

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:

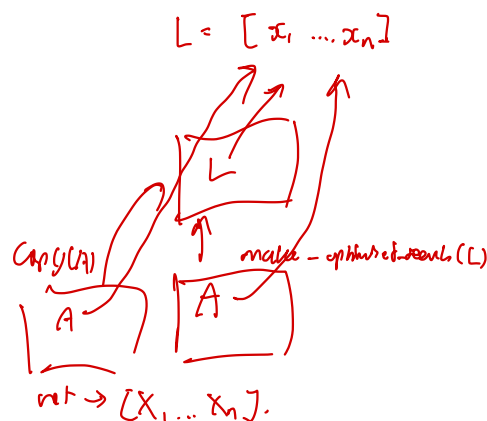
```
function make_search(A) {
    return x => linear_search(A, x);
}
```

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 => ???;
}
```



Handwritten notes and annotations:

- $O(n)$ (pointing to `linear_search`)
- preprocessing cost $\rightarrow O(n \log n)$
- binary search $\rightarrow O(\log n)$
- sorting $\rightarrow O(n \log n)$
- remember \rightarrow or not good for splicing
- tree structure \Rightarrow better
- for this search input range $\log n$
- lower bound to break even $\rightarrow n \log n + (\log n)^2 = O(n \log n)$
- $n \log n > \log^2 n$

for `copy(A)`:

```
let res = [];
for (let i=0; i < arr.length(A); i=i+1) {
    res[i] = A[i];
}
```

for `res`:

Trade off
in processing
time

2. Computing Fibonacci numbers.

- (a) Write a function `fib(n)` that returns the n th 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 .

```

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

```

Diagram illustrating the array `fibs` of size $n+1$. The first two elements are 0 and 1. The rest of the array is filled with Fibonacci numbers up to index n .

↳ `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 variables (do not count the parameter variable n and the loop control variable), instead of an array.

old $\rightarrow f_n, f_{n-1}$
 new $\rightarrow f_{n+1}, f_n$

let temp = f_n
 $f_n = f_{n-1} + f_n$
 $f_{n-1} = \text{temp} - f_n - f_{n-1}$

2 var:

let $a = \text{unknown-1}$
 $b = \text{unknown-2}$
 ... $\rightarrow ?$

display(a) $\rightarrow \text{unknown-2}$
 display(b) $\rightarrow \text{unknown-1}$.

let $a=x$ & $b=y$

$(a, b) \rightarrow (x+y, y)$
 \downarrow
 $(a+b, b) \rightarrow (x+y, (x+y)-y)$
 \downarrow
 $(a+b, a) \rightarrow ((x+y) - ((x+y)-y), (x+y)-y)$
 \downarrow
 $(b, a) \rightarrow (x+y-y, x+y)$

reason about property after each sequence of operations.

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;
  }
}
```

Store old values.

Seen = Spill it out.

not seen & compute.

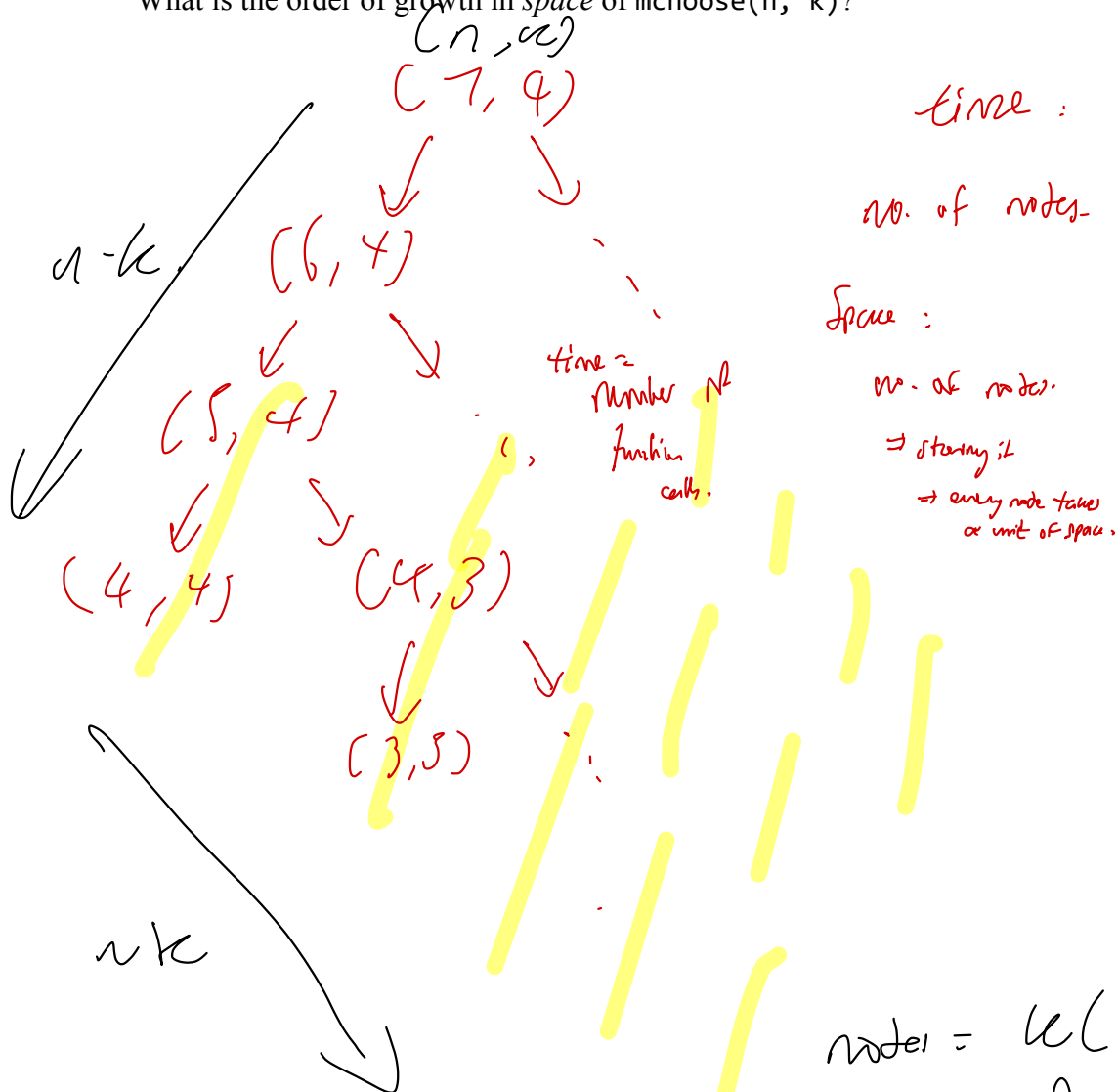
Store.

left first.

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)`?



$$\begin{aligned}
 \text{nodes} &= k(n-k) \\
 &= O(nk - k^2) \\
 &= O(nk)
 \end{aligned}$$