WEEK 10 STUDIO

AGENDA

- Arrays and Loops
- Sorting & Recursion problems with loops
- Arrays & Loops in Environment Model
- Studio Sheet

SOME ANNOUNCEMENTS

- Reading Assessment 2
- Studio 11 is during public holiday
- Returning of robot kits after studio
- Practical Assessment

ARRAYS

- Data structure that stores a sequence of data elements
- Elements in an array are accessed through indices.
    - The first element has an index of 0.
    - Accessing an element of an array by its index is $O(1)$. This is a very useful property that helps with a lot of problems.

```
const seq = [10, 5, 8];
seq[0]; // 10
seq[2]; // 8
```

ARRAYS

- Elements of an array can be reassigned. This implies that an array is a mutable object.

```
const seq = ["hello"];
seq[0] = 100;
seq[0]; // 100
```

- Elements can be assigned to any position in the array. Assigning an element to an index after the last element of the array will increase the size of the array.

```
const seq = ["hello"];
seq[3] = 100;
seq; // ["hello", , , 100]
array_length(seq); // 4
```

LOOPS

- 2 kinds of loops: `while` and `for` loops

```
// while loop
let count = 10;
while (count > 0) {
    // do something
    count = count - 1;
}

// for loop
for (let count = 10; count > 0; count = count - 1) {
    // do something
}
```

LOOPS

- Many of our recursion problems can also be solved with loops. Try implementing them yourself!
  - Factorial
  - Square root
  - Power function
  - Fibonacci
  - Greatest common divisor (GCD)
  - Least common multiple (LCM)
  - Hanoi tower
  - Coin change
  - Permutation / combination

ARRAYS & LOOPS IN ENVIRONMENT MODEL

- How do we represent arrays and loops in the environment model?
- Frame names and evaluate statements are optional.

MEMOIZATION

- Idea: Storing previously calculated values in a data structure so we don't need to recalculate them again.
- Very effective for tree recursion!

STUDIO SHEET 10

QUESTION 2

```
function bubble_sort(lst) {
    const len = length(lst);
    for (let i = len - 1; i >= 1; i = i - 1) {
        let xs = lst;
        for (let j = 0; j < i; j = j + 1) {
            const curr = head(xs);
            const next = head(tail(xs));
            if (curr > next) {
                set_head(tail(xs), curr);
                set_head(xs, next);
            } else {}
            xs = tail(xs);
        }
    }
    return lst;
}
```

QUESTION 2 EXTRA (IF TIME PERMITS)

```
function sort_recur(lst) {
    const len = length(lst);
    for (let i = len - 1; i >= 1; i = i - 1) {
        const curr = head(lst);
        const next = head(tail(lst));
        if (curr > next) {
            set_head(tail(lst), curr);
            set_head(lst, next);
            sort_recur(tail(lst));
        } else {
            sort_recur(tail(lst));
        }
    }
    return lst;
}
```

- What is happening here?
- Why does the recursion still work when there is no base case?
- What is the time complexity? How to analyze the time complexity?

QUESTION 3(A)

Solution

QUESTION 3(B)

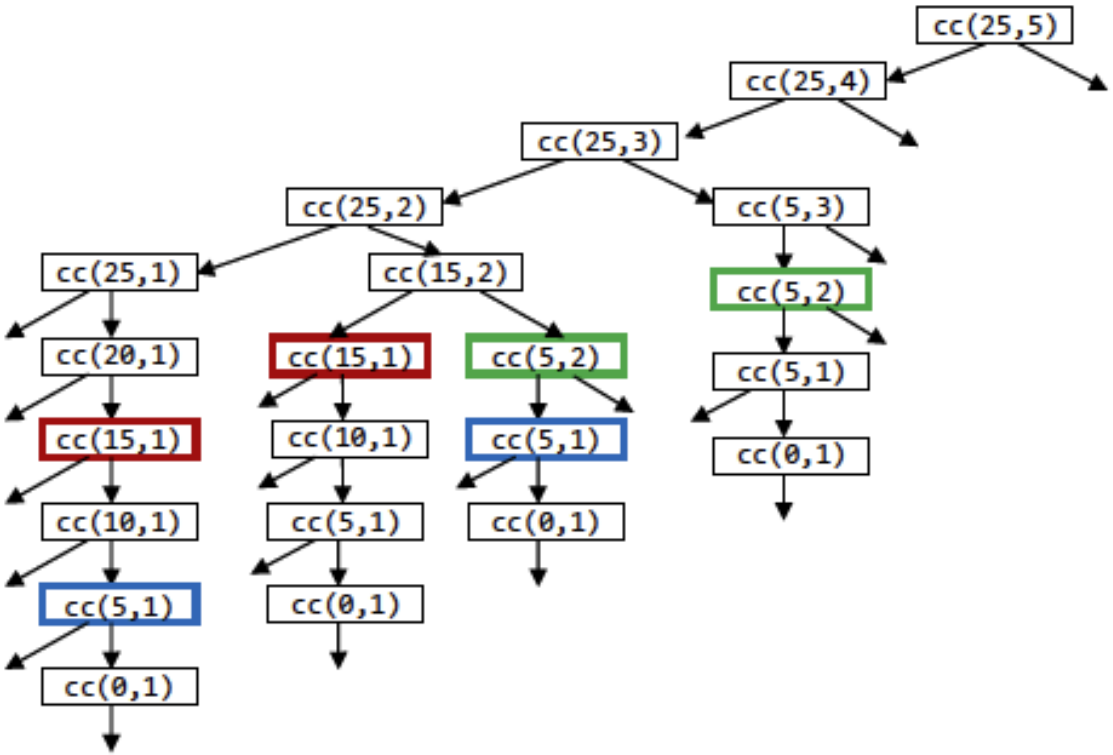
```
function num_of_live_neighbours(game, n, r, c) {
    let live_count = 0;
    for (let dr = -1; dr <= 1; dr = dr + 1) {
        for (let dc = -1; dc <= 1; dc = dc + 1) {
            live_count = live_count +
                game[(r + dr + n) % n][(c + dc + n) % n];
        }
    }
    return live_count - game[r][c];
}
```

QUESTION 3(C)

```
function next_generation(game, n) {
    const next = make_2D_zero_array(n, n);
    for (let row = 0; row < n; row = row + 1) {
        for (let col = 0; col < n; col = col + 1) {
            live_count = num_of_live_neighbours(game, n,
                                                row, col);
            if (live_count < 2 || live_count > 3) {
                next[row][col] = 0;
            } else if (live_count === 3) {
                next[row][col] = 1;
            } else {}
        }
    }
    return next;
}
```

QUESTION 4

The following shows a partial call tree for `cc(25, 5)` and we can see many duplicate function calls. Therefore, memoization will be suitable for optimizing `cc`.



QUESTION 4

Solutions

Time complexity: $O(nk)$

Space complexity: $O(nk)$

QUESTION 5

- Part (a)
  - Solution
  - Time complexity: $O(3^{R}C)$
- Part (b)
  - Solution
  - Time complexity: $O(RC)$