CSCI 432 Handout 09: Analyzing an Algorithm

Name: _____

5 March 2026

What algorithm are you analyzing today?

Answer

1 What?

When analyzing algorithms, we first ask WHAT? That is, what is the problem that we want solved? Typically, in this description, we need to understand both what are the inputs *and* what are the outputs of the algorithm. When describing WHAT, it should be independent of HOW.

So, describe the WHAT for the algorithm you wish to analyze today.

What is the input?

Answer

What is the output?

Answer

Use the previous two answers to concisely describe what is the problem? This often requires explaining the output in terms of the input.

Answer

2 How?

Often, it helps to walk through small examples. If you came up with the algorithm, this is a nice sanity check. If you are studying an algorithm given to you, this is helpful to better understand the algorithm. Do that now.

Answer

If you are presenting an algorithm, you should describe how it works in pseudocode. More importantly, explain what is going on in the pseudocode in the prose text. The prose should point to the algorithm. (Remember, in LaTex, algorithms are floating environments, so we need to be reminded to look at them). If you are working on analyzing an algorithm that is given to you in pseudocode, briefly explain HOW it works here. For now ignore base cases and focus on, *what is the key step*?

Answer

3 How Fast?

Option A: What is the runtime? If this is a recursive algorithm, you are expected to explicitly state the recurrence relation.

Answer

Option B: Sometimes, it is hard to pin down the runtime, yet we still want to ensure that the algorithm terminates. For this, we use decrementing functions to prove that loops/recursions terminate. For the next few questions, try to use a decrementing function to prove termination.

1. What is the decrementing function for this algorithm?
   Try it!

2. Justify (=prove) why function is well-defined. That is, why is the output of the function a natural number?
   Try it!

3. Justify (=prove) why this decrements each time it reaches the top of the loop (or each time it goes into a new recursive call).
   Try it!

4 Why? The Loop Invariant

Finally, we prove why the algorithm works. Typically, this is done with a loop (or recursion) invariant. Here, we focus on setting up the loop invariant. What are the following statements (start with your best guess, then come back and revise if needed):

The loop guard G:

The post-condition Q:

The pre-condition P:

The loop invariant $L = L_i$:

Next, we use use this loop invariant (and other statements above) to prove partial correctness There are three parts to partial correctness: Initialization, Maintenance, and End.

Initialization: If the preconditions are met and if we enter the loop, then the loop invariant is correct. $(P \wedge G \implies L)$.

Answer

Maintenance: If we entered the loop and the loop invariant held, then when we exit the loop, the loop invariant will still hold: $(G \wedge L_i \implies L_{i+1})$.

Answer

End: If the loop ends and the loop invariant was correct, then the algorithm is correct. $(L \wedge \neg G \implies Q)$.

Answer