CS 330, Fall 2023, Homework 1

Due: Wednesday 9/13 at 11:59 pm on Gradescope

Collaboration policy Collaboration on homework problems is permitted, you are allowed to discuss each problem with at most 3 other students currently enrolled in the class. Before working with others on a problem, you should think about it yourself for at least 45 minutes. Finding answers to problems on the Web or from other outside sources (these include anyone not enrolled in the class) is strictly forbidden.

You must write up each problem solution by yourself without assistance, even if you collaborate with others to solve the problem. You must also identify your collaborators. If you did not work with anyone, you should write "Collaborators: none." It is a violation of this policy to submit a problem solution that you cannot orally explain to an instructor or TA.

Typesetting Solutions should be typed and submitted as a PDF file on Gradescope. You may use any program you like to type your solutions. LaTeX, or "Latex", is commonly used for technical writing (`overleaf.com` is a free web-based platform for writing in Latex) since it handles math very well. Word, Google Docs, Markdown or other software are also fine.

1. Asymptotic (10 points)

Part 1

For the following pairs of functions, clearly state whether $f(n) = O(g(n))$, $g(n) = O(f(n))$, both or neither. Justify why this is true:

(a) $f = \binom{n}{5}$, $g = n^4$

(b) $f(n) = (h(n)/3)$, $g(n) = h(n/3)$, where $h(n) = 3^n$.

(c) $f(n) = 4^{\log n}$, $g(n) = n^4$

(d) $f(n) = 1.1^{n-1} + \sqrt{n} + log(n^2)$, $g(n) = n^{1.5}$

(e) $f = x^2 \sin x$, $g = x$

Part 2

Prove that $\forall a \in \mathbb{Z}^+$,

$$\log n \text{ is } O(n^{1/a})$$

using the combinatoric definition

You may use the fact that $\log x < x$ from calculus for all $x \in \mathbb{R}^+$ without proof.

2. Algorithm Analysis (10 points)

You are given an array B with n elements. We are also given an integer r *as part of the input.* (Remember, this means that you cannot treat it as a constant!) Help us design an algorithm to compute an array A such that the following property holds:

$$A[i] = B[i] + B[i+1] + B[i+2] ... + B[i+r-1]$$

for all i that $0 \leq i \leq n - r$.

In this problem we look at two ways of implementing this algorithm, the first being more straightforward, the second being an order of magnitude faster.

(a) (do not hand in) What is the length of array A? (use n and r.)

(b) Complete the missing parts of SlowFun [1] in Algorithm 1, such that the algorithm has $O(nr)$ iterations.

```
Algorithm 1: SlowFun (B, r)
  /* B is an integer array, r is an integer                */
1 n ← length(B);
2 A ←array of 0s of appropriate length;
3 for i from [      ] to [      ] do
4     for j from [      ] to [      ] do
5         A[i] ← [                ];
6 return A
```

(c) Complete the missing parts of FastFun in Algorithm 2, such that the algorithm has $O(n)$ iterations.

Note that line 3 doesn't add to the time complexity as $r \leq n$.

```
Algorithm 2: FastFun (B, r)
  /* B is an integer array, r is an integer                */
1 n ← length(B);
2 A ←array of 0s of appropriate length;
3 A[0] ← B[0] + B[1] + ... B[r − 1]/* runtime of this line is O(r)  */
4 for i from [      ] to [      ] do
5     A[i] ← [                ];
6 return A
```

[1] There are some comments in the tex file that can help you with formatting if you are using LaTeX.

(d) It is easy to see that SlowFun correctly computes the values of A, however it's not as obvious for FastFun. Use induction to formally prove that FastFun correctly computes the values $A[i]$ for every i.