CS 3000: Algorithms & Data — Fall 2022

Homework 1
Due Tuesday September 20 at 11:59pm via Gradescope

Name: Briana Torres
Collaborators: Sophia He, Ian Kaish, Jonah Nidorf, Ella Whitmer

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- This homework is due Tuesday September 20 at 11:59pm via Gradescope. No late assignments will be accepted. Make sure to submit something before the deadline.
- Solutions must be typeset. If you need to draw any diagrams, you may draw them by hand as long as they are embedded in the PDF. We recommend that you use LaTeX, in which case it would be best to use the source file for this assignment to get started.
- We encourage you to work with your classmates on the homework problems, but also urge you to attempt all of the problems by yourself first. If you do collaborate, you must write all solutions by yourself, in your own words. Do not submit anything you cannot explain. Please list all your collaborators in your solution for each problem by filling in the `yourcollaborators` command.
- Finding solutions to homework problems on the web, or by asking students not enrolled in the class is strictly forbidden.

Problem 1. *(4 + 8 = 12 points) What does this code do?*

You encounter the following mysterious piece of code.

Algorithm 1: Mystery Function

```
Function F(a, n)):
    If n = 0 :
        Return (1, 0)
    Else
        b ← 0
        For i from 1 to n
            b ← b + a
        (u, v) ← F(a, n − 1)
        Return (u · b, v + n · b)
```

(a) What are the results of $F(a, 2)$, $F(a, 3)$, and $F(a, 4)$?

$$F(a, 2) = (2a^2, 5a)$$

$$F(a, 3) = (6a^3, 14a)$$

$$F(a, 4) = (24a^4, 30a)$$

(b) What does the code do in general, when given input integer $n \geq 0$? Prove your assertion by induction on n.

Solution: Based on part (a), we guess that the code returns $F(a, n) = (n! \cdot a^n, \sum_{i=0}^{n} i^2 \cdot a)$ for all integers n ≥ 0.

Inductive Hypothesis: Let H(n) be the statement: $F(a, n) = (n! \cdot a^n, \sum_{i=0}^{n} i^2 \cdot a)$. We will prove that H(n) is true for every integer n ≥ 0.

Base Case: By the first branch of the if-statement: $F(a, 0) = (1, 0) = (0! \cdot a^0, \sum_{i=0}^{0} 0^2 \cdot 1)$. Thus, H(0) is true.

Inductive Step: We will show that for every $n \geq 1$, $H(n-1) \Rightarrow H(n)$. Assume $H(n-1)$ holds, that is, $F(a, n-1) = (n-1! \cdot a^{n-1}, \sum_{i=0}^{n-1} i^2 \cdot a)$. We can prove our hypothesis by solving for H(n). For $n \geq 1$, b is computed by the for loop to be $n \cdot a$ and (u,v) is $F(a, n-1)$ so therefore (u,v) = $(n-1! \cdot a^{n-1}, \sum_{i=0}^{n-1} i^2 \cdot a)$. Now we can solve for H(n) by plugging in our values into the return statement. Thus we have:

$$F(a,n) = (n-1! \cdot a^{n-1} \cdot n \cdot a, \sum_{i=0}^{n-1} i^2 \cdot a + n - 1 \cdot n \cdot a). \text{ (Inductive Hypothesis)}$$

$$\text{Thus, } F(a,n) = (n! \cdot a^n, \sum_{i=0}^{n} i^2 \cdot a) \checkmark$$

Problem 2. *(12 points) Making exact change*

You live in the country of Binaria where all coins (called binars) are in denomination of powers of 2. You are going out to your favorite store Exact Change to buy yourself a new winter jacket. Every item in the store costs an integer number of binars, but as the name of the store suggests, you can only purchase the item if you pay exactly the cost of the item.

You happen to have exactly one coin of value 2^i, for each i, $0 \leq i < n$ for some positive integer n. Prove using induction that you can purchase a jacket at Exact Change as long its price is strictly less than 2^n. In other words, show that for any $k < 2^n$, there exists a set of coins in your collection whose value adds up to exactly k.

Inductive Hypothesis: We will prove that for any integer k, $k < 2^n$, there exists a set of coins of value 2^i in our collection, for each i, $0 \leq i < n$, whose value add up to exactly k where $n \geq 1$.

Base Case: Let k = 1, n = 1, and i = 0. Now $2^1 = 2$, $k < 2$, and the one coin we have now is of value 2^0 which = k. ✓

Inductive Step: We know $H(n)$ is true according to the base case, now we will prove that $H(n+1)$ is also true. Let i be the highest power of 2 that is less than k, so that the last coin that is able to be used is 2^i. Now let $k = k - 2^i$, which represents the remaining price k' of the item after using the last possible coin of value 2^i. Based on our inductive hypothesis we know that we can make change for anything > 0, and therefore we can assume that k' can always be made in exact change. Proving that for $H(n+1)$ we can make change as it will always be greater than zero as n increases.

Problem 3. *(12 points) More induction practice*

Recall that the Fibonacci numbers are defined by the following recurrence:

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1, \text{ and} \\ F_i &= F_{i-1} + F_{i-2} \text{ for } i \geq 2. \end{aligned}$$

Prove that for any $n \geq 2$, the nth Fibonacci number equals the element in the first row and first column of the 2×2 matrix $A^{n-1}_{1,1}$, where A^n is the nth power of the following 2×2 matrix:

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}.$$

Recall that $A^n = \overbrace{A \times A \times \cdots \times A}^{n \text{ times}}$.

(Hint: Compute A^2, A^3, and A^4 to identify a pattern and a general formula for A^n, and then prove using induction that A^n equals the formula you found.)

Solution:

$$A^2 = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}, \quad A^3 = \begin{pmatrix} 3 & 2 \\ 2 & 1 \end{pmatrix}, \quad A^4 = \begin{pmatrix} 5 & 3 \\ 3 & 2 \end{pmatrix}$$

Inductive Hypothesis: Assuming that $H(n-1)$ is true for $n \geq 2$ we can let $H(n-1)$ be the statement:

$$A^{n-1} = \begin{pmatrix} n & n-1 \\ n-1 & n-2 \end{pmatrix}$$

Base Case: Let $H(n) = 2$, so that $F(0) = 0$, $F(1) = 1$, and $F(2) = 0 + 1$.

$$A^{2-1} = A^1 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Therefore, $F(2) = A^1_{1,1}$ ✓

Inductive Step: We will now prove that for n+1 this solution also proves true:

$$A^{n+1-1} = A^n = \begin{pmatrix} n+1 & n+1-1 \\ n+1-1 & n+1-2 \end{pmatrix} = \begin{pmatrix} n+1 & n \\ n & n-1 \end{pmatrix} \checkmark$$

Problem 4. *(12 points) Growth of functions*

Arrange the following functions in order from the slowest growing function to the fastest growing function. "

$$n^{3/2} \qquad n \lg n \qquad 2^{\sqrt{\lg n}} \qquad n^{1/3}$$

Justify your answers. Specifically, if your order is of the form

$$f_1 \qquad f_2 \qquad f_3 \qquad f_4,$$

you should establish $f_1 = O(f_2)$, $f_2 = O(f_3)$, and $f_3 = O(f_4)$. For each case, your justification can be in the form of a proof from first principles or a proof using limits, and can use any of the facts presented in the lecture or the text. (Hint: It may help to plot the functions and obtain an estimate of their relative growth rates. In some cases, it may also help to express the functions as a power of 2 and then compare.)

Solution:

$$f_1 = 2^{\sqrt{\lg n}} \qquad f_2 = n^{1/3} \qquad f_3 = n \lg n \qquad f_4 = n^{3/2}$$

$2^{\sqrt{\lg n}} = O(n^{1/3})$

$$\lim_{n\to\infty} \frac{2^{\sqrt{\lg n}}}{n^{1/3}} = \lim_{n\to\infty} \frac{2^{\lg n^{1/2}}}{n^{1/3}} = \lim_{n\to\infty} \frac{2^{\lg n^{1/2}}}{2^{\lg n^{1/3}}} = \frac{1}{2^{\lg n^{1/2} + \lg n^{-1/2}}} = \frac{1}{\infty} = 0 \checkmark$$

$n^{1/3} = O(n \lg n)$

$$\lim_{n\to\infty} \frac{n^{1/3}}{n \lg n} = \lim_{n\to\infty} \frac{1}{n^{2/3} \log n} = \frac{1}{\infty} = 0 \checkmark$$

$n \lg n = O(n^{3/2})$

We can prove this according to the Big-Oh rules given in lecture that state that any logarithm is Big-Oh of any polynomial. Therefore, $n \lg n = O(n^{3/2})$ ✓

Problem 5. *(2 × 5 = 10 points) Properties of asymptotic notation*

Let $f(n)$, $g(n)$, and $h(n)$ be asymptotically positive and monotonically increasing functions.

(a) Using the formal definition of the O and Ω notation, prove that if $f(n) = O(h(n))$ and $g(n) = \Omega(h(n))$, then $f(n) = O(g(n))$.

According to the formal definitions of O and Ω:
We know that $f(n) \leq C_1 h(n)$ for all $n \geq n_0^{'}$ and $g(n) \geq C_2 h(n)$ for all $n \geq n_0^{''}$
Therefore, $\frac{f(n)}{C_1}$ for all $n \geq n_0^{'}$
Now let $n_0 = max(n_0^{'}, n_0^{''})$, so that $g(n) \geq C_2 h(n) \geq h(n) \geq \frac{1}{C^1} f(n)$
Let $C_3 = C_1 + C_2$, so that $g(n) \geq C_2 h(n) \geq h(n) \geq C_3 f(n)$ for all $n \geq n_0$
So finally we see that $g(n) \geq C_3 f(n)$ for all $n \geq n_0$, proving that $f(n) = O(g(n))$ ✓

(b) Give distinct functions f and g satisfying both $f(n) = \Theta(g(n))$ and $2^{f(n)} = \Theta(2^{g(n)})$.
Give distinct functions f and g satisfying $f(n) = O(g(n))$ yet $2^{f(n)} \neq O(4^{g(n)})$.

For (b.1), Let $f(n) = \log n$ and $g(n) = 2\log n$:

$$\lim_{n\to\infty} \frac{\log n}{2\log n} = \lim_{n\to\infty} \frac{1}{2} = \frac{1}{2} < \infty \checkmark$$

For (b.1), Let $2^{f(n)} = 2^{\log n}$ and $2^{g(n)} = 2^{2\log n}$:

$$\lim_{n\to\infty} \frac{2^{\log n}}{2^{2\log n}} = \lim_{n\to\infty} \frac{1}{n} = 0 < \infty \checkmark$$

For (b.2), Let $f(n) = n + 2$ and $g(n) = \frac{n}{8}$:

$$\lim_{n\to\infty} \frac{n+2}{\frac{n}{8}} = \lim_{n\to\infty} 8 = 8 < \infty \checkmark$$

For (b.2), Let $2^{f(n)} = 2^{n+2}$ and $4^{g(n)} = 4^{\frac{n}{8}}$:

$$\lim_{n\to\infty} \frac{2^{n+2}}{4^{\frac{n}{8}}} = \frac{1}{4^{\frac{n}{8}}} \lim_{n\to\infty} 2^{n+2} = \frac{1}{2^{\frac{n}{4}}} \lim_{n\to\infty} 2^{n+2} = \frac{1}{2^{\frac{n}{4}}} \lim_{n\to\infty} e^{(n+2)(\ln 2)} = \frac{1}{2^{\frac{n}{4}}} \cdot \infty = \infty \checkmark$$

Because the limit goes to infinity it does not comply with the formal definition of Big-Oh and therefore, b.2 has been proven incorrect

Problem 6. *(12 points) Determining the smallest element in one list that is not present in another list*

We have learned that Mergesort sorts an array of n numbers in $O(n \log n)$ time and Binary Search determines if a given number is present in a sorted array of n numbers in $O(\log n)$ time.

Describe an $O(n \log n)$ time algorithm that takes as input two arrays A and B with n elements each (not necessarily sorted) and determines the smallest number in B that is not in A. If all elements of B are in A, then return "Identical Arrays".

Your algorithm should use mergesort and binary search and should not use hash tables. Give your algorithm in pseudocode. Justify the running time of your algorithm.

Solution:

Algorithm 2: Sorting Function

```
Function F(A, B)):
    MergeSort(A)
    MergeSort(B)
    For i = 1, ···, len(B)
        If BinarySearch(A, B[i], 1, len(A)) == -1 :
            Return B[i]
    Return "Identical Arrays"
```

Running Time:
The algorithm will run in $O(n \log n)$ time as seen when we calculate the total run time of the algorithm. For each call MergeSort the run time is $O(n \log n)$ and since we call MergeSort on both arrays the total run time for both calls is $O(2(n \log n))$. In the for loop BinarySearch is called and it's run time is $O(\log n)$ but since we recur through an array of length n, the runtime becomes $O(n \log n)$. When we add those run times together we find that the total run time is $O(3(n \log n))$, and because the constant can be ignored we have proven that the run time is $O(n \log n)$.