Lab 1 Solutions

Math Review Problem 1. Which of the following functions are increasing? eventually nondecreasing? If you remember techniques from calculus, you can make use of those.

(1) $f(x) = -x^2$

(2) $f(x) = x^2 + 2x + 1$

(3) $f(x) = x^3 + x$

Solution. (1) is not eventually nondecreasing. (2) is eventually nondecreasing. (3) is increasing

Math Review Problem 2. Compute the following limits at infinity:

(1) $\lim_{n\to\infty} (2n^2 + 3n)/(n^3 - 4)$.

(2) $\lim_{n\to\infty} n^2/2^n$.

Solution. Use repeated application of L'Hopital's rule.

Math Review Problem 3. Show that for all $n > 4$, $2^n < n!$. Hint: Use induction.

Solution. For $n = 5$, this is obvious. Assume $n \geq 5$ and $2^n < n!$. We prove $2^{n+1} < (n+1)!$. Since $n \geq 5$, we have

$$\begin{aligned} (n+1)! &= n \cdot n! \\ &> 2 \cdot n! \\ &> 2 \cdot 2^n \\ &= 2^{n+1}. \end{aligned}$$

This completes the induction and the proof.

Problem 1.

(A) Explain (in your own words):

(i) What is a decision problem? A decision problem is a problem that asks a question that can be answered "true" or "false."

(ii) What does it mean to say that a decision problem *belongs to NP*? It means that a solution to the problem can be verified in polynomial time.

(iii) What is the Halting Problem?. The Halting Problem is the question: Is there a Java program P that accepts as input a (normal) Java program R and an integer n, and that outputs 1 if R terminates normally when running on n, 0 otherwise.

(iv) What is a universal Java program? A universal Java program U is a Java program that accepts as input an integer n and any normal Java program P, with public method *method* having just one integer argument, and that, when run on P, n returns the value output produced by P when running on n.

(B) Why is BigInteger used as an argument for the method of a normal Java program? There must not be a limit on the data type size in order for our programs to be general enough. In particular, for the final unprovability argument, it is necessary to run SelfApp.apply to the encoding of SelfApp as an integer value – the only way such an encoding would fit into a numeric Java data type is if that type were a BigInteger.

(C) When you examine the code in `HaltingCalculator`, it seems obvious that the `halts` method will never be able to detect that an input program *fails* to terminate normally, unless the program happens to throw a runtime exception (if, however, the input program goes into an infinite loop, the `halts` method has no way to detect this). So, it should be obvious that we have failed to provide an algorithm that solves the Halting Problem. Why don't these observations provide us with a *proof* that there is no algorithmic solution to the Halting Problem? Even though `HaltingCalculator` fails, it is conceivable that some other program could work. We need to argue that any attempt to solve the Halting Problem will fail.

(D) Even though `HaltingCalculator` does not accurately detect whether an input program terminates normally on a given BigInteger input, nevertheless, the `UniversalProgram` class *does* work correctly. Explain why. `UniversalProgram`, when running on input P, n (where P is a normal Java program), is not required to output a value in those cases in which P does not terminate normally; in those cases, `UniversalProgram` will just repeat whatever behavior P performs.

Problem 4. You are given a solution T to a SubsetSum problem with a $S = \{s_0, s_1, \ldots, s_{n-1}\}$ and k some non-negative integer. (Recall that T is a solution if it is a subset of S the sum of whose elements is equal to k.) Suppose that s_{n-1} belongs to T. Is it necessarily true that the set $T - \{s_{n-1}\}$ is a solution to the SubsetSum problem with inputs S', k' where $S' = \{s_0, s_1, \ldots, s_{n-2}\}$ and $k' = k - s_{n-1}$? Explain.

Solution. This is correct. We must show that the sum of the elements of $T' = T - \{s_{n-1}\}$ is $k - s_{n-1}$. But the sum of the elements of T (which is the set $T' \cup \{s_{n-1}\}$) is k. Since $s_{n-1} \notin T'$, the sum of the elements of T' must be s_{n-1} less than the sum of the elements of T; that is the sum of the elements of T' is $k - s_{n-1}$. (Note that it is possible that the only element of T is s_{n-1}. In that case the sum of elements of T is s_{n-1} (so it must be that $k = s_{n-1}$). Then the sum of elements of $T' = T - \{s_{n-1}\}$, which is now empty, must be $k - k = 0$; since the sum of an empty set of integers is 0, this result is still correct.)