# FOCS Homework 19

You may edit your answers into this file, or add a separate file in the same directory.

If you add a separate file, please include the following at the top:

```

Student Name: Leon Lam [change to your name]

Check one:

[ ] I completed this assignment without assistance or external resources.

[X] I completed this assignment with assistance from \_\_\_

and/or using these external resources: Wikipedia

```

## 0. [Not a question] Terminology

In class on Monday, we drew [\*\*call graphs\*\*](https://en.wikipedia.org/wiki/Call\_graph) of the `fib` function, with and without memoization.

This particular kind of call graph is a \*\*dynamic call graph\*\*, whose nodes are function \*invocations\* during the execution of a program. The nodes of a \*\*static call graph\*\* are function \*names\*; arrows indicate appearances of a call in the program source text.

A dynamic call graph of a \*recursive\* function is also called a \*\*recursion graph\*\*.

## 1. Predicate Calculus – Models

Consider Table 1:

| x | y |

| ---- | ---- |

| a | b |

| b | c |

| a | c |

| c | d |

| d | c |

| c | c |

| b | a |

\*Table 1\*: A model for `loves(x, y)`

a. For each of the following, is Table 1 a model? Explain briefly why or why not.

1. ∃x. ∀y. loves(x, y) **Nope. Each letter only loves two other letters at most, but there are four letters in total.**

2. ∃y. ∀x. loves(x, y) **Yes. c is loved by all the letters.**

3. ∀x. ∃y. loves(x, y) **Yes.** **Every letter loves at least one other letter.**

4. ∀y. ∃x. loves(x, y) **Yes. Every letter is loved by at least one other.**

5. ∃x. ∃y. loves(x, y) **Yes. a loves b.**

6. ∃x. ∀y. ¬loves(x, y) **No. There is no letter in x that does not love a letter in y.**

7. ∃x. ¬∀y. loves(x, y) **Yes if we take ‘NOT ALL Y’ to mean ‘some elements of Y, but not every single one’ - a loves b and c but not d. No if we take ‘NOT ALL Y’ to mean ‘the inverse of the set Y’, since every element in set X loves something in set Y.**

b. Consider three models: (i) objects are a, b, c, d; `loves` is Table 1; (ii) objects are integers; `loves(x, y) ` ≝ x ≧ y; (iii) `loves(x, y)` ≝ x = y. In which of these models are these statements true:

| Statement | Table 1 | x ≧ y | x = y |

| ---------------------------------------- | ------- | ----- | ----- |

| ∀x, x. loves(x, x) | **F** | **T** | **T** |

| ∀x, y. loves(x, y) → loves(y, x) | **F** | **F** | **T** |

| ∀x, y, z. loves(x, y) ∧ loves(y, z) → loves(x, z) | **F** | **T** | **T** |

c. (Optional) (Because we love graphs) Draw Table 1 as a digraph. What claims do each of the following make about a node x or y, in terms of its indegree and outdegree?

1. ∀x. loves(x, y) **Outdegree = number of nodes**

2. ∀y. loves(x, y) **Indegree = number of nodes**

3. ∃x. loves(x, y) **Graph is not entirely disconnected. At least one node x has outdegree of 1 or above. (∃y vs ∀y will change interpretation.)**

4. ∃y. loves(x, y) **Graph is not entirely disconnected. At least one node y has indegree of 1 or above. (∃x vs ∀x will change interpretation.)**

## 2. Predicate Calculus – Proofs

Given axioms:

1. ∀x. sum( x, 0, x )

2. ∀x, y, z. sum( x, y, z ) → sum( x, s(y), s(z) )

3. ∀x . product( x, 0, 0 )

4. ∀x, y, z, k. product( x, y, z ) ∧ sum( x, z, k ) → product( x, s(y), k )

Prove:

1. ∃x. sum( x, x, s(s(0)) )

**Let x = s(0). From axiom 1 we get sum(s(0),0,s(0)). We then apply axiom 2 to get sum(s(0),s(0),s(s(0))). Therefore there exists an x = s(0) for which sum(x,x,s(s(0))) is true.**

2. ∀x. sum( 0, x, x ) [hint: induction]

**Let x = 0. Axiom 1 gives us sum(0,0,0). Axiom 2 applied once then gives us sum(0,s(0),s(0)), and applying axiom 2 again and again will result in sum(0,s^n(0),s^n(0)). Therefore, for any x that can be expressed as s^n(0), sum(0,x,x) is true.**

3. [optional super-challenge] ∀x, y, z. sum( x, y, z ) → sum( y, x, z )

## 3. Maximum Subarray

The [maximum subarray problem](https://en.wikipedia.org/wiki/Maximum\_subarray\_problem) (\*aka\* maximum contiguous subsequence, \*aka\* maximum value contiguous subsequence) is “the task of finding the contiguous subarray within a one-dimensional array of numbers which has the largest sum. For example, for the sequence of values −2, 1, −3, 4, −1, 2, 1, −5, 4; the contiguous subarray with the largest sum is 4, −1, 2, 1, with sum 6” ([Wikipedia](https://en.wikipedia.org/wiki/)).

```python

# Source: wikipedia

def max\_subarray(xs):

max\_ending\_here = max\_so\_far = 0

for x in xs:

max\_ending\_here = max(0, max\_ending\_here + x)

max\_so\_far = max(max\_so\_far, max\_ending\_here)

return max\_so\_far

```

a. Draw a recursion graph of `max\_subarray([−2, 1, −3, 4, −1, 2, 1, −5, 4])`.

**max\_subarray doesn’t seem like it does any recursion at all. It iterates through the elements in the list, making two constant time comparisons at each step. The inbuilt function ‘max’ is also called, but I don’t think that’s what you’re looking for?**

b. Is `max\_subarray` an example of divide and conquer? Why or why not?

**No. It iterates through the entire problem.**

c. Is `max\_subarray` an example of dynamic programming? Why or why not?

**Maybe. It keeps track of the sum from the beginning, discarding the sum and starting again if the sum ever drops below zero. It then remembers the largest sum and compares all subsequent sums against the largest, replacing when necessary. This means it never has to handle more elements than the input at one time, and in most cases will handle less at once.**

d. Consider a \*memoized\* version of `max\_subarray`. (You don't need to produce code for this. You do need to understand what memoization does to a call graph.) Draw the recursion graph for `memoized\_max\_subarray([−2, 1, −3, 4, −1, 2, 1, −5, 4])`.

**Max\_subarray will call max on two items, then store the result in a dictionary where key = (num1, num2) and value = max(num1, num2). This way, the max function will only have to be called once each time.**

e. [Optional challenge] Produce working code for memoized `max\_subarray`.

## 4. Binary Search

```python

def binary\_search\_array\_helper(x, xs, left, right):

if left == right: return None

middle = int((left + right) / 2)

if x < xs[middle]:

return binary\_search\_array\_helper(x, xs, left, middle)

elif xs[middle] < x:

return binary\_search\_array\_helper(x, xs, middle, right)

else:

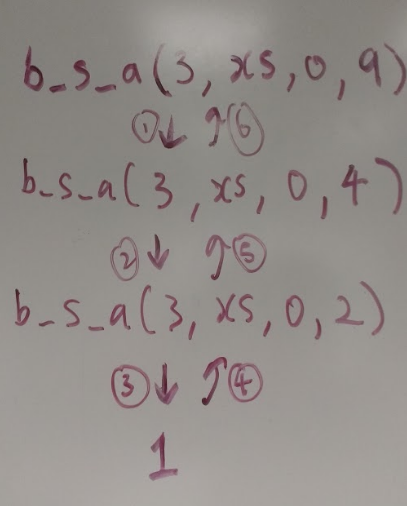
return middle

def binary\_search\_array(x, xs):

return binary\_search\_array\_helper(x, xs, 0, len(xs))

```

a. Above is an implementation of the [binary search algorithm](https://en.wikipedia.org/wiki/Binary\_search\_algorithm). Draw the recursion graph for `binary\_search\_array(3, [1, 3, 4, 6, 7, 8, 10, 13, 14]))`.



Where b\_s\_a = binary\_search\_array\_helper and xs = [1, 3, 4, 6, 7, 8, 10, 13, 14].

b. Is `binary\_search\_array` an example of divide and conquer? Why or why not?

**Yup, it cuts the input in half each time.**

c. Is `binary\_search\_array` an example of dynamic programming? Why or why not?

**Each subproblem is unique, so I don’t think it’s dynamic programming?**

d. Consider a memoized version of `binary\_binary\_search\_array\_helper`. Draw the recursion graph for a function memoized `memoized\_binary\_search\_array(3, [1, 3, 4, 6, 7, 8, 10, 13, 14]))` that calls `memoized\_binary\_search\_array\_helper`.

**I… I feel like this makes the function take up more memory without improving performance. The graph would look like the one above, except you hash the result of each call. But on a sorted array you’d only be calling each part once.**

e. Under what circumstances does the `memoized\_binary\_search\_array` present any benefits over the unmemoized original? How does this relate to (i) its recursion graph; (ii) the key attributes of a dynamic program?

**If the array were made out of 2^n smaller identical sorted arrays, the memoized version would be able to (with some tweaks) find every index of a certain value more quickly, since it’d calculate the result for one index and then apply that for each other small identical sorted array, for a speedup on O(n). Otherwise it seems like more memory usage with no time improvement (the additional overhead from setting up the hash and then hashing things is not great).**

f. [Optional challenge] Find the bug in `binary\_search\_array`. What input will cause it to fail? How can this be fixed? (Hint: it is a bug that appears for small arrays. Python is not susceptible to [this bug](<https://research.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html)>.)

**An input that doesn’t exist in the array and is not smaller than the smallest element in the array will cause the array to loop indefinitely. The case where h(middle,right) is called appears to be the cause. This is because middle = int((left + right)/2), and python rounds it down. If right is only 1 smaller than left, middle = int((left + right)/2)) = left, and h(middle,right) is equivalent to h(left,right). This puts the program in an infinite loop.**

## 5. Datalog Tutorial

Dust off your DrRacket, and follow the Datalog tutorial [here](https://docs.racket-lang.org/datalog/Tutorial.html).

## 6. SQL

Install [SQLite](https://www.sqlite.org):

\* \*\*Linux\*\*: `sudo apt-get install sqlite`

\* \*\*macOS\*\* (w/ [homebrew](http://brew.sh)): `brew install sqlite3`

\* \*\*Windows\*\*; \*\*macOS\*\* (w/out homebrew): download the pre-compiled binary from the [SQLite Download Page](https://sqlite.org/download.html)

Kick its tires:

```

$ sqlite3

SQLite version 3.13.0 2016-05-18 10:57:30

Enter ".help" for usage hints.

Connected to a transient in-memory database.

Use ".open FILENAME" to reopen on a persistent database.

sqlite> create table course(title string, number string, area string, credits int);

sqlite> insert into course values ("FOCS", "ENGR3520", "ENGR", 4);

sqlite> insert into course values ("SoftDes", "ENGR2510", "ENGR", 4);

sqlite> insert into course values ("Discrete", "MTH2110", "MTH", 4);

sqlite> .mode column

sqlite> .headers on

sqlite> select \* from course;

title number area credits

---------- ---------- ---------- ----------

FOCS ENGR3520 ENGR 4

SoftDes ENGR2510 ENGR 4

Discrete MTH2110 MTH 4

sqlite> select \* from course where area="ENGR";

title number area credits

---------- ---------- ---------- ----------

FOCS ENGR3520 ENGR 4

SoftDes ENGR2510 ENGR 4

sqlite> .quit

```