Lab 1

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

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

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

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

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

Problem 1. *Short Answer and Coding.* The questions for Problem 1 are mostly about the Halting Problem, and the discussion given in the slides for Lesson 1.

(A) Explain (in your own words):

- (i) What is a decision problem?
- (ii) What does it mean to say that a decision problem *belongs to NP*?
- (iii) What is the Halting Problem?.
- (iv) What is a universal Java program?

(B) Why is BigInteger used as an argument for the method of a normal Java program?

(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?

Problem 2. *GCD Algorithm.* Write a Java method `int gcd(int m, int n)` which accepts positive integer inputs m, n and outputs the greatest common divisor of m and n.

The rest of this lab is an exploration of the SubsetSum Decision Problem (mentioned in the slides) and possible algorithms to solve it. Here is the statement of the problem once again: You are given a set $S = \{s_0, s_1, \ldots, s_{n-1}\}$ of positive integers and a non-negative integer k; S and k are inputs to your algorithm. Your algorithm will return "true" if there is a subset T of S such that the sum of the elements of T is precisely k, "false," otherwise. Alternatively, when you discover a subset T that works, you can simply return T; and if no such T can be found, then return `null`. (This alternative approach provides a solution to the SubsetSum *Optimization Problem.*)

Problem 3. *Brute Force Solution.* Formulate your own procedure for solving the SubsetSum Problem. Think of it as a Java method `subsetsum` that accepts as input S, k, and outputs a subset T of S with the property that the sum of the s_i in T is k if such a T exists, or `null` if no such T can be found. (A non-null return value can be thought of as a return of "true" and a null return value signifies "false.") Implement your idea in Java code.

Problem 4. Suppose you have been given a solution T to an instance of the SubsetSum problem with $S = \{s_0, s_1, \ldots, s_{n-1}\}$ 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. *Hint.* The sum of an empty set of integers is (by convention) equal to 0.