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

R9
Searching and Sorting II; Memoization

Problems:

1. Consider the following `make_search` function, which uses the `linear_search` function from Lecture L9:

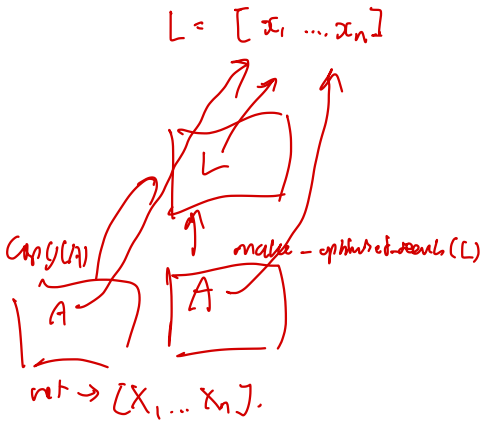
```
function make_search(A) {
    return x => linear_search(A, x);
}
```

The following shows a typical use of `make_search`:

```
const my_array = [3,41,20,1,5,16,4,0,14,6,17,8,4,0,2];
const my_search = make_search(my_array);
my_search(14); // returns true
...            // many more calls to my_search
my_search(30); // returns false
```

Based on the above use case, complete the function `make_optimized_search` to make an optimized version of `make_search`. Note that the input array `A` must not be modified by your function, or by any subsequent search operations. You should consider only the search and sort algorithms covered in Lecture L9.

```
function make_optimized_search(A) {
    // ???
    return x => ???;
}
```

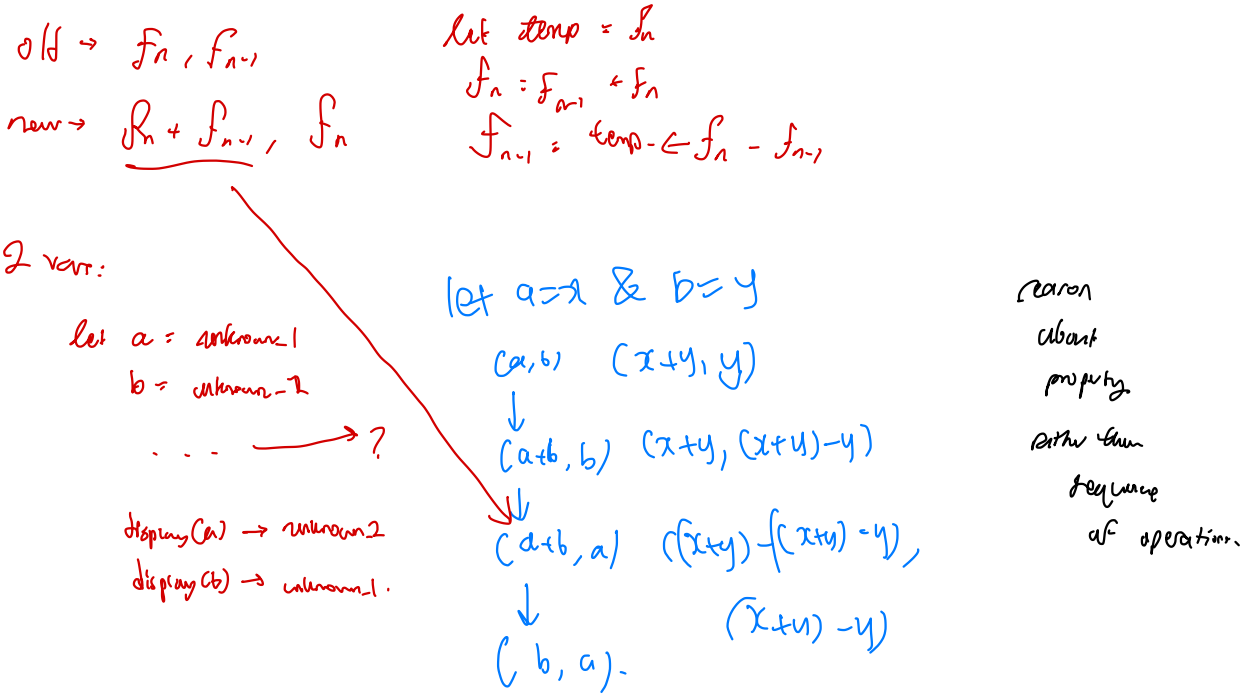


2. Computing Fibonacci numbers.

(a) Write a function `fib(n)` that returns the nth Fibonacci number, using the following idea: As usual, `fib(0)` should return 0 and `fib(1)` should return 1. In order to compute `fib(n)`, place the first two Fibonacci numbers in an array, and then use a loop to fill the array with the Fibonacci numbers until the desired n.

```
function fib(n)
    const fibs = [0,1];
    for (let i=2; i<=n; i++) {
        fibs[i] = fibs[i-1] + fibs[i-2];
    }
    return fibs[n];
}
```

(b) Observe that we don't really need the whole array to compute the next Fibonacci number. The previous two values suffice. Use this observation to write a version that uses *only ~~two~~ three variables* (do not count the parameter variable n and the loop control variable), instead of an array.



3. In Lecture L9, we saw the memoized *n*-choose-*k* function `mchoose`:

```
function mchoose(n, k) {
    if (read(n, k) !== undefined) {
        return read(n, k);
    } else {
        const result = k > n
                       ? 0
                       : k === 0 || k === n
                       ? 1
                       : mchoose(n - 1, k) + mchoose(n - 1, k - 1);
        write(n, k, result);
        return result;
    }
}
```

Draw the call tree for the evaluation of the function call `mchoose(7, 4)`. Show only the calls to `mchoose`, and for calls whose return values can be found in the table `mem`, indicate so.

What is the order of growth in *time* of `mchoose(n, k)`?
What is the order of growth in *space* of `mchoose(n, k)`?