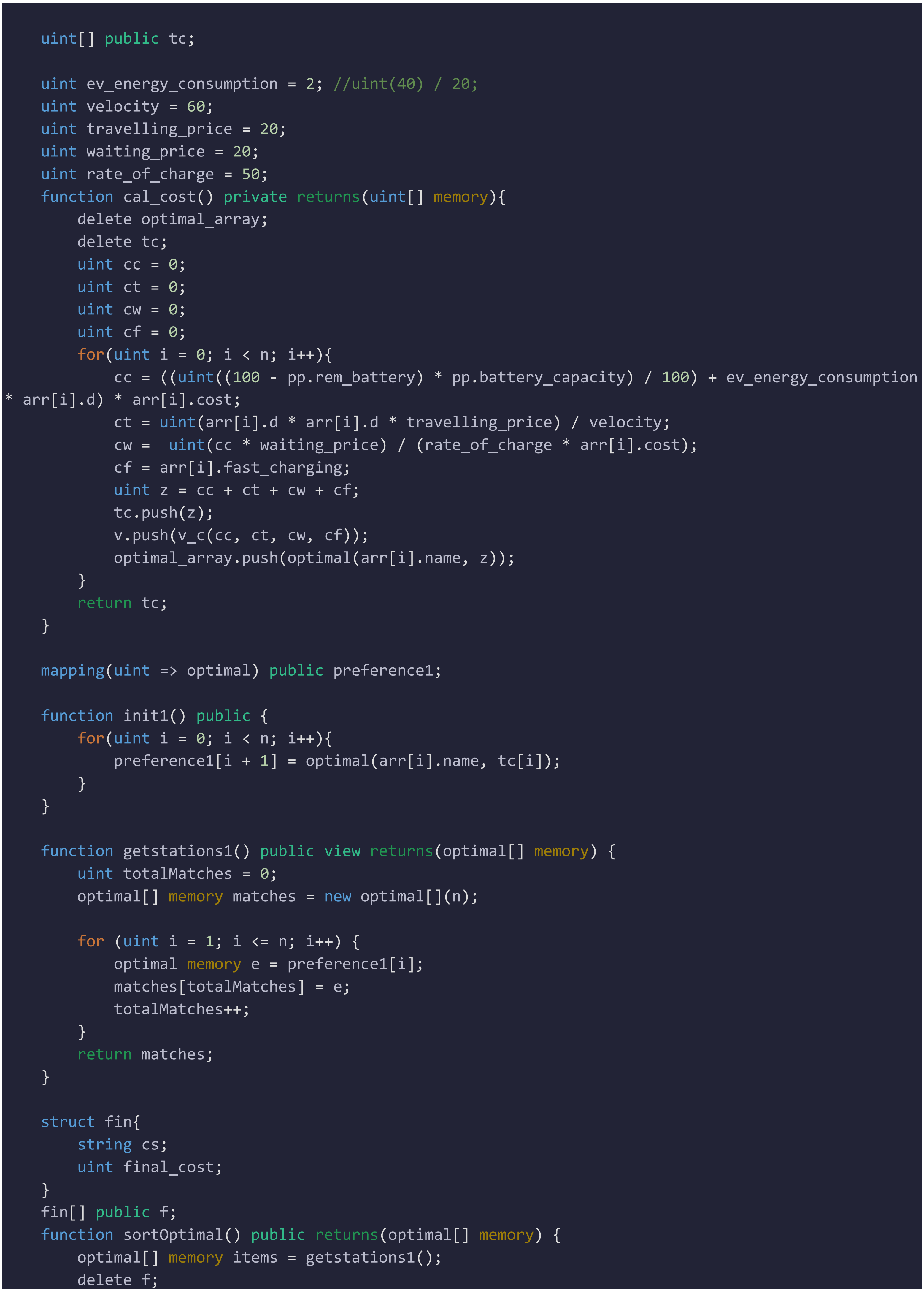
```
        sortByDist();
        sortByCost();
        cal_cost();
        init1();
        sortOptimal();
    }
}

        uint battery_capacity;
        uint rem_battery;
        uint fast_charge;
    }
    parameters public pp;
    function initial(uint x_coo, uint y_coo, uint bc, uint rb, uint fc) public{
        pp.x = x_coo;
        pp.y = y_coo;
        pp.battery_capacity = bc;
        pp.rem_battery = rb;
        pp.fast_charge = fc;
    }

    struct stations{
        string name;
        uint x;
        uint y;
        uint d;
        uint cost;
        uint fast_charging;
    }


    uint n = 4;

    stations[] public arr;

    function init_stat() public{
        delete arr;
        if(pp.fast_charge == 0){
            arr.push(stations("CS1", 10, 10, 0, 5, 0));
            arr.push(stations("CS2", 14, 14, 0, 5, 0));
            arr.push(stations("CS3", 15, 15, 0, 3, 0));
            arr.push(stations("CS4", 8, 12, 0, 3, 0));
        }else{
            arr.push(stations("CS1", 10, 10, 0, 5, 4));
            arr.push(stations("CS2", 14, 14, 0, 5, 4));
            arr.push(stations("CS3", 15, 15, 0, 3, 4));
            arr.push(stations("CS4", 8, 12, 0, 3, 4));
        }

    }

    uint[] x_c;
    uint[] y_c;

    function coordinates() public{
        delete x_c;
        delete y_c;
```



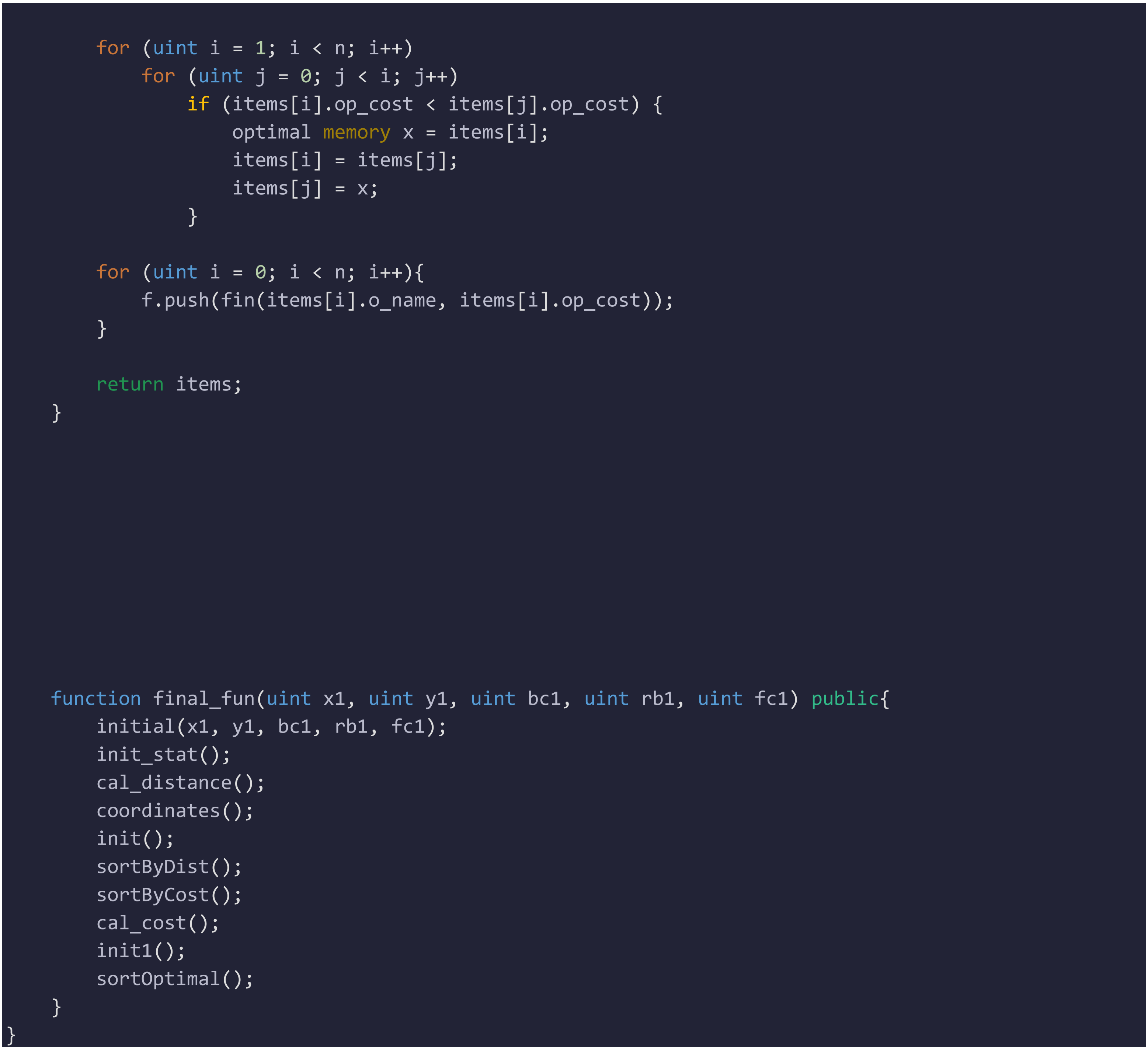
```
        for(uint i = 0; i < n; i++){
            x_c.push(arr[i].x);
            y_c.push(arr[i].y);
        }
    }

    function cal_distance() public {
        for(uint i = 0; i < n; i++){
            uint temp1 = arr[i].x - pp.x;
            uint temp2 = arr[i].y - pp.y;
            arr[i].d = (uint(temp1 + temp2));
        }
    }

    mapping(uint => stations) public preference;

    function init() public {
        for(uint i = 0; i < n; i++){
            preference[i + 1] = stations(arr[i].name, x_c[i], y_c[i], arr[i].d, arr[i].cost,
arr[i].fast_charging);
        }
    }

    function getstations() public view returns(stations[] memory) {
        uint totalMatches = 0;
        stations[] memory matches = new stations[](n);

        for (uint i = 1; i <= n; i++) {
            stations memory e = preference[i];
            matches[totalMatches] = e;
            totalMatches++;
        }
        return matches;
    }

    struct dist_arr{
        string d_name;
        uint dist_d;
    }

    dist_arr[] public r_dist;
    function sortByDist() public returns(dist_arr[] memory) {
        stations[] memory items = getstations();
        delete r_dist;

        for (uint i = 1; i < n; i++)
            for (uint j = 0; j < i; j++)
                if(items[i].d == items[j].d){
                    if (items[i].cost < items[j].cost) {
                        stations memory x = items[i];
                        items[i] = items[j];
                        items[j] = x;
                    }
                }else if (items[i].d < items[j].d) {
                    stations memory x = items[i];
                    items[i] = items[j];
                    items[j] = x;
```



```
            }

    for (uint i = 0; i < n; i++){
        r_dist.push(dist_arr(items[i].name, items[i].d));
    }

    return r_dist;
}

struct cost_arr{
    string c_name;
    uint cost_c;
}

cost_arr[] public cost_array;
function sortByCost() public returns(cost_arr[] memory) {
    stations[] memory items = getstations();
    delete cost_array;

    for (uint i = 1; i < n; i++)
        for (uint j = 0; j < i; j++)
            if(items[i].cost == items[j].cost){
                if (items[i].d < items[j].d) {
                    stations memory x = items[i];
                    items[i] = items[j];
                    items[j] = x;
                }
            }else if (items[i].cost < items[j].cost) {
                stations memory x = items[i];
                items[i] = items[j];
                items[j] = x;
            }

    for (uint i = 0; i < n; i++){
        cost_array.push(cost_arr(items[i].name, items[i].cost));
    }

    return cost_array;
}


struct v_c{
    uint cc_v;
    uint ct_v;
    uint cw_v;
    uint cf_v;
}
v_c[] public v;


struct optimal{
    string o_name;
    uint op_cost;
}
optimal[] public optimal_array;
```



```
    uint[] public tc;

    uint ev_energy_consumption = 2; //uint(40) / 20;
    uint velocity = 60;
    uint travelling_price = 20;
    uint waiting_price = 20;
    uint rate_of_charge = 50;
    function cal_cost() private returns(uint[] memory){
        delete optimal_array;
        delete tc;
        uint cc = 0;
        uint ct = 0;
        uint cw = 0;
        uint cf = 0;
        for(uint i = 0; i < n; i++){
            cc = ((uint((100 - pp.rem_battery) * pp.battery_capacity) / 100) + ev_energy_consumption
* arr[i].d) * arr[i].cost;
            ct = uint(arr[i].d * arr[i].d * travelling_price) / velocity;
            cw =  uint(cc * waiting_price) / (rate_of_charge * arr[i].cost);
            cf = arr[i].fast_charging;
            uint z = cc + ct + cw + cf;
            tc.push(z);
            v.push(v_c(cc, ct, cw, cf));
            optimal_array.push(optimal(arr[i].name, z));
        }
        return tc;
    }

    mapping(uint => optimal) public preference1;

    function init1() public {
        for(uint i = 0; i < n; i++){
            preference1[i + 1] = optimal(arr[i].name, tc[i]);
        }
    }

    function getstations1() public view returns(optimal[] memory) {
        uint totalMatches = 0;
        optimal[] memory matches = new optimal[](n);

        for (uint i = 1; i <= n; i++) {
            optimal memory e = preference1[i];
            matches[totalMatches] = e;
            totalMatches++;
        }
        return matches;
    }

    struct fin{
        string cs;
        uint final_cost;
    }
    fin[] public f;
    function sortOptimal() public returns(optimal[] memory) {
        optimal[] memory items = getstations1();
        delete f;
```



```
        for (uint i = 1; i < n; i++)
            for (uint j = 0; j < i; j++)
                if (items[i].op_cost < items[j].op_cost) {
                    optimal memory x = items[i];
                    items[i] = items[j];
                    items[j] = x;
                }

        for (uint i = 0; i < n; i++){
            f.push(fin(items[i].o_name, items[i].op_cost));
        }

        return items;
    }









    function final_fun(uint x1, uint y1, uint bc1, uint rb1, uint fc1) public{
        initial(x1, y1, bc1, rb1, fc1);
        init_stat();
        cal_distance();
        coordinates();
        init();
        sortByDist();
        sortByCost();
        cal_cost();
        init1();
        sortOptimal();
    }
}
```