CSC 656 – Foundations of Computer Science – Fall 2019
Assignment 2 – Problem listing (due Wednesday, September 25)

Remember to provide full solutions. Proofs should be complete, and computation/analysis problems should always show work or justification (never just state the final answer!).

1. In this problem, you have a robot that moves on a grid, similar to the robot in Section 6.2.1, but with different state transitions. In particular, a robot at position (x, y) can more to square (y, x) or to square $(x + 1, y - 4)$. The robot starts at square $(10, 6)$, and you need to determine if it is possible for the robot to get to square $(17, 10)$.

   (a) Find a useful invariant that is preserved for this state machine (hint: it is similar to examples we did in class, but slightly more complex).

   (b) Can the robot get to $(17, 10)$? Explain your answer.

2. Textbook Problem 7.1 (page 258).

3. Textbook Problem 7.25 (page 278).

4. In programming languages, we can type expressions that are arithmetic (i.e., produce numerical results) or boolean (produce true/false results). In the textbook, the authors developed a recursive datatype "Aexp" that was a simplied version of an arithmetic expression. For this problem, you are to expand on this to add a simple form of boolean expression. And boolean expressions can be made out of arithmetic expressions!

   (a) Define a "Bexp" data type that includes boolean constants (`true` and `false`), a boolean "OR" operation, and also a "$\leq$" comparison of two Aexp's (in other words, something of the form "Aexp $\leq$ Aexp" is a Bexp). Define this similarly to how Aexp's are defined in Definition 7.4.1 of the book.

   (b) Patterned after Definition 7.4.2, define an "evalb" function for evaluating Bexp's. You can use the "eval" function from the book for Aexp's (you don't have to repeat the definition — you can just use the function).

   (c) Patterned after Definition 7.4.3, define a "substb" function function for Bexp's.

5. When learning induction proofs, summations are commonly used as examples, where the value of the sum is given as a formula, and then an induction proof is used to prove that it is correct. This can also be flipped around: If you know what the basic form of a formula for a summation value, then you can use an induction proof to derive the precise formula.

   In this problem, you are interested in finding a formula for $\sum_{i=1}^{n} i^2$, for $n \geq 0$. You believe that there is a formula of the form $an^3 + bn^2 + cn + d$ for this, where a, b, c, and d are constants. You're just not sure what values they should be.

   (a) Write out a standard proof by induction, to prove that the summation has value $an^3 + bn^2 + cn + d$ — keep everything as variables for now, and just assume everything works out properly.

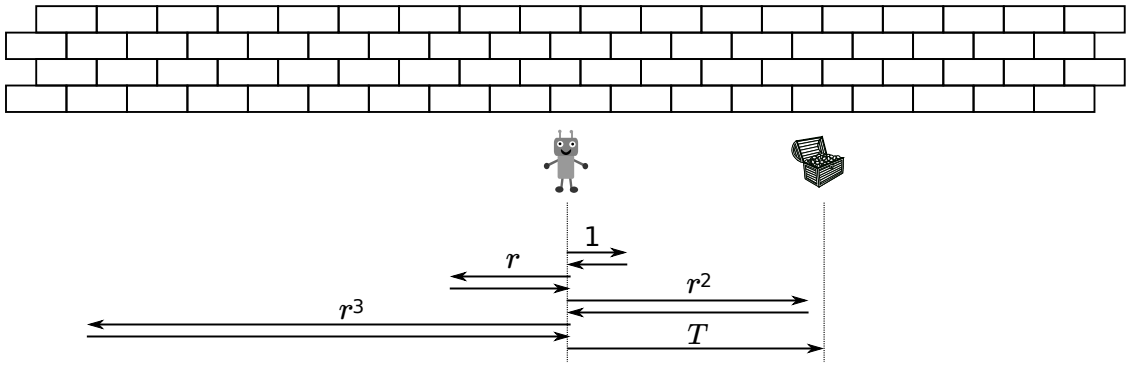
(b) Identify conditions that must be met by a, b, c, and d for your proof to work out. For example, just the base case will determine the value for d.

(c) Solve for a, b, c, and d, and write out the resulting formula for $\sum_{i=1}^{n} i^2$.

Side note: There is an easier way to find a, b, c, and d for this particular problem, but as a demonstration of this technique you are required to find them as described above!

6. You have been hired to write a program for a robot that is searching for treasure. The treasure is buried next to a really long wall that stretches east and west, and your robot starts in the middle of the wall. The robot can detect when it reaches the treasure, but gets no other information. The robot can move east any distance it wants, or west any distance it wants, and the goal is to find the treasure as fast as possible. If you knew the treasure was to the east, you'd just travel east until you found it. But if the treasure were in the other direction, when do you give up and turn around?

   Consider this algorithm, with unknown parameter $r > 1$: You start by traveling $r^0 = 1$ foot east, searching for the treasure. If you don't find the treasure, you turn around and travel back to your starting point, and then go r feet west. If you again don't find the treasure, return to the starting point and search r^2 feet east. In general, if you fail in your search at distance r^d, you return to the starting point and search distance r^{d+1} in the other direction. This is illustrated below.



   (a) The worst case is when the robot just barely misses the treasure on one search, and doesn't find it until the next search. In other words, for some small value ϵ, the treasure is at $T = r^k + \epsilon$, and you just miss it when searching at distance r^k. In this case you end up making an unsuccessful search in the other direction of distance r^{k+1} before turning around and finding the treasure. What is the total distance traveled by the robot in this case?

   (b) (*This part requires some basic Calculus. If you are rusty in Calculus, feel free to come see me to discuss this part.*) If $D(r)$ is the distance traveled (calculated in part (a)), then we are interested in minimizing the fraction $\frac{D(r)}{T}$ as T gets large. Use your formula for D from part (a), take the limit as T goes to infinity, and find the value of r that minimizes this fraction.

   (c) Give your optimal r from part (b), what is the value of $\frac{D(r)}{T}$ for this optimal search algorithm?