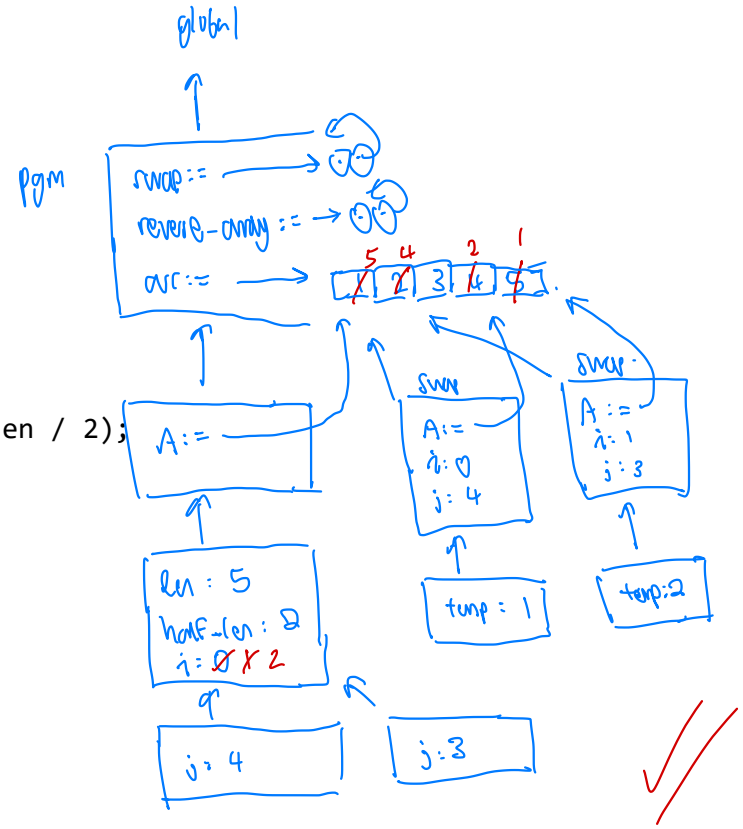
National University of Singapore
School of Computing
CS1101S: Programming Methodology
Semester I, 2021/2022

S10
Searching and Sorting II; Memoization

Problems:

1. Consider the following Source program:

```
function swap(A, i, j) {
    let temp = A[i];
    A[i] = A[j];
    A[j] = temp;
}

function reverse_array(A) {
    const len = array_length(A);
    const half_len = math_floor(len / 2);
    let i = 0;
    while (i < half_len) {
        const j = len - 1 - i;
        swap(A, i, j);
        i = i + 1;
    }
}

const arr = [1, 2, 3, 4, 5];
reverse_array(arr);
arr;
```



Draw the diagram to show the environment during the evaluation of the program. Show all the frames that are created during the program evaluation. Show the final value of each binding. Note that calls of primitive functions, such as `array_length` and `math_floor`, do not create any frames.

2. The following function, `bubblesort_array`, is an implementation of the ***Bubble Sort*** algorithm to sort an array of numbers into ascending order:

```
function bubblesort_array(A) {
    const len = array_length(A);
    for (let i = len - 1; i >= 1; i = i - 1) {
        for (let j = 0; j < i; j = j + 1) {
            if (A[j] > A[j + 1]) {
                const temp = A[j];
                A[j] = A[j + 1];
                A[j + 1] = temp;
            }
        }
    }
}
```

(a) What is the order of growth of its runtime for an input array of n elements?

(b) Write the function, `bubblesort_list`, that takes as argument a **list** of numbers and uses the bubble sort algorithm to sort the list into ascending order. Your function **must not create any new pair or array**, and **must not use the function `set_tail`**. Its runtime must have the same order of growth as that of `bubblesort_array`.

```
function bubblesort_list(L) {
    // ???
}
```

Example use:

```
const LL = list(3, 5, 2, 4, 1);
bubblesort_list(LL);
LL; // should show [1, [2, [3, [4, [5, null]]]]]
```

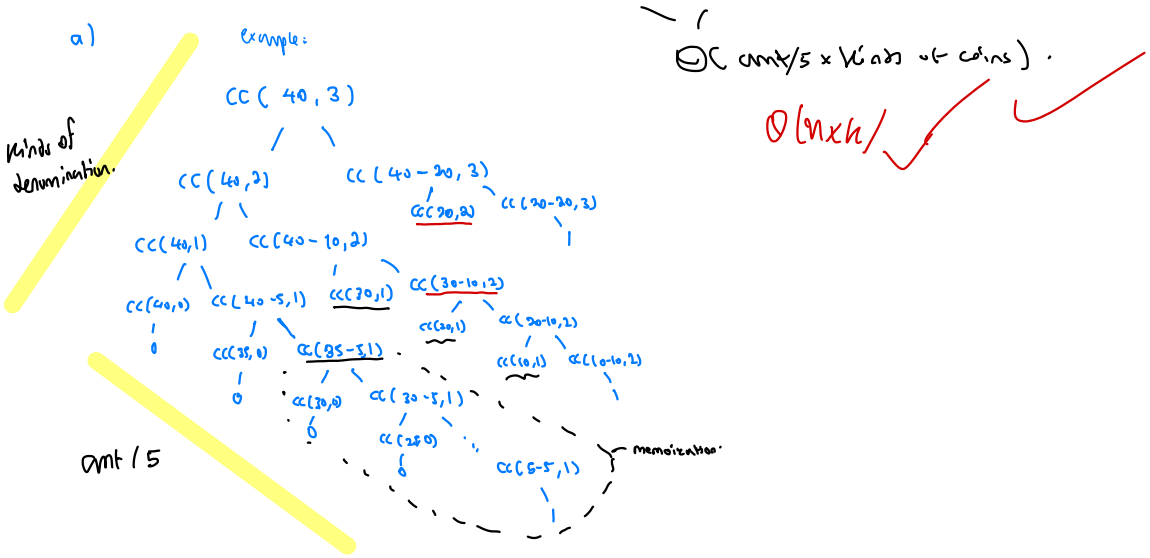
3. Consider the `cc` (coin change) function presented in Lecture L3:

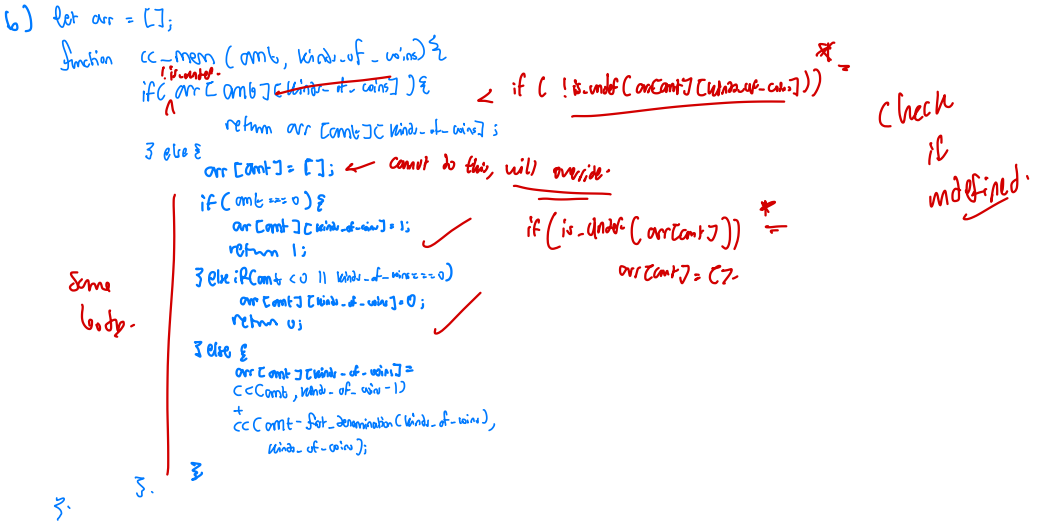
```
function cc(amount, kinds_of_coins) {
    return amount === 0
           ? 1
           : amount < 0 || kinds_of_coins === 0
           ? 0
           : cc(amount, kinds_of_coins - 1)
             +
             cc(amount - first_denomination(kinds_of_coins),
                kinds_of_coins);
}

function first_denomination(kinds_of_coins) {
    return kinds_of_coins === 1 ?   5 :
           kinds_of_coins === 2 ?  10 :
           kinds_of_coins === 3 ?  20 :
           kinds_of_coins === 4 ?  50 :
           kinds_of_coins === 5 ? 100 : 0;
}
```

(a) Is function `cc` a good candidate for memoization? Support your answer with an example.

(b) If memoization is suitable for function `cc`, provide the implementation.

(c) What are the orders of growth in *time* and *space* of the memoized version?





(0,0) (0,3) (3,0) at (0,0) ~ corner in (0,3).

1, 2, 3, 4 → 4 3 2 1

5, 6, 7, 8 → 8 7 6 5

9, 10, 11, 12 ⇒ 12 11 10 9.

13, 14, 15, 16. → 16 15 14 13. ~ (0,3) at (3,3).

(3,0) (3,3).

(3,3) at (0,3).

(0,3)

1 5 9 13 ~ (0,0)

2 6 10 14

3 7 11 15

4 8 12 16. ~ (3,3)

(3,0).

Swap row and col.