CS 330, Spring 2025, Homework 1
Due Wednesday, September 10th, 11:59 pm EST, via Gradescope

Homework Guidelines

Collaboration policy If you choose to collaborate on some problems, 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.

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.

David Bunger, Collaborators: None

Problem 1: Asymptotics

For the following pairs of functions, clearly state whether $f(n) = O\big(g(n)\big)$, $g(n) = O\big(f(n)\big)$, both or neither. Prove your answer by showing that either the combinatoric or analytic definition applies:

a) $f(n) = n^{1.1}$ and $g(n) = \sqrt{2n}$

b) $f(n) = 2^n + n$ and $g(n) = n^3$

c) $f(n) = \log_2(n)$ and $g(n) = \log_2(n^2 + n)$

d) $f(n) = n\log_2(n)$ and $g(n) = n \cdot \sqrt{n}$
You can use the fact that $\lim_{n\to\infty} \frac{\log_2(n)}{\sqrt{n}} = 0$

Hint1: Recall that if $f(n) = O(g(n))$ and $g(n) = O(f(n))$ then $f(n) = \Theta(g(n))$.
Hint2: Recall the combinatoric and analytic definitions of the asymptotic notations.

- $f(n) = O(g(n))$ means: there exist constants $c > 0$ and n_0 such that $\forall n \geq n_0, \quad f(n) \leq c{\cdot}g(n)$.
- $f(n) = \Omega(g(n))$ means: there exist constants $c > 0$ and n_0 such that $\forall n \geq n_0, \quad f(n) \geq c{\cdot}g(n)$.
- $f(n) = \Theta(g(n))$ means: there exist constants $c_1, c_2 > 0$ and n_0 such that $\forall n \geq n_0$, $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$.

Equivalently, in analytic terms:

- If $\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$, then $f(n) = O(g(n))$.
- If $\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$, then $f(n) = \Omega(g(n))$.
- If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c > 0$, then $f(n) = \Theta(g(n))$.

Solution: Your solution to Problem 1 here.

a) The limit to infinity of the ratio $\frac{f(n)}{g(n)}$, or $\lim_{n\to\infty} \frac{n^{1.1}}{\sqrt{2n}} = \infty$, as both the numerator and denomonator are polynomials, and the power of n in the numerator(1.1) is greater than the power of n in the denomonator(1/2). This proves that $f(n) \neq O(g(n))$.
This same logic can be used to show that, because $lim_{n\to\infty} \frac{g(n)}{f(n)}, \lim_{n\to\infty} \frac{\sqrt{2n}}{n^{1.1}} = 0 < \infty$, $g(n) = O(f(n))$ is true.

b) We know that the rate of growth for exponential functions is higher than that of polynomial functions. Because $f(n)$ is an exponential function and $g(n)$ is a polynomail function, we can conclude that $f(n) \neq O(g(n))$ and $g(n) = O(f(n))$. This can also be shown by taking the limit of the ratio between the functions: $\lim_{n\to\infty} \frac{2^n+n}{n^3} = \infty, \lim_{n\to\infty} \frac{n^3}{2^n+n} = 0 < \infty$.

c) To take the limit to infinity of $\frac{f(n)}{g(n)}$, or $\lim_{n\to\infty} \frac{\log_2(n)}{\log_2(n^2+n)}$, we can use L'Hopital's Rule, since $\lim_{n\to\infty} \log_2(n) = \infty$ and $\lim_{n\to\infty} \log_2(n^2+n) = \infty$.

$\frac{f'(n)}{g'(n)} = \frac{(\log_2(n))'}{(\log_2(n^2+n))'} = \frac{\frac{1}{n\ln 2}}{\frac{2n+1}{(n^2+n)\ln 2}} = \frac{(n^2+n)\ln 2}{n(2n+1)\ln 2} = \frac{n^2+n}{2n^2+n} = \frac{1}{2}$

Because $\lim_{x\to\infty} \frac{f(n)}{g(n)} = \frac{1}{2} < \infty$, $f(n) = O(g(n))$.

Taking the limit to infinity of $\frac{g(n}{f(n)}$ can be done in a similar way.

$\frac{g'(n)}{f'(n)} = \frac{(\log_2(n^2+n))'}{(\log_2(n))'} = \frac{\frac{2n+1}{(n^2+n)\ln 2}}{\frac{1}{n\ln 2}} = \frac{n(2n+1)\ln 2}{(n^2+n)\ln 2} = \frac{2n^2+n}{n^2+n} = 2$

Because $\lim_{x\to\infty} \frac{g(n)}{f(n)} = 2 < \infty$, $g(n) = O(f(n))$.

d) The ratio $\frac{f(n)}{g(n)}$, or $\frac{n\log_2(n)}{n\sqrt{n}}$ can be simplified to $\frac{\log_2(n)}{\sqrt{n}}$. Because $\lim_{x\to\infty} \frac{\log_2(n)}{\sqrt{n}} = 0 < \infty$, $f(n) = O(g(n))$ is true. Likewise, the ratio $\frac{g(n)}{f(n)}$, or $\frac{n\sqrt{n}}{n\log_2(n)}$ can be simplified to $\frac{\sqrt{n}}{\log_2(n)}$. Because $\lim_{x\to\infty} \frac{\log_2(n)}{\sqrt{n}} = \infty, g(n) \neq O(f(n))$.

Problem 2: Algorithm Analysis

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.

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

2. Complete the missing parts of SlowFun [1] in Algorithm 3, 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
```

3. Complete the missing parts of FastFun in Algorithm 4, 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.

4. 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.

Solution: Your solution to Problem 2 here. Feel free to copy-paste the algorithm environments above and replace the answer boxes with your solution.

2.

Algorithm 3: 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 0 to (n-r) do
4 |   for j from i to i+r-1 do
5 |   |   A[i] ← A[i] + B[j] ;
6 return A
```

3.

Algorithm 4: 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 1 to (n-r) do
5 |   A[i] ← A[i − 1] − B[i − 1] + B[i + r − 1] ;
6 return A
```

4. Base Case ($A[1]$): In this case, line 5 is: $A[1] \leftarrow A[0] - B[0] + B[r]$. This results in $A[1] = B[1] + B[2] + B[3] ... + B[r]$. The defining property of the algorithm holds for this result, meaning FastFun correctly computes the value of $A[1]$.
Induction Hypothesis: Assume that FastFun correctly computes the value of $A[i]$, where i is an integer in the range $1 \leq i \leq (n - r)$. This would mean $A[i] = B[i] + B[i + 1] + B[i + 2] ... + B[i + r - 1]$, based on the definition of the algorithm.
Inductive Step: For $A[i+1]$ to be considered correctly computed, $A[i+1] = B[i+1] + B[i+2] + B[i+3] ... + B[i+r]$, based on the definition of the algorithm. The difference between $A[i+1]$ and $A[i]$ is that $A[i]$ includes the value of $B[i]$, but does not include the value of $B[i+r]$.
During FastFun, the following is performed: $A[i+1] \leftarrow A[i] - B[i] + B[i+r]$. Because $A[i]$ is assumed to be correctly computed, $A[i+1]$ must also be correctly computed.
Since the inductive step is valid for all k in the range $1 \leq i \leq (n - r)$, and $A[1]$ $(i = 1)$ has been proved to be correctly computed, the statement that FastFun correctly computes the values of $A[i]$ for every i is true by induction.