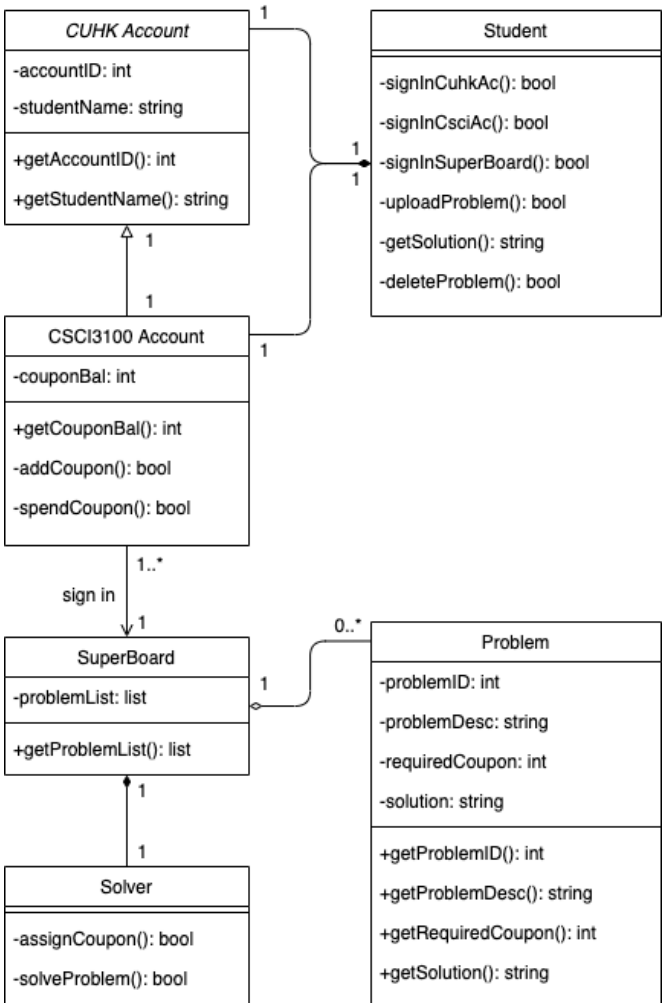
CSCI 3100 Software Engineering

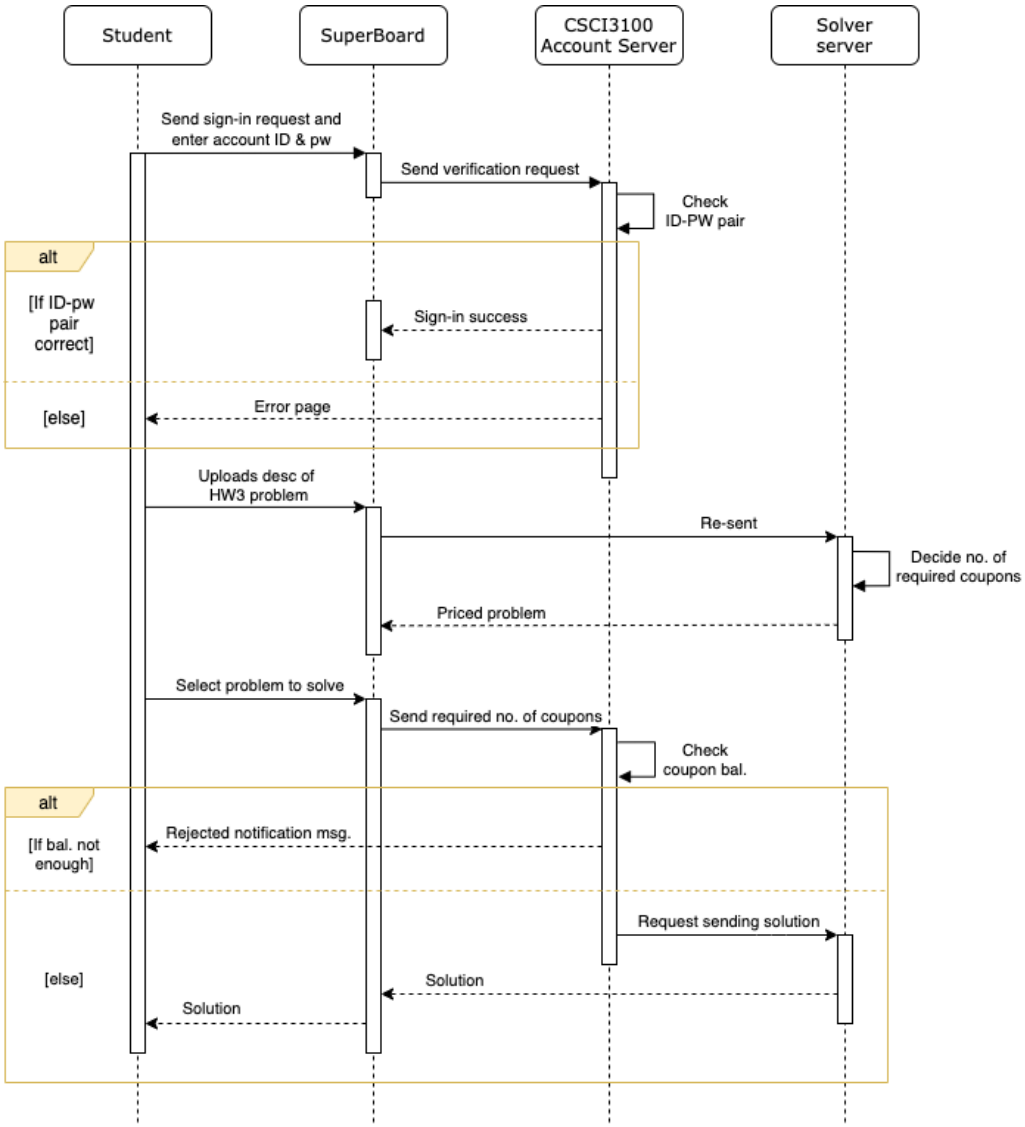
Assignment 5

Jonathan Lam Pui Hong (1155093597)

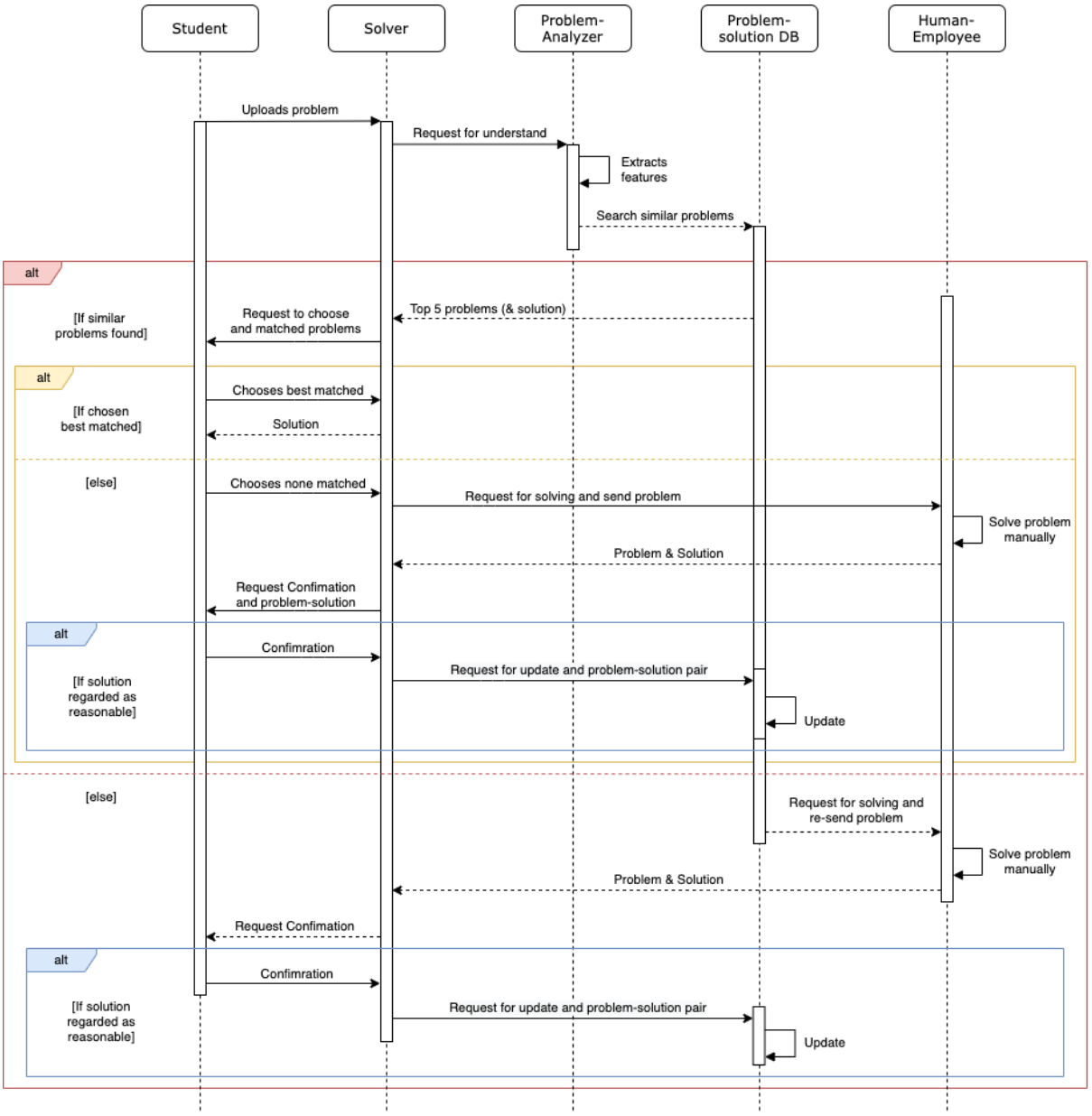
Question 1.1



Question 1.2


Student
SuperBoard
CSCI3100 Account Server
Solver server
Send sign-in request and enter account ID & pw
Send verification request
Check ID-PW pair
alt
[If ID-pw pair correct]
Sign-in success
[else]
Error page
Uploads desc of HW3 problem
Re-sent
Decide no. of required coupons
Priced problem
Select problem to solve
Send required no. of coupons
Check coupon bal.
alt
[If bal. not enough]
Rejected notification msg.
Request sending solution
[else]
Solution
Solution

Question 2


Student
Solver
Problem-Analyzer
Problem-solution DB
Human-Employee
Uploads problem
Request for understand
Extracts features
Search similar problems
alt
[If similar problems found]
Top 5 problems (& solution)
Request to choose and matched problems
alt
Chooses best matched
[If chosen best matched]
Solution
[else]
Chooses none matched
Request for solving and send problem
Solve problem manually
Problem & Solution
Request Confimation and problem-solution
alt
Confimration
[If solution regarded as reasonable]
Request for update and problem-solution pair
Update
[else]
Request for solving and re-send problem
Solve problem manually
Problem & Solution
Request Confimation
alt
Confimration
[If solution regarded as reasonable]
Request for update and problem-solution pair
Update

Question 3

Table 3-1 Pair	Jaccard	Hamming	Edit
Query-S1	0.15	17	12
Query-S2	2/7 ≈ 0.286	10	9

Table 3-2 Pair	Jaccard	Hamming	Edit
Query-S1	9/67 ≈ 0.134	58	60
Query-S2	17/45 ≈ 0.378	28	28

Jaccard function (JS)

```
function jaccard(str1, str2) {
    function union(set1, set2) {
        let _union = new Set(set1)
        for (let elem of set2)
            _union.add(elem)
        return _union
    }
    function intersection(set1, set2) {
        let _intersection = new Set()
        for (let elem of set2)
            if (set1.has(elem))
                _intersection.add(elem)
        return _intersection
    }

    let set1 = new Set(str1.split(' '));
    let set2 = new Set(str2.split(' '));
    console.log(intersection(set1, set2).size)
    console.log(union(set1, set2).size)
    return intersection(set1, set2).size / union(set1, set2).size
}
```

Hamming function (JS)

```
function hamming(str1, str2) {
    function union(list1, list1) {
        let set1 = new Set(list1);
        let set2 = new Set(list2);
        let _union = new Set(set1);
        for (let elem of set2)
            _union.add(elem)
        return _union
    }
```

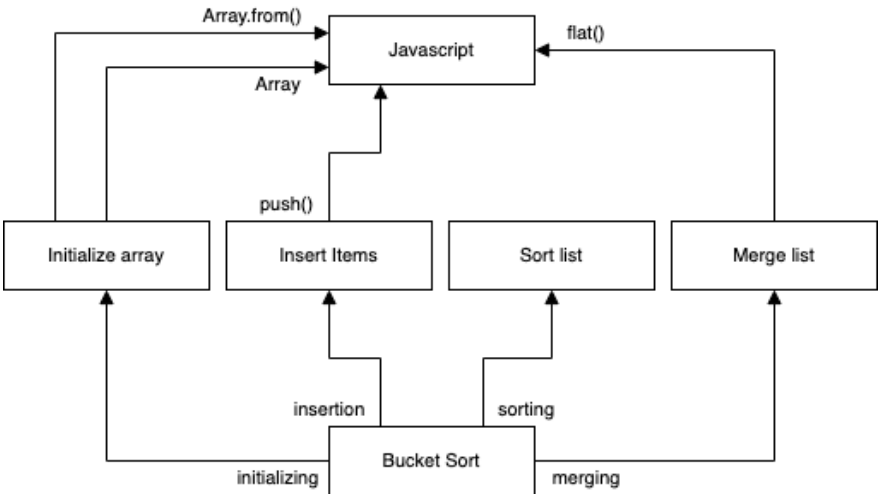
```
    function onehot(vocab, list) {
        let _onehot = []
        for (let elem of vocab)
            if (list.includes(elem))
                _onehot.push(1)
            else _onehot.push(0)
        return _onehot
    }

    let list1 = str1.split(' ');
    let list2 = str2.split(' ');
    let vocab = union(list1, list2);
    let onehot1 = onehot(vocab, list1);
    let onehot2 = onehot(vocab, list2);

    let count = 0;
    for (let i = 0; i < vocab.size; i++)
        if (onehot1[i] !== onehot2[i])
            count++;
    return count;
}
```

Edit distance function (JS)

Following the similar question explained in the tutorial, instead of a genuine and straight-forward recursion approach, the following algorithm checks the substring of the two strings and finds the minimum cost from the neighbouring indexes. It will run through the strings (arrays) in the nested for loops to prevent duplicate checking.

```
function editDis(str1, str2) {
    let list1 = str1.split(' ');
    let list2 = str2.split(' ');
    let arr = Array.from(Array(list1.length+1), () => new Array(list2.length+1));
    arr[0][0] = 0;
    for (let row=1; row < list1.length+1; row++) arr[row][0] = row;
    for (let col=1; col < list2.length+1; col++) arr[0][col] = col;
    for (let row=1; row < list1.length+1; row++)
        for (let col=1; col < list2.length+1; col++) {
            let subCost;
            if (list1[row-1] == list2[col-1])
                subCost = 0;
            else subCost = 1;
            arr[row][col] = Math.min(
                arr[row-1][col]+1,
                arr[row][col-1]+1,
                arr[row-1][col-1]+subCost);
        };
    return arr[list1.length][list2.length]
}
```

Question 4



Per the instruction in Piazza, the GDN above treats each steps as a module that is USE(D) by the module **Bucket Sort**. These steps-derived modules further use functions and prototypes exported by JavaScript.

The following stepwise refinement will be implemented with pseudocode (in B&W) and JavaScript (in color).

```
// -- INITIAL

function bucketSort(numArr)
    const MAX_NUM = 10000;
    const NUM_OF_BUCKETS = 1000;
    let buckets = Array of length NUM_OF_BUCKETS
    insert items into buckets
    sort items in each bucket
    merge buckets
end function
```

```
function bucketSort(numArr) {
    const MAX_NUM = 10000;
    const NUM_OF_BUCKETS = 1000;
    let buckets = Array(NUM_OF_BUCKETS);
    // TODO: Insert items into buckets
    // TODO: Sort items in each bucket
    // TODO: Merge buckets
}
```

```
// -- REFINEMENT 1: Add outmost components

function bucketSort(numArr)
    const MAX_NUM = 10000;
    const NUM_OF_BUCKETS = 1000;
>   let buckets = initialize each elem in Array (of length NUM_OF_BUCKETS) to []
>   for i = 0 to numArr.length-1 do
>       let index = floor( numArr[i] / (MAX_NUM+1) * NUM_OF_BUCKETS )
>       push numArr[i] into buckets[index]
>   end for
>   for i = 0 to NUM_OF_BUCKETS-1 do
>       sort buckets[i]
>   end for
>   for i = NUM_OF_BUCKETS-1 to 0 do
>       concat buckets[i] and buckets[i-1]
>   end for
end function
```

```
function bucketSort(numArr) {
    const MAX_NUM = 10000;
    const NUM_OF_BUCKETS = 1000;
    let buckets = Array.from(Array(NUM_OF_BUCKETS), () => []);
    for (let i = 0; i < numArr.length; i++)
        buckets[~~(numArr[i]/(MAX_NUM+1) * NUM_OF_BUCKETS)].push(numArr[i])
    for (let i = 0; i < NUM_OF_BUCKETS; i++)
        // TODO: sort
        sort(buckets[i])
    return buckets.flat()
}
```

```
// -- REFINEMENT 2: Add sorting function

function bucketSort(numArr)
    const MAX_NUM = 10000;
    const NUM_OF_BUCKETS = 1000;
    let buckets = Array.from(Array(NUM_OF_BUCKETS), () => []);
    for i = 0 to numArr.length-1 do
        let index = floor( numArr[i] / (MAX_NUM+1) * NUM_OF_BUCKETS )
        push numArr[i] into buckets[index]
    end for
    for i = 0 to NUM_OF_BUCKETS-1 do
>       for n = 1 to buckets[i].length-1 do
>           for m = n to 1 do
>               if buckets[i][m] < buckets[i][m-1] then
>                   swap buckets[i][m] and buckets[i][m-1]
>               end if
>           end for
>       end for
    end for
    for i = NUM_OF_BUCKETS-1 to 0 do
>       for elem in buckets[i-1]
>           push elem into buckets[i]
>       end for
    end for
end function


function bucketSort(numArr) {
    function insertionSort(arr) {
        for (let n = 1; n < arr.length; n++)
            for (let m = n; m > 0; m--)
                if (arr[m] < arr[m-1]) {
                    let temp = arr[m];
                    arr[m] = arr[m-1];
                    arr[m-1] = temp;
                }
        return arr;
    }
    const MAX_NUM = 10000;
    const NUM_OF_BUCKETS = 1000;
    let buckets = Array.from(Array(NUM_OF_BUCKETS), () => []);
    for (let i = 0; i < numArr.length; i++)
        buckets[~~(numArr[i]/(MAX_NUM+1) * NUM_OF_BUCKETS)].push(numArr[i])
    for (let i = 0; i < NUM_OF_BUCKETS; i++)
        insertionSort(buckets[i])
    return buckets.flat()
}
```

```
// -- REFINEMENT 3: Clean up insertionSort and add single case condition

function bucketSort(numArr)
    const MAX_NUM = 10000;
    const NUM_OF_BUCKETS = 1000;
    let buckets = Array.from(Array(NUM_OF_BUCKETS), () => []);
    for i = 0 to numArr.length-1 do
        let index = floor( numArr[i] / (MAX_NUM+1) * NUM_OF_BUCKETS )
        push numArr[i] into buckets[index]
    end for
    for i = 0 to NUM_OF_BUCKETS-1 do
>       if buckets[i].length <= 1 then
>           exit loop
>       else
            for n = 1 to buckets[i].length-1 do
                for m = n to 1 do
                    if buckets[i][m] < buckets[i][m-1] then
                        swap buckets[i][m] and buckets[i][m-1]
                    end if
                end for
            end for
>       end if
    end for
    for i = NUM_OF_BUCKETS-1 to 0 do
        for elem in buckets[i-1]
            push elem into buckets[i]
        end for
    end for
end function


function bucketSort(numArr) {
    function insertionSort(arr) {
        if (arr.length <= 1) return arr;
        for (let n = 1; n < arr.length; n++)
            for (let m = n; m > 0 && arr[m] < arr[m-1]; m--)
                [arr[m], arr[m-1]] = [arr[m-1], arr[m]];
        return arr;
    }
    const MAX_NUM = 10000;
    const NUM_OF_BUCKETS = 1000;
    let buckets = Array.from(Array(NUM_OF_BUCKETS), () => []);
    for (let i = 0; i < numArr.length; i++)
        buckets[~~(numArr[i]/(MAX_NUM+1) * NUM_OF_BUCKETS)].push(numArr[i])
    for (let i = 0; i < NUM_OF_BUCKETS; i++)
        insertionSort(buckets[i])
    return buckets.flat()
}
```