Objectives: Proof Techniques (I)

Instruction

This is a written assignment and is worth 30% of the marks for individual assignments. Please type your answers or write in legible handwriting. Answers that are messy and hard to read will be deducted marks.

The assignment is due by March 11 (Friday), 11:59pm. **Please submit a digital copy of your answers in pdf on Canvas.**

In this assignment, there are questions that require you to write a proof. Before working on these questions, please read Section 4.3.3 on writing good proofs. You may also want to review the examples we did in class.

Part A) Problems and Proofs (6 points)

Rewrite the following claim as a universal proposition in symbolic form:

"The algorithm bruteForceSearch() is a linear-time algorithm".

and use the strategy of generalizing from generic particular to prove that the claim is correct. For a detailed explanation of the claim and related concepts in algorithm analysis, see the figure on the next page.

Part B) Proof by Cases (6 points)

Use the method of proof by cases to prove the following statement

$$\forall n \geq 3,\ n^2 - 3n + 3 \text{ is odd.}$$

Part C) Do Question 4.52 in the textbook (Section 4.3) (6 points)

Part D) Do Question 4.56 in the textbook (Section 4.3) (6 points)

Part E) (Proof by Induction) Do Question 5.1 in the textbook (Section 5.2) (6 points)

More Information on Part A.

The claim is about the **running time** of an algorithm described as a Java method in Figure 1.

The running time of an algorithm is measured by the number of "primitive steps" required to complete the execution of the algorithm. For our purpose, let's count each program statement in the Java code as one primitive step. An algorithm is said to be a linear-time algorithm if its running time for any input is no greater than the number of elements in the input multiplied by some constant (a value that doesn't depend on the number of elements in the input).

```
public int bruteForceSearch(int[] values, int value){
    int numOfSteps = 0;
    boolean found = false;

    while( !found &&  numOfSteps < values.size){
      numOfSteps++;
      if(values[numOfSteps] == value) found = true;
    }

    return  numOfSteps;
}
```

Figure 1