CS 3000: Algorithms & Data — Spring 2024

Sample Solutions to Homework 1
Due Monday January 22 at 11:59pm via Gradescope

Name: Ryan Tietjen
Collaborators: Please note, I used Mathway/ChatGPT to help format my answers in latex. These tools were not used to obtain any answers, solely for formatting. (I have never used latex before and these tools were convient for learning it.) I do not believe this should violate academic integrity, nor do I intend to do so.

- 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 Monday January 22 at 11:59pm via Gradescope. No late assignments will be accepted. Make sure to submit something before the deadline.
- Solutions must be typed. 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. Your submitted file must be a PDF file.
- 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(n):
    If n = 0 :
        Return (2, 1)
    Else
        b ← 1
        For i from 1 to n
            b ← 2b
        (u, v) ← F(n − 1)
        Return (u + b, v · b)
```

(a) What are the results of $F(1)$, $F(2)$, $F(3)$, and $F(4)$?

Solution: $F(1) = (4, 2); F(2) = (8, 8); F(3) = (16, 64); F(4) = (32, 1024)$

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

Solution: For all integers $n \geq 0$, $F(n) = \left(2^{n+1}, 2^{\frac{n(n+1)}{2}}\right)$

Proof by induction on n:

Inductive Hypothesis: Let $H(n)$ be the statement: $F(n) = \left(2^{n+1}, 2^{\frac{n(n+1)}{2}}\right)$ for all integers $n \geq 0$

Base Case: By the first branch of the `if-statement`, $F(0) = (2, 1) = \left(2^{0+1}, 2^{\frac{0(0+1)}{2}}\right) = \left(2^1, 2^0\right)$

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, $F(n-1) = \left(2^n, 2^{\frac{n(n-1)}{2}}\right)$. For $n \geq 1$, b is computed to be 2^n.

Then, we have the Inductive Hypothesis

$$(u, v) = F(n-1) = \left(2^n, 2^{\frac{n(n-1)}{2}}\right)$$

Thus,

$$F(n) = \left(2^n + 2^n, 2^{\frac{n(n-1)}{2}} 2^n\right) = \left(2^{n+1}, 2^{\frac{n(n+1)}{2}}\right)$$

thus establishing $H(n)$. Therefore, $H(n)$ is true for all integers $n \geq 0$ by induction.

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

In the country of Perfect Squares, all coins are in denominations that are perfect squares i.e. i^2 for some integer i. You need to buy a jacket whose price is an integer $n \geq 1$. The country is obsessed with being perfect and you can only buy an item if you pay the exact price. You happen to have exactly one coin of value i^2 for each integer $i \geq 2$. Additionally you have 4 coins of value 1 each. Use induction to show that you can pay the exact cost of the jacket using your coins.

Solution:

Inductive Hypothesis: Let H(n) be the statement: It is possible to buy a jacket whose price is an integer $n \geq 1$, given that you have exactly one coin of value i^2 for each integer $i \geq 2$ and four coins of value 1 each.

Base Case:

$H(1)$: one 1-value coin is sufficient.
$H(2)$: two 1-value coins are sufficient.
$H(3)$: three 1-value coins are sufficient.
$H(4)$: four 1-value coins are sufficient.
So, H(n) holds for $1 \leq n \leq 4$.

Inductive step: We will show that for every $n \geq 4$, $H(n) \Rightarrow H(n+1)$. Assume $H(n)$ holds.

Case $n+1$ is a perfect square: Then, there exists a coin i^2 where $i^2 = n+1$. So, the jacket can be bought using only the coin of value i^2. Thus $H(n+1)$ holds.

Case $n+1$ is not a perfect square: Then, let i^2 be the largest coin such that $i^2 < n+1$. So, using the coin of i^2 value, the remaining price of the jacket can be represented by $n+1-i^2$. Since $i^2 \geq 1$, $n+1-i^2 \leq n$. In other words, the remaining price of the jacket is less than or equal to n. By the inductive hypothesis $H(n)$, any jacket up to value n can be purchased. Therefore, by combining the coin valued i^2 with the coins that value $n+1-i^2$, any jacket up to value $n+1$ can be purchased. Thus, $H(n+1)$ holds.

In both cases, $H(n+1)$ is true. Therefore, $H(n)$ is true for all integers $n \geq 1$

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

Consider the following function f defined on the nonnegative integers.

$$f(0) = 3$$
$$f(1) = 4$$
$$f(n) = 3f(n-2) + 2f(n-1), \text{ for } n \geq 2$$

Prove by induction that $f(n) = (5 \cdot (-1)^n + 7 \cdot 3^n)/4$ for all integer $n \geq 0$.

Solution:

Inductive Hypothesis: Let $H(n)$ be the statement: $f(n) = (5 \cdot (-1)^n + 7 \cdot 3^n)/4$ for all integers $n \geq 0$

Base Case: $f(0) = \frac{5\cdot(-1)^0+7\cdot 3^0}{4} = \frac{12}{4} = 3$; $f(1) = \frac{5\cdot(-1)^1+7\cdot 3^1}{4} = \frac{16}{4} = 4$. Thus, $H(0)$ and $H(1)$ are true

Inductive Step: We will show that for every $n \geq 1$, $H(n-1)$ and $H(n) \Rightarrow H(n+1)$. Assume $H(n)$ and $H(n-1)$ hold, that is $f(n) = \frac{5(-1)^n+7\cdot 3^n}{4}$ and $f(n-1) = \frac{5(-1)^{n-1}+7\cdot 3^{n-1}}{4}$

Then, we have the Inductive Hypothesis:

$$f(n+1) = \frac{5(-1)^{n+1} + 7 \cdot 3^{n+1}}{4}$$

Also, by the definition of $f(n)$, we have

$$f(n+1) = 3f(n-1) + 2f(n) = \frac{3(5(-1)^{n-1} + 7 \cdot 3^{n-1})}{4} + \frac{2(5(-1)^n + 7 \cdot 3^n)}{4}$$

Simplify, note that $(-1)^n = -(-1)^{n-1}$, $(-1)^{n-1} = (-1)^{n+1}$ and $3^n = 3(3)^{n-1}$:

$$f(n+1) = \frac{15(-1)^{n-1} + 21 \cdot 3^{n-1} + 10(-1)^n + 14 \cdot 3^n}{4} = \frac{15(-1)^{n-1} - 10(-1)^{n-1} + 63 \cdot 3^{n-1}}{4}$$

$$= \frac{5(3(-1)^{n-1} - 2(-1)^{n-1}) + 3 \cdot 3 \cdot 7 \cdot 3^{n-1}}{4} = \frac{5(-1)^{n-1} + 3 \cdot 7 \cdot 3^n}{4} = \frac{5(-1)^{n+1} + 7 \cdot 3^{n+1}}{4}$$

thus establishing $H(n+1)$. Therefore, $H(n)$ is true for all integers $n \geq 0$

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

Arrange the following functions in order from the slowest growing function to the fastest growing function. Note that $\lg n = \log_2 n$.

$$n^{2/3} \qquad n + \lg n \qquad 2^{\sqrt{\lg n}} \qquad (\lg n)^{\lg n}$$

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:

In order from the slowest growing function to the fastest growing function:
let $f_1(n) = 2^{\sqrt{\lg n}}$, $f_2(n) = n^{2/3}$, $f_3(n) = n + \lg n$, $f_4(n) = (\lg n)^{\lg n}$

To show $f_1 = O(f_2)$, we must show that $\lim_{n\to\infty} \frac{2^{\sqrt{\log_2 n}}}{n^{\frac{2}{3}}} < \infty$.
Rearrange:

$$2^{\sqrt{\log_2 n}} = n^{\frac{\sqrt{\log_2 n}}{\log_2 n}}$$

Then, we have

$$\lim_{n\to\infty} \frac{n^{\frac{\sqrt{\log_2 n}}{\log_2 n}}}{n^{\frac{2}{3}}} = \lim_{n\to\infty} n^{\left(\frac{\sqrt{\log_2 n}}{\log_2 n} - \frac{2}{3}\right)}$$

But since $\sqrt{\log_2 n}$ will increase at a slower rate than $\log_2 n$,

$$\lim_{n\to\infty} n^{\left(\frac{\sqrt{\log_2 n}}{\log_2 n} - \frac{2}{3}\right)} < \infty$$

as required. Hence, $f_1 = O(f_2)$

To show $f_2 = O(f_3)$, we must show that $\lim_{n\to\infty} \frac{n^{\frac{2}{3}}}{n+\log_2 n} < \infty$
Since $2/3 < 1$, $n^{2/3}$ increase at a slower rate than n^1
Also, $\log_2 n > 0$ for $n > 1$
Therefore, $\lim_{n\to\infty} \frac{n^{\frac{2}{3}}}{n+\log_2 n} < \infty$
Hence, $f_2 = O(f_3)$

To show $f_3 = O(f_4)$, we must show $\lim_{n\to\infty} \frac{n+\log_2 n}{(\log_2 n)^{\log_2 n}} < \infty$
Let $a = \log_2 n$ and rearrange $\frac{n+\log_2 n}{(\log_2 n)^{\log_2 n}}$ to get $\frac{2^a+a}{a^a}$
Since 2^a dominates the numerator, and since a^a increases faster than 2^a, $\lim_{n\to\infty} \frac{2^a+a}{a^a} < \infty$
Hence, $f_3 = O(f_4)$

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 $g(n)^2 = \Omega(h(n)^2)$, then $f(n) = O(g(n))$.

Solution:

Assume $f(n) = O(h(n))$ and $g(n)^2 = \Omega(h(n)^2)$ are true.

Since $f(n) = O(h(n))$, there exists some constants c_1 and n_1 such that $f(n) \leq c_1 \cdot h(n)$ for every $n \geq n_1$

Since $g(n)^2 = \Omega(h(n)^2)$, there exists some constants c_2 and n_2 such that $g(n)^2 \geq c_2 \cdot h(n)^2$ for every $n \geq n_2$

To prove $f(n) = O(g(n))$, we must show that there exists some constants c_3 and n_3 such that $f(n) \leq c_3 \cdot g(n)$ for every $n \geq n_3$

By taking the square root of both sides of $g(n)^2 \geq c_2 \cdot h(n)^2$, we get

$$g(n) \geq \sqrt{c_2} \cdot h(n)$$

Since $f(n) \leq c_1 \cdot h(n)$, we can substitute $\frac{f(n)}{c_1}$ for $h(n)$

$$g(n) \geq \sqrt{c_2} \cdot \frac{f(n)}{c_1} = g(n) \geq \frac{\sqrt{c_2}}{c1} \cdot f(n)$$

Replace $\frac{\sqrt{c_2}}{c1}$ with some constant c_3, and assume n_3 is the maximum value of n_1 and n_2:

$$g(n) \geq c_3 \cdot f(n)$$

So, we have $f(n) \leq c_3 \cdot g(n)$ for some constant c_3 and for every $n \geq n_3$ as required.

Hence, $f(n) = O(g(n))$

(b) Give distinct functions f and g satisfying both $f(n) = \Theta(g(n))$ and $3^{f(n)} = \Theta(3^{g(n)})$.

Give distinct functions f and g satisfying $f(n) = O(g(n))$ yet $3^{f(n)} \neq O(27^{g(n)})$.

Solution:

part 1: $f(n) = n$ and $g(n) = n + 1$

Since constants can be ignored, $f(n) = \Theta(g(n))$ implies $n = \Theta(n)$ as required.

Also, since constants can be ignored $3^n = \Theta(3^n)$ as required.

part 2: $f(n) = \log_3 n$ and $g(n) = \log_3(n + 1)$

Since constants can be ignored, $f(n) = O(g(n))$ implies $\log_3 n = O(\log_3 n)$ as required.

However, $3^{f(n)} = 3^{\log_3 n} = n$, but $27^{\log_3 n} = 3^{3(\log_3 n)} = n^3$. Since n is linear and n^3 is polynomial, $n \neq O(n^3)$, meaning that $3^{f(n)} \neq O(27^{g(n)})$

Problem 6. *(12 points) Determining the largest 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 largest number in B that is not in A. If all elements of B are in A, then return "All elements of B are in A".

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:

Used functions:

$MergeSort(A)$ takes in an array of numbers, performs merge sort on the array, and returns the array sorted by values ascending.

$BinarySearch(num, A)$ takes in an number and a sorted array, performs binary search, and returns the index of number if it is found in the array or -1 if the number cannot be found.

Algorithm 2: Finds the Largest number in Array B that is not in Array A

```
Function FindLargestNumInBNotInA(A, B)):
    sortedA = MergeSort(A)
    highestNum = NULL
    For i from 0 to length of B:
        If BinarySearch(B[i], A) ≠ −1 :
            If highestNum == NULL or B[i] ≥ highestNum :
                highestNum = B[i]
    If highestNum == NULL :
        Return "All elements of B are in A"
    Return highestNum
```

Time complexity justification:

-Sorting A with Mergesort takes $O(n \log n)$ time

-The for loop runs n times

-Inside the loop, a Binary Search is performed, taking $O(\log n)$ time

-Thus, the for loop takes a cumulative $O(n \log n)$ time.

Since both Mergesort and the for loop take $O(n \log n)$ time, the time complexity of the function is $O(n \log n)$